Дотаток А

Код програмної реалізації псевдоголографічного кодування

attack

```
function distorted_image = attack(image, attack_type, param)
switch attack_type
    case 'gaussian'
        noise_mean = param(1);
        noise_var = param(2);
        distorted_image = imnoise(image, 'gaussian', noise_mean,
noise_var);
    case 'salt & pepper'
        noise_density = param(1);
        distorted_image = imnoise(image, 'salt & pepper',
noise_density);
    case 'rotation'
        angle = param(1);
        distorted_image = imrotate(image, angle, 'crop');
    case 'hole'
        distorted_image = image;
        w = param(1)/2;
        distorted_image(end/2+(-w+1:w),end/2+(-w+1:w)) = 0;
    case 'jpeg'
        quality = param(1);
        imwrite(image,'dwm.jpg', 'Quality', quality)
        distorted_image = imread('dwm.jpg');
    otherwise
        distorted_image = image;
end
```

imlog

```
function imlog(im, descr)
global resPath
global num

if isempty(resPath)||isempty(num)||isempty(im)
    return;
end

name = inputname(1);
name = sprintf('%s%04i_%s_%s.png', resPath, num, name, descr);
num = num+1;

if isa(im, 'matlab.ui.Figure')
    saveas(im, name)
else
    imwrite(im, name)
```

```
end
```

perm

```
function permutated_image = perm(image, cond)

sz = numel(image);

rng(17)
ind = randperm(sz);

switch cond
    case +1
        permutated_image = image(ind);
    case -1
        permutated_image(ind) = image;
    otherwise
        permutated_image = image;
end

permutated_image = reshape(permutated_image, size(image));
```

test_haar1

```
clc
clear
close all

cond = 1;
w = 16;

host_image = imread('../_src/host/cameraman.tif');

qr = imread('../_src/qr/nure.png');
qr = imresize(qr(:,:,1)>0, 1/6, 'nearest');
qr = imresize(qr, w, 'nearest');

watermark = qr;

% encode
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % %

watermark = perm(watermark, cond);

host_image = imresize(host_image(:,:,1), 2*size(watermark));
[LL, HL, LH, HH] = dwt2(host_image, 'haar');

alpha = 0.25;
watermarked_image = idwt2(LL.*(alpha*watermark+1), HL, LH, HH,
'haar');
watermarked_image = uint8(watermarked_image);

% attacks
```

```
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % %

% Gaussian white noise with noise mean = noise_mean and variance
of 0.01
noise_mean = 0.15;
watermarked_image = imnoise(watermarked_image, 'gaussian',
noise_mean);

% salt and pepper noise with noise density = noise_density
noise_density = 0.15;
watermarked_image = imnoise(watermarked_image, 'salt & pepper',
noise_density);

angle = 4;
watermarked_image = imrotate(watermarked_image, angle,'crop');
angle = [];
for n=1:100
    angle(n) = -5+0.1*n;
    temp = imrotate(watermarked_image, angle(n),'crop');
    [L, HL, LH, HH] = dwt2(temp, 'haar');

    watermark = L-LL;

    watermark = perm(watermark, -cond);
    [y,x] = gradient(watermark);
    s(n) = mean2(y.^2+x.^2);
    n
end
figure, plot(angle, s), xlabel('angle'), ylabel('s'), grid on

[~,n] = min(s);
watermarked_image = imrotate(watermarked_image,angle(n),'crop');

watermarked_image(end/2+(-end/8:end/8),end/2+(-end/8:end/8)) =
0;

imwrite(watermarked_image,'dwm.jpg', 'Quality', 10)
watermarked_image = imread('dwm.jpg');

figure, imshow(watermarked_image)

% decode
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % %

[L, HL, LH, HH] = dwt2(watermarked_image, 'haar');

extracted_watermark = L-LL;

extracted_watermark  = perm(extracted_watermark, -cond);

extracted_watermark(extracted_watermark<0) = 0;
```

```
extracted_watermark = extracted_watermark  >
mean2(extracted_watermark );
extracted_watermark = uint8(255*extracted_watermark);

figure, imshow(extracted_watermark)


% post process
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % %

proc = @(x) mode(x);
fun = @(block_struct) repmat(proc(block_struct.data(:)),
size(block_struct.data));
extracted_watermark = blockproc(extracted_watermark, [w w],
fun);
figure, imshow(extracted_watermark)

figure, imshow(xor(extracted_watermark,qr))
```

