METODE NUMERICE: Tema #1
Identificarea obiectelor în imagini

Termen de predare: 3 aprilie 2016

Titulari curs: *Florin Pop, George Popescu*

Responsabili: Darius Neațu, Denisa Sandu, Radu Stochițoiu, Alexandru Țifrea, Radu Vișan

Obiectivele Temei

În urma parcurgerii acestei teme studentul va fi capabil să:

- utilizeze metode de transformare și interpolare noi.
- rezolve un task folosind informații noi.

Task 1

Această problemă acoperă două subiecte: interpolarea și aplicarea de transformări pe imagini. În fișierul sursă se găsesc template-uri pentru toate funcțiile pe care trebuie să le implementați(nu sunteți limitați la folosirea acelor funcții).

1.1 Interpolare

Vom începe problema prin implementarea celui mai simplu model de interpolare, interpolarea liniară. Presupunem că avem un set de perechi (x, y) uniform distribuite spațial: $(1, 0.5), (2, 0.2), (3, 0.6), (4, 0.9)$. În interpolarea liniară, valoarea unui punct aflat într două puncte consecutive este dată de valoarea punctului pe dreapta care conectează cele două puncte consecutive, după cum este exemplificat și în Fugure 1. Prima voastră sarcină este să completați funcția lerp, care primește un vector unidimensional și o valoare x, și întoarce valoare estimată de funcție prin interpolare liniară in punctul x. În cazul în care x este in afara intervalului, se întoarce 0.

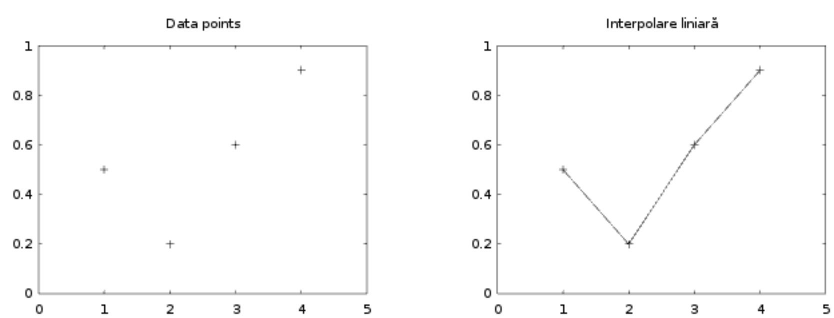


Figure 1: Exemplu de interpolare liniară

Următorul pas este implementarea unei interpolări liniare pentru matrici, adică interpolare biliniara. În acest caz, vrem să calculăm valoarea funcției intr-un punct (x, y) știind valoarea funcției în cei mai apropiați

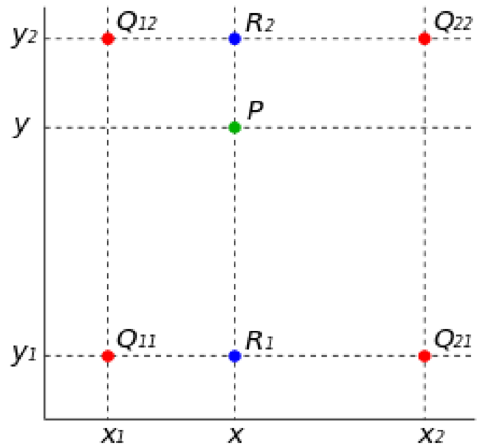


Figure 2: Interpolare biliniară

patru vecini, ca în Figure 2. Pentru a calcula valoarea funcției în punctele (x, y), se poate aplica funcția de interpolare liniara de mai multe ori: prima dată se realizează interpolarea pe direcția x (pentru a obține valorile cu albastru) iar după aceea se realizează intrepolarea pe direcția y (pentru a obține valoarea verde).

Interpolarea biliniară se va implementa in bilerp.m. Funcția primește o matrice (adică o imagine) și pozițiile (x, y) pentru care vrem sa obținem valoarea.

