Explicatii ale functiilor folosite

In realizarea proiectului au fost folosite urmatoarele functii:

Modulul de criptare/ decriptare

- void Dimensiuni_img(char *cale_imagine,int *w,int *h)

-returneaza prin parametrii w si h latimea, respectiv inaltimea unei imagini (in pixeli)

- void Secret_Key(unsigned int *R0,unsigned int *SV)

-returneaza prin parametrii R0 si SV cheia secreta data de fisierul secret_key.txt

- void Header(char *cale_imagine,char *cale_header)

-copiaza headerul imaginii transmise prin “cale_imagine” in fisierul binar transmis prin “cale_header” (fisierul “header.bin”)

- int *Liniarizare(char *cale_imagine)

  -returneaza adresa de inceput a vectorului ce reprezinta forma liniarizata a imaginii transmise prin “cale_imagine”

  -fiecare element al vectorului L reprezinta un pixel al imaginii; ordinea pixelilor in vector difera de cea a citirii lor, astfel incat pozitia unui pixel de coordonate i si j (i=linia, j=coloana) este h*(w-j-1)+i (h=inaltimea, w=latimea); am notat cu n=h*(w-j-1) si am crescut n la fiecare pas i

  - in liniarizarea imaginii s-a tinut cont de octetii de padding

- void Salvare_memExt(char *cale_imagine,int *L,char *header,int w,int h)

  -salveaza in memoria externa imaginea transmisa prin “cale_imagine”, a carei forma liniarizata este data de vectorul L; headerul imaginii este dat de fisierul binar transmis prin “header”; dimensiunile imaginii sunt w (latimea) si h (inaltimea); w si h ar fi putut fi aflate din header, dar le-am transmis ca parametri pentru a nu face operatii suplimentare in cadrul functiei

- in incarcarea imaginii in memoria externa s-a tinut cont de octetii de padding
(cu valoarea 0)

- unsigned int XORSHIFT32(unsigned int R0)

-returneaza un numar pseudoaleator, pornind de la valoarea data de R0 (secventa de 2*w*h-1 numere se genereaza in main)

- unsigned int *Durstenfeld(int w,int h,unsigned int *R)

-returneaza adresa de inceput a permutarii (tablou unidimensional) obtinute prin aplicarea algoritmului lui Durstenfeld

- void Criptare (char *imag_init,char *imag_cript,char *sec_k)

-cripteaza imaginea data de “imag_init” si o salveaza in memoria externa (imag_cript), urmand pasii algoritmului de criptare; calea fisierului ce contine cheia secreta este transmisa prin “sec_k”

- void Decriptare(char *imag_cript,char *imag_decript,char *sec_k)

-decripteaza imaginea data de “imag_cript” si o salveaza in memoria externa
(imag_decript), urmand pasii algoritmului de decriptare

-inversa permutarii generate prin algoritmul lui Durstenfeld se obtine prin formula: permutare_inversa[permutare[i]]=i, unde i este pozitia unui element din permutare

- void chi_patrat (char *cale_imag)

-calculeaza si afiseaza valoarea testului chi patrat pentru o imagine transmisa prin “cale_imag”, pentru fiecare canal RGB

-se initializeaza trei vectori de frecventa corespunzatori celor trei canale RGB, in care se va memora numarul de aparitii ale unei valori i (0<=i<=255; valoarea i reprezinta intensitatea unui pixel pe un canal RGB), dupa care se calculeaza conform formulei valorile testului chi patrat

Modulul de recunoastere de patternuri

- void Dimensiuni_img(char *cale_imagine,int *w,int *h)
- int max (int a,int b), int min (int a,int b)

-returneza maximul, respectiv minimul dintre doua numere (aceste functii vor fi folosite ulterior in cadrul functiei de eliminare a non-maximelor, mai exact in calculul suprapunerii a doua detectii, pentru aflarea ariei intersectiei lor)

- void Copy_Image(char *imagine,char *copie)

-creeaza o copie a imaginii transmise prin “imagine” in “copie”

(am preferat sa folosesc o copie a imaginii initiale , desi nu era neaparat necesar, pentru a urmari colorarea ferestrelor pentru toate cele 10 sabloane, colorarea facandu-se dupa gasirea detectiilor unui sablon; daca as fi lucrat de fiecare data pe imaginea initiala, aceasta ar fi continut si ferestrele colorate cu sablonul precedent, i.e. imaginea initiala s-ar fi modificat o data cu apelarea functiei pentru un anumit sablon, ceea ce ar fi dus la obtinerea unor detectii incorecte; ulterior, colorarea imaginii se va doar la final, adica dupa gasirea tuturor detectiilor si eliminarea non-maximelor, motiv pentru care crearea unei copii a imaginii initiale nu este neaparat necesara)