test_haar2

```
clc
clear variables
close all

global resPath
global num

resPath = '../_dst/';
num = 1;

rng('default');

cond = 1;
w = 16;

host_image = imread('../_src/host/cameraman.tif');

qr = imread('../_src/qr/nure.png');
qr = imresize(qr(:,:,1)>0, 1/6, 'nearest');
qr = imresize(qr, w, 'nearest');

imlog(qr, '');

watermark = qr;

% encode
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % %

watermark = perm(watermark, cond);
imlog(watermark, '');
```

```
host_image = imresize(host_image(:,:,1), 2*size(watermark));
imlog(host_image, '');

wave = 'db1';
[LL, HL, LH, HH] = dwt2(host_image, wave);

alpha = 0.1;
watermarked_image = idwt2(LL.*(alpha*watermark+1), HL, LH, HH,
wave);
watermarked_image = uint8(watermarked_image);
imlog(watermarked_image, '');


% attacks
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % %
watermarked_image = attack(watermarked_image, 'gaussian', 7.5);


figure, imshow(watermarked_image)

% synchronize
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % %
krnl = fspecial('gaussian', 3, 15);
original = imfilter(host_image, krnl, 'symmetric');
distorted = imfilter(watermarked_image, krnl, 'symmetric');


ptsOriginal  = detectORBFeatures(original);
ptsDistorted = detectORBFeatures(distorted);
[featuresOriginal,validPtsOriginal] = ...
    extractFeatures(original,ptsOriginal);
[featuresDistorted,validPtsDistorted] = ...
    extractFeatures(distorted,ptsDistorted);

index_pairs = matchFeatures(featuresOriginal,featuresDistorted);
matchedPtsOriginal  = validPtsOriginal(index_pairs(:,1));
matchedPtsDistorted = validPtsDistorted(index_pairs(:,2));

s = [];
for t=1:1
[tform{t},inlierPtsDistorted,inlierPtsOriginal] = ...

estimateGeometricTransform(matchedPtsDistorted,matchedPtsOrigina
l,...
    'affine', 'MaxDistance', .75, 'Confidence', 99);
s(t) = length(inlierPtsDistorted);
end
[~,t] = max(s);

figure;
showMatchedFeatures(original,distorted,...
```

```
    inlierPtsOriginal,inlierPtsDistorted);

angle = rad2deg(atan(tform{t}.T(2,1)/tform{t}.T(1,1)));
watermarked_image = imrotate(watermarked_image, angle, 'crop');
figure; imshow(watermarked_image);

% outputView = imref2d(size(original));
% watermarked_image =
imwarp(watermarked_image,tform{t},'cubic','OutputView',outputVie
w);
% figure; imshow(watermarked_image);

% decode
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % %

[L, HL, LH, HH] = dwt2(watermarked_image, wave);

extracted_watermark = L-LL;

extracted_watermark  = perm(extracted_watermark, -cond);

extracted_watermark(extracted_watermark<0) = 0;
extracted_watermark = extracted_watermark  >
mean2(extracted_watermark );
extracted_watermark = uint8(255*extracted_watermark);

% figure, imshow(extracted_watermark)


% post-process
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % %

proc = @(x) mode(x);
fun = @(block_struct) repmat(proc(block_struct.data(:)),
size(block_struct.data));
extracted_watermark1 = blockproc(extracted_watermark, [w w],
fun);

proc = @(x) mean(x);
fun = @(block_struct) repmat(proc(block_struct.data(:)),
size(block_struct.data));
extracted_watermark2 = blockproc(im2double(extracted_watermark),
[w w], fun);

% figure, imshow(extracted_watermark2)

extracted_watermark2 = extracted_watermark2 >
graythresh(extracted_watermark2);
extracted_watermark2 = im2uint8(extracted_watermark2);
% figure, imshow([extracted_watermark1 extracted_watermark2])
```

```
diff1 = xor(extracted_watermark1,qr);
diff2 = xor(extracted_watermark2,qr);

% figure, imshow(diff1)
% figure, imshow(diff2)
```