1.2 Transformarea imaginilor

Pentru a realiza o transformare 2D pe o imagine, trebuie să aplicăm o funcție pe fiecare pixel din imagine, $f(x, y) = (x', y')$, care ne va spune pozițiile la care trebuie sa mutăm fiecare pixel pentru a realiza transformarea asupra imaginii. Transformarile sunt reprezentate de o matrice:

$$T = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Pentru a realiza transformarea asupra unui pixel care se află la poziția (x, y), aplicam transformarea T:

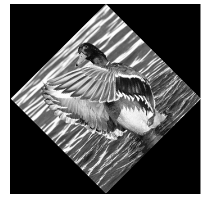
$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

Exemple de transformări:

$T = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix}$ Rotește imaginea cu α.

T = $\begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix}$ Scaleaza matricea cu s.

(a) Rața inițială



(b) Rața rotită

Figure 3: Figura (b) a fost realizată cu transformarea $T = \begin{bmatrix} cos(45) & -sin(45) \\ sin(45) & cos(45) \end{bmatrix}$

(a) Rața inițială

(b) Rața rotită

Figure 4: Figura (b) a fost realizată cu transformarea $T = \begin{bmatrix} 0.4 & 0 \\ 0 & 0.4 \end{bmatrix}$

1.2.1 Forward Mapping

Primul task pentru transformarea imaginilor este să implementați algoritmul de forward mapping in forward_mapping.m. Algoritmul forward mapping folosește o matrice de transformare, T, pentru a modifica pozitiile pixelilor in imagine, cum este descris mai sus. În implementarea voastră, trebuie să aveți în vedere faptul ca imaginea își poate schimba dimensiunea după ce a fost aplicată transformarea(un exemplu de imagine greșit scalată este prezentat in imaginea de mai jos).
Funcția primește o imagine sub forma unei matrici și matricea de transformare. Pe baza matricei întoarse de către funcție se va crea noua imagine. Pentru a a crea o imagine dintr-o matrice in matlab se recomanda folosirea functie:

imwrite(mat2gray(image_matrix), 'file.png')

- image_matrix este matricea pe care dorim să o transformăm în imagine.

Deoarece înmulțirile între pozițiile pixelilor și matricea de transformare nu dau valori întregi, veți fii nevoiți să folosiți funcția round pentru a aproxima noile poziții ale pixelilor.

Este normal ca dupa aplicarea algoritmului forward mapping să aveți pixeli morți(negri) în imagine.

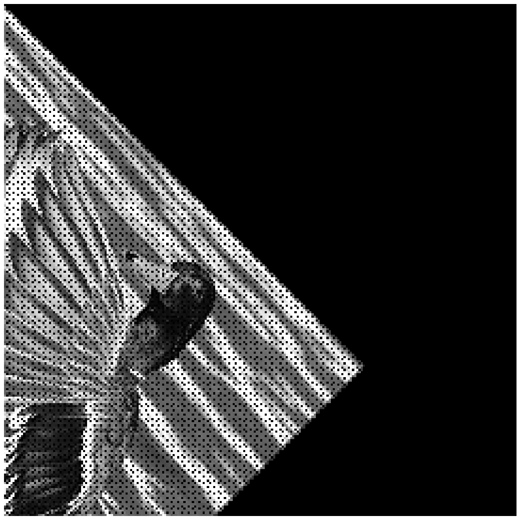


Figure 5: Imagine nescalată

1.2.1 Inverse Mapping

După cum se poate observa în imaginea de mai sus, forward mapping realizează transformările dorite asupra imaginii, dar ne lasă cu destul de mulți pixeli morți. Am putea să rezolvăm problema pixelilor morți prin interpolare, folosind o variantă modificată a bilerp, dar aceasta variantă nu este fezabilă deoarece pot să existe mai mulți pixeli morți adiacenți.