- void grayscale_image(char* nume_fisier_sursa,char* nume_fisier_destinatie)

-transforma o imagine color intr-o imagine grayscale

- unsigned char ** Matrice(char *imagine)

-pune pixelii unei imagini intr-o matrice de tip unsigned char, in ordinea citirii lor; matricea este de tip unsigned char deoarece, in urma apelarii functiei grayscale_image, toti cei trei octeti ai unui pixel vor avea aceeasi valoare, cuprinsa intre 0 si 255. Prin atribuirea *fread(&x,1,3,f); L[i][j]=x,* pe pozitiile i si j din matrice se va memora valoarea unui singur octet

- float Corelatia(char *imagine,char *sablon,int l,int c,int ws,int hs,int wi,int hi,unsigned char**I,unsigned char**S)

-calculeaza corelatia dintre sablonul curent si continutul corespunzator al imaginii dat de fereastra (l, c); ws si hs sunt dimensiunile sablonului, iar wi

si hi sunt dimensiunile imaginii; I si S reprezinta matricea de pixeli a imaginii, respectiv a sablonului

- void Colorare_imag(char *imag,int l,int c,int ws,int hs,int wi,int hi,unsigned char *color)

-coloreaza in imaginea “imag” conturul ferestrei date de coordonatele l si c (unde l este linia, iar c este coloana corespunzatoare pixelului in imagine); ws si hs sunt dimensiunile sablonului (implicit ale ferestrei, intrucat am implemenat varianta “mai putin eleganta”, i.e. am considerat doar pozitiile l si c pentru care sablonul incape in imagine), wi si hi sunt dimensiunile imaginii mari, iar color este culoarea conturului ferestrei

-modificarile sunt facute direct pe imagine, motiv pentru care aceasta a fost deschisa in modul “rb+”

-conturul ferestrei reprezinta un dreptunghi. Am folosit patru foruri pentru fiecare linie in ordinea: orizontala jos, orizontala sus, verticala stanga, verticala dreapta, parcurgand imaginea cu ajutorul functiei fseek

- Corelatii* template_matching(char *imagine,char *sablon,float prag,int *n)

-implementeaza operatia de template matching dintre o imagine si un sablon, furnizand ferestrele care au corelatia mai mare decat pragul “prag”

-returneaza adresa de inceput a vectorului ce contine aceste ferestre, vectorul fiind de tip Corelatii *; un element de tip Corelatii retine coordonatele unui pixel in imagine (linie, coloana), valoarea corelatiei dintre un sablon si fereastra data de aceste coordonate si culoarea conturului ferestrei (pe fiecare canal RGB)

-retine prin parametrul n numarul de ferestre gasite, adica dimensiunea vectorului

- int compDescrescator(const void*a,const void *b)

-compara corelatiile a doua elemente, returnand o valoare pozitiva daca primul element are corelatia mai mica, una negativa daca primul

element are corelatia mai mare si 0 daca cele doua elemente au corelatii egale

-functia va fi folosita pentru ordonarea descrescatoare a tabloului de detectii, mai exact in apelarea functiei **qsort**

- Corelatii * Sortare_detectii(Corelatii *D,int m)

-sorteaza descrescator tabloul de detectii D, ce contine m elemente, cu ajutorul functiei qsort (in ordinea descrescatoare a detectiilor) si furnizeaza adresa de inceput a tabloului sortat

- float Suprapunere(Corelatii x,Corelatii y,int ws,int hs)

-calculeaza si returneaza valoarea suprapunerii dintre doua detectii x si y

-deoarece am implementat varianta “mai putin eleganta”, aria celor doua detectii este egala (ws*hs, unde ws si hs sunt latimea, respectiv inaltimea ferestrelor)

- Corelatii *Non_Maxime(Corelatii *D,int *m,int ws,int hs)

-implementeaza algoritmul de eliminare a non-maximelor: sorteaza tabloul de detectii apeland functia Sortare_detectii, calculeaza suprapunerea dintre doua detectii D[i] si D[j], cu i<j si, implicit, D[i].val_corelatie>D[j].val_corelatie (cu ajutorul functiei Suprapunere) si elimina detectiile D[j] pentru care suprapunerea este mai mare de 0.20

-returneaza adresa de inceput a tabloului in urma implementarii acestui algoritm