test_haar3gauss

```
clc
clear variables
close all

global resPath
global num

resPath = '../_dst/';
num = 1;

rng('default');

cond = 1;
w = 16;

host_image = imread('../_src/host/cameraman.tif');

qr = imread('../_src/qr/nure.png');
qr = imresize(qr(:,:,1)>0, 1/6, 'nearest');
qr = imresize(qr, w, 'nearest');

imlog(qr, '');

watermark = qr;

% encode
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % %

watermark = perm(watermark, cond);
imlog(watermark, '');

host_image = imresize(host_image(:,:,1), 2*size(watermark));
imlog(host_image, '');

wave = 'db1';
[LL, HL, LH, HH] = dwt2(host_image, wave);

alpha = 0.1;
watermarked_image = idwt2(LL.*(alpha*watermark+1), HL, LH, HH,
wave);
watermarked_image = uint8(watermarked_image);
imlog(watermarked_image, '');
```

```
% attacks
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % %
gaussian_mean = 0:0.01:0.5;
gaussian_var = 0:0.001:0.05;

% for n=21
%     for m=26

for n=1:length(gaussian_mean)
    for m=1:length(gaussian_var)
        distorted_image = attack(watermarked_image, ...
            'gaussian', [gaussian_mean(n) gaussian_var(m)]);

        if n==21 && m==26
            imlog(distorted_image, '');
        end

        % decode
        % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % % % %

        [L, HL, LH, HH] = dwt2(distorted_image, wave);

        extracted_watermark = L-LL;
        extracted_watermark(extracted_watermark<0) = 0;
        extracted_watermark = mat2gray(extracted_watermark);

        if n==21 && m==26
            imlog(extracted_watermark, '');
        end

        extracted_watermark  = perm(extracted_watermark, -cond);
        if n==21 && m==26
            imlog(extracted_watermark, '');
        end


        extracted_watermark0 = extracted_watermark >
mean2(extracted_watermark );
        extracted_watermark0 = uint8(255*extracted_watermark0);
        if n==21 && m==26
            imlog(extracted_watermark0, '');
        end

        % post-process
        % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % % % %

        % % % % % % % % % % % % % %
        proc = @(x) mode(x);
        fun = @(block_struct) repmat(proc(block_struct.data(:)),
size(block_struct.data));
```

```
        extracted_watermark1 = blockproc(extracted_watermark0,
[w w], fun);
        if n==21 && m==26
            imlog(extracted_watermark1, '');
        end

        % % % % % % % % % % % % %
        proc = @(x) mean(x);
        fun = @(block_struct) repmat(proc(block_struct.data(:)),
size(block_struct.data));
        extracted_watermark2 =
blockproc(im2double(extracted_watermark0), [w w], fun);
        if n==21 && m==26
            imlog(extracted_watermark2, '');
        end

        extracted_watermark2 = extracted_watermark2 >
graythresh(extracted_watermark2);
        extracted_watermark2 = im2uint8(extracted_watermark2);
        if n==21 && m==26
            imlog(extracted_watermark2, '');
        end

        % % % % % % % % % % % % %
        extracted_watermark3 =
blockproc(im2double(extracted_watermark), [w w], fun);
        if n==21 && m==26
            imlog(extracted_watermark3, '');
        end

        extracted_watermark3 = extracted_watermark3 >
graythresh(extracted_watermark3);
        extracted_watermark3 = im2uint8(extracted_watermark3);
        if n==21 && m==26
            imlog(extracted_watermark3, '');
        end


        diff1 = xor(extracted_watermark1,qr);
        if n==21 && m==26
            imlog(diff1, '');
        end
        diff2 = xor(extracted_watermark2,qr);
        if n==21 && m==26
            imlog(diff2, '');
        end
        diff3 = xor(extracted_watermark3,qr);
        if n==21 && m==26
            imlog(diff3, '');
        end

        err1(n,m) = sum(sum(diff1));
        err2(n,m) = sum(sum(diff2));
```

```
        err3(n,m) = sum(sum(diff3));
    end
    n
end

figure, pcolor(gaussian_var,gaussian_mean,log(err1+1)),
xlabel('var'), ylabel('mean'), title('method #1')
colorbar;
imlog(gcf, 'err1');

figure, pcolor(gaussian_var,gaussian_mean,log(err2+1)),
xlabel('var'), ylabel('mean'), title('method #2')
colorbar;
imlog(gcf, 'err2');

figure, pcolor(gaussian_var,gaussian_mean,log(err3+1)),
xlabel('var'), ylabel('mean'), title('method #3')
colorbar;
imlog(gcf, 'err3');
```

test_haar3hole

```
clc
clear variables
close all

global resPath
global num

resPath = '../_dst/';
delete([resPath '*.*'])

num = 200;

rng('default');

cond = 1;
w = 16;

host_image = imread('../_src/host/cameraman.tif');

qr = imread('../_src/qr/nure.png');
qr = imresize(qr(:,:,1)>0, 1/6, 'nearest');
qr = imresize(qr, w, 'nearest');

imlog(qr, '');

watermark = qr;

% encode
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
```

```
watermark = perm(watermark, cond);
imlog(watermark, '');

host_image = imresize(host_image(:,:,1), 2*size(watermark));
imlog(host_image, '');

wave = 'db1';
[LL, HL, LH, HH] = dwt2(host_image, wave);

alpha = 0.1;
watermarked_image = idwt2(LL.*(alpha*watermark+1), HL, LH, HH,
wave);
watermarked_image = uint8(watermarked_image);
imlog(watermarked_image, '');


% attacks
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % %
a = 0:50:900;
n0 = 16;

for n=1:length(a)
    distorted_image = attack(watermarked_image, ...
        'hole', [a(n)]);

    if n==n0
        imlog(distorted_image, '');
    end

    % decode
    % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % %

    [L, HL, LH, HH] = dwt2(distorted_image, wave);

    extracted_watermark = L-LL;
    extracted_watermark(extracted_watermark<0) = 0;
    extracted_watermark = mat2gray(extracted_watermark);

    if n==n0
        imlog(extracted_watermark, '');
    end

    extracted_watermark  = perm(extracted_watermark, -cond);
    if n==n0
        imlog(extracted_watermark, '');
    end

    extracted_watermark0 = extracted_watermark >
mean2(extracted_watermark );
    extracted_watermark0 = uint8(255*extracted_watermark0);
    if n==n0
```

```
        imlog(extracted_watermark0, '');
    end

    % post-process
    % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % %

    % % % % % % % % % % % % % %
    proc = @(x) mode(x);
    fun = @(block_struct) repmat(proc(block_struct.data(:)),
size(block_struct.data));
    extracted_watermark1 = blockproc(extracted_watermark0, [w
w], fun);
    if n==n0
        imlog(extracted_watermark1, '');
    end

    % % % % % % % % % % % % % %
    proc = @(x) mean(x);
    fun = @(block_struct) repmat(proc(block_struct.data(:)),
size(block_struct.data));
    extracted_watermark2 =
blockproc(im2double(extracted_watermark0), [w w], fun);
    if n==n0
        imlog(extracted_watermark2, '');
    end

    extracted_watermark2 = extracted_watermark2 >
graythresh(extracted_watermark2);
    extracted_watermark2 = im2uint8(extracted_watermark2);
    if n==n0
        imlog(extracted_watermark2, '');
    end

    % % % % % % % % % % % % % %
    extracted_watermark3 =
blockproc(im2double(extracted_watermark), [w w], fun);
    extracted_watermark3 = mat2gray(extracted_watermark3);
    if n==n0
        imlog(extracted_watermark3, '');
    end

    extracted_watermark3 = extracted_watermark3 >
graythresh(extracted_watermark3);
    extracted_watermark3 = im2uint8(extracted_watermark3);
    if n==n0
        imlog(extracted_watermark3, '');
    end


    diff1 = xor(extracted_watermark1,qr);
    if n==n0
        imlog(diff1, '');
```

```
    end
    diff2 = xor(extracted_watermark2,qr);
    if n==n0
        imlog(diff2, '');
    end
    diff3 = xor(extracted_watermark3,qr);
    if n==n0
        imlog(diff3, '');
    end

    err1(n) = sum(sum(diff1));
    err2(n) = sum(sum(diff2));
    err3(n) = sum(sum(diff3));

    n
end

figure, plot(a, err1, '-r',...
    a, err2, '-g', ...
    a, err3, '-b'), grid on
legend('method #1','method #2','method #3')
xlabel('size'), ylabel('# errors'),
axis([0 900 -1 100])
imlog(gcf, 'err');
```