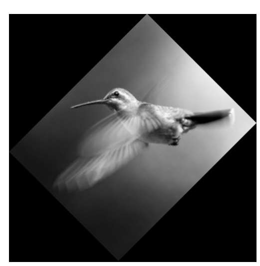
Pentru a rezolva această problemă, putem sa abordăm problema complet invers.

$$T^{-1}T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$
$$T^{-1} = 1/(ad - bc) * \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Putem să începem de la matricea la care vrem sa ajungem, și folosim T^{-1} pentru a calcula poziția fiecărui pixel în matricea inițială. Doarece rezultatele obținute nu sunt întregi, veți fii nevoiți să folosiți funcția bilerp pentru a calcula intensitatea interpolată a pixelului.

Completați functia inverse_mapping.m cu algoritmul inverse mapping. Funcția primește aceeași parametrii ca și forward_mapping și întoarce aceeași matrice, doar că matricea intoarsă nu va mai avea pixeli morți. Nu uitați să porniți de la o matrice scalată.

(a) Imaginea inițială



(b) Imaginea rotită

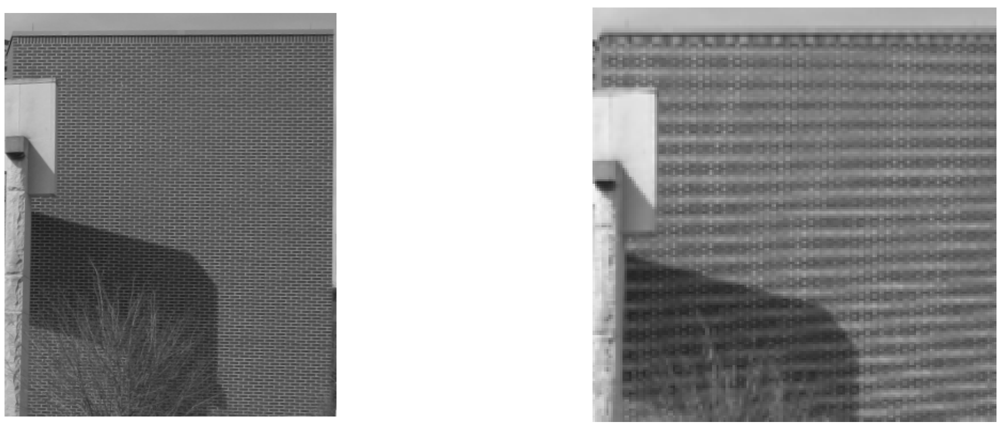
Figure 6: Transformare corecta a unei figuri

Pentru testare, trebuie sa implementați o funcție demo in care să testați funcția voastră inverse_mapping pe imaginile flapping_bird.png si flapping_duck.png. Funcția demo trebuie sa testeze cel puțin transformarile:

- Rotire față de orizontală
- Scalare cu 0.4
- Rotație cu 45 de grade în sens trigonometric.

Trebuie să menționați în README care sunt transformările, T, necesare pentru testare.

1.3 Anti-aliasing



(a) (b)

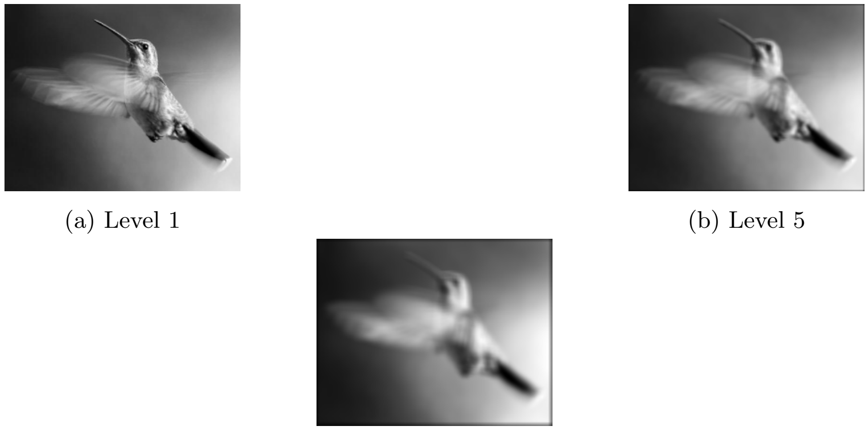
Figure 7: Exemplu de aliasing. (a) Imaginea originală și (b) imagine scalată cu 0.4 cu aliasing evident după zoom.