test_haar3jpeg

```
clc
clear variables
close all

global resPath
global num

resPath = '../_dst/';
delete([resPath '*.*'])

num = 200;

rng('default');

cond = 1;
w = 16;

host_image = imread('../_src/host/cameraman.tif');

qr = imread('../_src/qr/nure.png');
qr = imresize(qr(:,:,1)>0, 1/6, 'nearest');
qr = imresize(qr, w, 'nearest');

imlog(qr, '');

watermark = qr;
```

```
% encode
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % %

watermark = perm(watermark, cond);
imlog(watermark, '');

host_image = imresize(host_image(:,:,1), 2*size(watermark));
imlog(host_image, '');

wave = 'db1';
[LL, HL, LH, HH] = dwt2(host_image, wave);

alpha = 0.1;
watermarked_image = idwt2(LL.*(alpha*watermark+1), HL, LH, HH,
wave);
watermarked_image = uint8(watermarked_image);
imlog(watermarked_image, '');


% attacks
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % %
quality = 1:100;
n0 = 9;

for n=1:length(quality)
    distorted_image = attack(watermarked_image, ...
        'jpeg', [quality(n)]);

    if n==n0
        imlog(distorted_image, '');
    end

    % decode
    % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % %

    [L, HL, LH, HH] = dwt2(distorted_image, wave);

    extracted_watermark = L-LL;
    extracted_watermark(extracted_watermark<0) = 0;
    extracted_watermark = mat2gray(extracted_watermark);

    if n==n0
        imlog(extracted_watermark, '');
    end

    extracted_watermark  = perm(extracted_watermark, -cond);
    if n==n0
        imlog(extracted_watermark, '');
    end
```

```
    extracted_watermark0 = extracted_watermark >
mean2(extracted_watermark );
    extracted_watermark0 = uint8(255*extracted_watermark0);
    if n==n0
        imlog(extracted_watermark0, '');
    end

    % post-process
    % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % %

    % % % % % % % % % % % % % %
    proc = @(x) mode(x);
    fun = @(block_struct) repmat(proc(block_struct.data(:)),
size(block_struct.data));
    extracted_watermark1 = blockproc(extracted_watermark0, [w
w], fun);
    if n==n0
        imlog(extracted_watermark1, '');
    end

    % % % % % % % % % % % % % %
    proc = @(x) mean(x);
    fun = @(block_struct) repmat(proc(block_struct.data(:)),
size(block_struct.data));
    extracted_watermark2 =
blockproc(im2double(extracted_watermark0), [w w], fun);
    if n==n0
        imlog(extracted_watermark2, '');
    end

    extracted_watermark2 = extracted_watermark2 >
graythresh(extracted_watermark2);
    extracted_watermark2 = im2uint8(extracted_watermark2);
    if n==n0
        imlog(extracted_watermark2, '');
    end

    % % % % % % % % % % % % % %
    extracted_watermark3 =
blockproc(im2double(extracted_watermark), [w w], fun);
    extracted_watermark3 = mat2gray(extracted_watermark3);
    if n==n0
        imlog(extracted_watermark3, '');
    end

    extracted_watermark3 = extracted_watermark3 >
graythresh(extracted_watermark3);
    extracted_watermark3 = im2uint8(extracted_watermark3);
    if n==n0
        imlog(extracted_watermark3, '');
    end
```

```
    diff1 = xor(extracted_watermark1,qr);
    if n==n0
        imlog(diff1, '');
    end
    diff2 = xor(extracted_watermark2,qr);
    if n==n0
        imlog(diff2, '');
    end
    diff3 = xor(extracted_watermark3,qr);
    if n==n0
        imlog(diff3, '');
    end

    err1(n) = sum(sum(diff1));
    err2(n) = sum(sum(diff2));
    err3(n) = sum(sum(diff3));

    n
end

figure, plot(quality, err1, '-r',...
    quality, err2, '-g', ...
    quality, err3, '-b'), grid on
legend('method #1','method #2','method #3')
xlabel('quality'), ylabel('# errors'),
% axis([0 900 -1 100])
imlog(gcf, 'err');
```

test_haar3rotation

```
clc
clear variables
close all

global resPath
global num

resPath = '../_dst/';
delete([resPath '*.*'])

num = 200;

rng('default');

cond = 1;
w = 16;

host_image = imread('../_src/host/cameraman.tif');

qr = imread('../_src/qr/nure.png');
qr = imresize(qr(:,:,1)>0, 1/6, 'nearest');
qr = imresize(qr, w, 'nearest');
```

```
imlog(qr, '');

watermark = qr;

% encode
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % %

watermark = perm(watermark, cond);
imlog(watermark, '');

host_image = imresize(host_image(:,:,1), 2*size(watermark));
imlog(host_image, '');

wave = 'db1';
[LL, HL, LH, HH] = dwt2(host_image, wave);

alpha = 0.1;
watermarked_image = idwt2(LL.*(alpha*watermark+1), HL, LH, HH,
wave);
watermarked_image = uint8(watermarked_image);
imlog(watermarked_image, '');


% attacks
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % %
angle = -2:0.1:2;
n0 = 23;

for n=1:length(angle)
    distorted_image = attack(watermarked_image, ...
        'rotation', [angle(n)]);
    if n==n0
        imlog(distorted_image, '');
    end

% % %     % synchronize
% % %     % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % % % % %
% % %     krnl = fspecial('gaussian', 3, 15);
% % %     blurred_host = imfilter(host_image, krnl,
'symmetric');
% % %     if n==n0
% % %         imlog(blurred_host, '');
% % %     end
% % %
% % %     blurred_distorted = imfilter(distorted_image, krnl,
'symmetric');
% % %     if n==n0
% % %         imlog(blurred_distorted, '');
% % %     end
```

```
% % %
% % %     ptsOriginal  = detectORBFeatures(blurred_host);
% % %     ptsDistorted = detectORBFeatures(blurred_distorted);
% % %     [featuresOriginal,validPtsOriginal] = ...
% % %         extractFeatures(blurred_host,ptsOriginal);
% % %     [featuresDistorted,validPtsDistorted] = ...
% % %         extractFeatures(blurred_distorted,ptsDistorted);
% % %
% % %     index_pairs =
matchFeatures(featuresOriginal,featuresDistorted);
% % %     matchedPtsOriginal  =
validPtsOriginal(index_pairs(:,1));
% % %     matchedPtsDistorted =
validPtsDistorted(index_pairs(:,2));
% % %
% % %     [tform,inlierPtsDistorted,inlierPtsOriginal] = ...
% % %
estimateGeometricTransform(matchedPtsDistorted,matchedPtsOrigina
l,...
% % %         'affine', 'MaxDistance', .75, 'Confidence', 99);
% % %
% % %     if n==n0
% % %         figure,
showMatchedFeatures(blurred_host,blurred_distorted,...
% % %             inlierPtsOriginal,inlierPtsDistorted);
% % %         imlog(gcf, 'showMatchedFeatures');
% % %     end
% % %
% % %     estimated_angle =
rad2deg(atan(tform.T(2,1)/tform.T(1,1)));
% % %     distorted_image = imrotate(distorted_image,
estimated_angle, 'crop');
% % %     if n==n0
% % %         imlog(distorted_image, '');
% % %     end

    % decode
    % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % %

    [L, HL, LH, HH] = dwt2(distorted_image, wave);

    extracted_watermark = L-LL;
    extracted_watermark(extracted_watermark<0) = 0;
    extracted_watermark = mat2gray(extracted_watermark);

    if n==n0
        imlog(extracted_watermark, '');
    end

    extracted_watermark  = perm(extracted_watermark, -cond);
    if n==n0
        imlog(extracted_watermark, '');
```

```
    end

    extracted_watermark0 = extracted_watermark >
mean2(extracted_watermark );
    extracted_watermark0 = uint8(255*extracted_watermark0);
    if n==n0
        imlog(extracted_watermark0, '');
    end

    % post-process
    % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % %

    % % % % % % % % % % % % % %
    proc = @(x) mode(x);
    fun = @(block_struct) repmat(proc(block_struct.data(:)),
size(block_struct.data));
    extracted_watermark1 = blockproc(extracted_watermark0, [w
w], fun);
    if n==n0
        imlog(extracted_watermark1, '');
    end

    % % % % % % % % % % % % % %
    proc = @(x) mean(x);
    fun = @(block_struct) repmat(proc(block_struct.data(:)),
size(block_struct.data));
    extracted_watermark2 =
blockproc(im2double(extracted_watermark0), [w w], fun);
    if n==n0
        imlog(extracted_watermark2, '');
    end

    extracted_watermark2 = extracted_watermark2 >
graythresh(extracted_watermark2);
    extracted_watermark2 = im2uint8(extracted_watermark2);
    if n==n0
        imlog(extracted_watermark2, '');
    end

    % % % % % % % % % % % % % %
    extracted_watermark3 =
blockproc(im2double(extracted_watermark), [w w], fun);
    extracted_watermark3 = mat2gray(extracted_watermark3);
    if n==n0
        imlog(extracted_watermark3, '');
    end

    extracted_watermark3 = extracted_watermark3 >
graythresh(extracted_watermark3);
    extracted_watermark3 = im2uint8(extracted_watermark3);
    if n==n0
        imlog(extracted_watermark3, '');
```

```
    end


    diff1 = xor(extracted_watermark1,qr);
    if n==n0
        imlog(diff1, '');
    end
    diff2 = xor(extracted_watermark2,qr);
    if n==n0
        imlog(diff2, '');
    end
    diff3 = xor(extracted_watermark3,qr);
    if n==n0
        imlog(diff3, '');
    end

    err1(n) = sum(sum(diff1));
    err2(n) = sum(sum(diff2));
    err3(n) = sum(sum(diff3));

    n
end

figure, plot(angle, err1, '-r',...
    angle, err2, '-g', ...
    angle, err3, '-b'), grid on
legend('method #1','method #2','method #3')
xlabel('angle'), ylabel('# errors'),
imlog(gcf, 'err');
```