Metoda pe care am folosit-o până acum, nu este foarte buna la micșorarea imaginilor(downsampling), deoarece apare fenomenul numit aliasing. Spre exemplu, dacă încercăm sa scalăm o imagine cu 0.4,figura 7 b, este evident aliasing-ul în liniile albe de pe perete. Această problema apare doarece fiecarui pixel din imaginea scalată îi corespund aproximativ 2.8 pixeli din imaginea inițială. Rezolvarea acestei probleme se numește anti-aliasing. Vom rezolva această problema folosind o structură de date în care vom construi variante mânjite(blurred) ale imaginii inițiale.. Vom folosi un image stack, adică o matrice 3D. Image stack-ul nostru va avea dimensiunea $row * col * num\_levels$, adica avea num_levels imagini derivate din imaginea inițială. Prima imagine din stack este imaginea inițială, iar imaginile de pe nivelurile mai înalte vor avea un grad mai ridicat de mânjire(blur), ca în figura de mai jos.

Prima voastră sarcină la acest task este să implementați funcția image_stack.m care primește imaginea inițială și numărul de niveluri pe care trebuie să le adaugăm în stack și întoarce matricea 3D care conține image stack-ul. Matricea de pe primul nivel reprezintă matricea inițială, iar pe alte niveluri vor fi matrici cu un grad mai mare de mânjire(blur) față de matricele de pe nivelul precedent. Pentru a realiza funcția de blurring se recomandă folosirea functiei $conv2$ și folosirea unei funcții de blur:

$$K = \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\1 & 1 & 1\\1 & 1 & 1\end{bmatrix}$$

După implementarea acestei funcții, putem să facem un downsample cu un factor de k folosind matricea de pe nivelul k din stack.



(a) Level 1

(b) Level 5

(c) Level 10

Figure 8: Trei imagini dintr-un image stack.

Pentru a face un downsample cu un factor de 2.8 trebuie să realizăm o interpolare între matricea din stack de pe nivelul 2 și matricea din stack de pe nivelul 3. Pentru a rezolva aceasta problema, trebuie sa implementați funcția trilerp care realizează o interpolare triliniară, adica fiecare punct are opt vecini din care trebuie sa obținem noua valoare.

Detalii de implementare și redactare

Tema de casă va implementa funcțiile menționate la fiecare cerință în parte. Pentru implementarea temei puteţi folosi și alte funcţii definite de voi, dar cele menţionate mai sus sunt obligatorii. Trebuie să ţineţi cont de următoarele aspecte:

- codul sursă va conţine comentarii semnificative şi sugestive cu privire la implementarea algoritmilor;
- existenţa unui fişier `readme.txt` care va prezenta detaliile legate de implementarea şi testarea temei; de asemenea, la task-urile care cer explicarea anumitor aspecte, acestea trebuie să fie conținute în fisierul `readme.txt`
- fişierele care compun tema de casa vor fi incluse într-o arhivă `.zip` care respectă specificaţiile din regulamentul cursului;
- tema se va implementa în Octave şi va fi testată în mediul Octave.
- pentru o implementare corectă se vor putea obține maximum 90 de puncte și 10 puncte vor fi acordate pe README.
- pentru a obține punctajul pentru Task 3 va trebui ca modelul descris de parametrii w determinați de voi să aibă acuratețe mai mare decât 80% (acuratețea este măsura de evaluare descrisă la Task 4).