test_haar3salt

```
clc
clear variables
close all

global resPath
global num

resPath = '../_dst/2/';
num = 100;

rng('default');

cond = 1;
w = 16;

host_image = imread('../_src/host/cameraman.tif');

qr = imread('../_src/qr/nure.png');
qr = imresize(qr(:,:,1)>0, 1/6, 'nearest');
qr = imresize(qr, w, 'nearest');
```

```
imlog(qr, '');

watermark = qr;

% encode
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % %

watermark = perm(watermark, cond);
imlog(watermark, '');

host_image = imresize(host_image(:,:,1), 2*size(watermark));
imlog(host_image, '');

wave = 'db1';
[LL, HL, LH, HH] = dwt2(host_image, wave);

alpha = 0.1;
watermarked_image = idwt2(LL.*(alpha*watermark+1), HL, LH, HH,
wave);
watermarked_image = uint8(watermarked_image);
imlog(watermarked_image, '');


% attacks
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % %
noise_density = 0:0.01:0.5;
n0 = 16;

for n=1:length(noise_density)
    distorted_image = attack(watermarked_image, ...
        'salt & pepper', [noise_density(n)]);

    if n==n0
        imlog(distorted_image, '');
    end

    % decode
    % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % % %

    [L, HL, LH, HH] = dwt2(distorted_image, wave);

    extracted_watermark = L-LL;
    extracted_watermark(extracted_watermark<0) = 0;
    extracted_watermark = mat2gray(extracted_watermark);

    if n==n0
        imlog(extracted_watermark, '');
    end

    extracted_watermark  = perm(extracted_watermark, -cond);
```

```
    if n==n0
        imlog(extracted_watermark, '');
    end


    extracted_watermark0 = extracted_watermark >
mean2(extracted_watermark );
    extracted_watermark0 = uint8(255*extracted_watermark0);
    if n==n0
        imlog(extracted_watermark0, '');
    end

    % post-process
    % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% % % % %

    % % % % % % % % % % % % %
    proc = @(x) mode(x);
    fun = @(block_struct) repmat(proc(block_struct.data(:)),
size(block_struct.data));
    extracted_watermark1 = blockproc(extracted_watermark0, [w
w], fun);
    if n==n0
        imlog(extracted_watermark1, '');
    end

    % % % % % % % % % % % % %
    proc = @(x) mean(x);
    fun = @(block_struct) repmat(proc(block_struct.data(:)),
size(block_struct.data));
    extracted_watermark2 =
blockproc(im2double(extracted_watermark0), [w w], fun);
    if n==n0
        imlog(extracted_watermark2, '');
    end

    extracted_watermark2 = extracted_watermark2 >
graythresh(extracted_watermark2);
    extracted_watermark2 = im2uint8(extracted_watermark2);
    if n==n0
        imlog(extracted_watermark2, '');
    end

    % % % % % % % % % % % % %
    extracted_watermark3 =
blockproc(im2double(extracted_watermark), [w w], fun);
    extracted_watermark3 = mat2gray(extracted_watermark3);
    if n==n0
        imlog(extracted_watermark3, '');
    end

    extracted_watermark3 = extracted_watermark3 >
graythresh(extracted_watermark3);
```

```
    extracted_watermark3 = im2uint8(extracted_watermark3);
    if n==n0
        imlog(extracted_watermark3, '');
    end


    diff1 = xor(extracted_watermark1,qr);
    if n==n0
        imlog(diff1, '');
    end
    diff2 = xor(extracted_watermark2,qr);
    if n==n0
        imlog(diff2, '');
    end
    diff3 = xor(extracted_watermark3,qr);
    if n==n0
        imlog(diff3, '');
    end

    err1(n) = sum(sum(diff1));
    err2(n) = sum(sum(diff2));
    err3(n) = sum(sum(diff3));

    n
end

figure, plot(noise_density, err1, '-r',...
    noise_density, err2, '-g', ...
    noise_density, err3, '-b'), grid on
legend('method #1','method #2','method #3')
xlabel('density'), ylabel('# errors'),
imlog(gcf, 'err');
```

Дотаток Б

Код програмної реалізації хаотичних карт на предмет забезпечення стійкості цифрового водяного знака

```
from PIL import Image
import numpy as np
import os
from matplotlib.pyplot import imshow
import matplotlib.pyplot as plt
import cv2
import random
from math import sqrt
from math import log
np.random.seed(1000)
x =np.random.randint(0, 199, 1024)
y =np.random.randint(0, 199, 1024)
image = "nure1"
ext = ".jpg"
img = cv2.imread(image + ext)[:, :, ::-1]
plt.figure()
plt.imshow(img, plt.cm.gray)
plt.title('Original image')
#imshow(np.asarray(pil_im))

plt.figure(figsize=(8,4.4))
plt.rcParams['font.size'] = '15'
plt.rcParams['font.family'] = 'Times New Roman'
histogram_blue = cv2.calcHist([img],[0],None,[256],[0,256])
plt.plot(histogram_blue, color='blue')
histogram_green = cv2.calcHist([img],[1],None,[256],[0,256])
plt.plot(histogram_green, color='green')
histogram_red = cv2.calcHist([img],[2],None,[256],[0,256])
plt.plot(histogram_red, color='red')
plt.title('',  fontname='Times New Roman',fontsize=14)
plt.xlabel('Values of pixels', fontname='Times New Roman')
plt.ylabel('Number of pixels', fontname='Times New Roman')
current_values = plt.gca().get_yticks()
plt.gca().set_yticklabels(['{:,.0f}'.format(x) for x in
current_values])
plt.savefig("3. Ориг. зображення Nure_en.pdf", dpi=600,
format="pdf")
#plt.savefig("3. Ориг. зображення Nure.raw", dpi=300,
format="raw")
plt.show()
def getImageMatrix_gray (img):
    rows, cols, ch = img.shape
    img_gr = np.zeros([rows, cols])
    for row in range(0, rows):
```

```
        for col in range(0, cols):
            img_gr[row][col]=((0.3 * img[row][col][0]) + (0.59 *
img[row][col][1]) + (0.11 * img[row][col][2]))
    return img_gr
def corr(samples_x, samples_y,n):
    samples_x=np.array(samples_x)
    samples_y=np.array(samples_y)
    x_=np.sum(samples_x)/len(samples_x)
    y_=np.sum(samples_y)/len(samples_y)
    xy_=np.sum(samples_x*samples_y)/len(samples_y)
    s2x=np.sum(samples_x**2)/len(samples_x)-x_**2
    s2y=np.sum(samples_y**2)/len(samples_y)-y_**2
    sx=sqrt(s2x)
    sy=sqrt(s2y)
    r=(xy_-x_*y_)/(sx*sy)
    return (r)
def corrDiag(ImageMatrix, font='15', tit='', fonts=15
,fig=(8,4), sav="1"):
    samples_x = []
    samples_y = []
    for i in range(1024 ):
        samples_x.append(ImageMatrix[x[i]][y[i]])
        samples_y.append(ImageMatrix[x[i]+1][y[i]])
    print (corr(samples_x, samples_y,1024))
    plt.figure(figsize=fig)
    plt.rcParams['font.size'] = font
    plt.rcParams['font.family'] = 'Times New Roman'
    plt.scatter(samples_x,samples_y,s=2)
    plt.title(tit,  fontname='Times New Roman',fontsize=fonts)
    plt.xlabel('Pixel 1', fontname='Times New Roman')
    plt.ylabel('Pixel 2', fontname='Times New Roman')
    plt.savefig(sav, dpi=600, format="pdf")
    plt.show()
Image = getImageMatrix_gray(img)
corrDiag(ImageMatrix=Image, tit='', font='15', fonts=15
,fig=(8,5), sav="4. Автокореляція сусідніх пікселів для
оригінального зображення_en.pdf" )
Перемішування зображення за допомогою карт кота Арнольда
Перемішування:
$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix}=\begin{pmatrix} 1 & P
\\ Q & PQ+1 \end{pmatrix}\begin{pmatrix} x_1 \\ y_1
\end{pmatrix} mod (N)   $$
Відновлення :
$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}=\begin{pmatrix} PQ+1
& -P \\ -Q & 1 \end{pmatrix}\begin{pmatrix} x_2 \\ y_2
\end{pmatrix} mod (N)   $$
def ArnoldCatTransform(im, mx):
    rows, cols, ch = im.shape
    n = rows

    img_arnold = np.zeros([rows, cols, ch])
    for x in range(0, rows):
            for y in range(0, cols):
```

```
                x_y = np.array([x,y], int)
                x_y2 = np.dot(mx,x_y)%n
                img_arnold[x_y2[0]][x_y2[1]] = im[x][y]
    return img_arnold
def EncArnoldCat (img, count):
    mx = np.array([[1, 3],
                   [1, 4]], int)
    encimg=np.copy(img)
    for x in range(0, count):
        encimg = ArnoldCatTransform(encimg, mx)
    return encimg.astype(int)
def DecArnoldCat (img, count):
    mx = np.array([[4, -3],
                   [-1, 1]], int)
    encimg=np.copy(img)
    for x in range(0, count):
        encimg = ArnoldCatTransform(encimg, mx)
    return encimg.astype(int)
plt.figure()
trans =EncArnoldCat (img, 44)
plt.imshow(trans, plt.cm.gray)
plt.title('Original image')
plt.savefig("5. NureEncArnoldCat.png", dpi=300)
     plt.figure(figsize=(8,4.4))
plt.rcParams['font.size'] = '15'
plt.rcParams['font.family'] = 'Times New Roman'
histogram_blue =
cv2.calcHist([trans.astype('uint8')],[0],None,[256],[0,256])
plt.plot(histogram_blue, color='blue')
histogram_green =
cv2.calcHist([trans.astype('uint8')],[1],None,[256],[0,256])
plt.plot(histogram_green, color='green')
histogram_red =
cv2.calcHist([trans.astype('uint8')],[2],None,[256],[0,256])
plt.plot(histogram_red, color='red')
plt.xlabel('Values of pixels', fontname='Times New Roman')
plt.ylabel('Number of pixels', fontname='Times New Roman')
current_values = plt.gca().get_yticks()
plt.gca().set_yticklabels(['{:,.0f}'.format(x) for x in
current_values])
plt.savefig("6. Зображення після перемішування за допомогою карт
кота Арнольда_en.pdf", dpi=600, format="pdf")
plt.show(
Image = getImageMatrix_gray(trans)
corrDiag(ImageMatrix=Image, tit='', font='15', fonts=15
,fig=(8,5), sav="7. Автокореляція Після перемішування за до
помогою карт кота Арнольда_en.pdf"
```

Дотаток В

Список публікацій здобувача

1. Бологова Н.М. Дослідження моделей та методів обробки зображень та шляхи вдосконалення технологій розпізнавання маркерів в системах доповненої реальності / Н.М. Бологова, І. В. Рубан // Сучасний стан наукових досліджень та технологій в промисловості. – 2019. – № 1 (7). – С. 25 –33 (Належить до категорії Б).

2. Ruban I. Method of sustainable detection of augmented reality markers by changing deconvolution / I. Ruban, N. Bolohova, V. Martovytskyi, V. Lebediev, N. Lukova-Chuiko // International Journal of Advanced Trends in Computer Science and Engineering. 2020. – № 9 (2). – P. 1113–1120.

3. Makoveichuk O. Development of a method for improving stability method of applying digital watermarks to digital images / O. Makoveichuk, I. Ruban, N. Bolohova, A. Kovalenko, V. Martovytskyi, T. Filimonchuk // Eastern-European Journal of Enterprise Technologies. 2021. – № 3 (2 (111)). – P. 45–56. (Належить до категорії А, входить до міжнародної наукометричної бази Scopus).

4. Ruban I. Digital image authentication model / I. Ruban, N. Bolohova, V. Martovytskyi, O Koptsev // Advanced Information Systems. 2021. – № 5 (1). – P. 113–117 (Належить до категорії Б).

5. Ruban I. Methodology for assessing the effectiveness of methods for embedding digital watermarks / I. Ruban, N. Bolohova, V. Martovytskyi, R. Yaroshevych // Advanced Information Systems. 2021. – № 5 (3). – P. 112–118 (Належить до категорії Б).

6. Martovytskyi V. Development of methods for generation of digital watermarks resistant to distortion / V. Martovytskyi, I. Ruban, N. Bolohova, O. Sievierinov, O. Zhurylo, O. Permiakov, A. Nosyk, D. Nepokrytov, I. Krylenko // Eastern-European Journal of Enterprise Technologies. 2021. – № 6

(2 (114)). – P. 103–116. (Належить до категорії А, входить до міжнародної наукометричної бази Scopus).

7. Ruban I. Information technology for confirming property rights to digital images / I. Ruban, N. Bolohova, V. Martovytskyi // Advanced Information Systems. 2022.– № 6 (1). – P. 118–123 (Належить до категорії Б).

8. Ruban I. et al. Method of neural network recognition of ground-based air objects //2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT). – IEEE, 2018. – С. 589-592.

9. Бологова Н.М. Аналіз сучасного підходу обробки зображень для розпізнавання маркерів в системі доповненої реальності / Н.М. Бологова, І.В. Рубан, К.Р. Локотецька // Тези доповідей шостої міжнародної науково-практичної конференції «Проблеми інформатизації». 14 – 16 листопада 2018 р., Черкаси, Баку, Бельско-Бяла, Харків. – 2018. – С.36.

10. Bolohova N. Analysis of restoration methods for optical-electronic images lubricated at motion / N. Bolohova, Y. Kortiak // Тези доповідей шостої міжнародної науково-практичної конференції «Проблеми інформатизації». 13 – 15 листопада 2019 р., Черкаси, Харків, Баку, Бельско-Бяла. – 2019. – С.8.

11. Bolohova N. Analysis of the current status of additional reality technologies / N. Bolohova, Y. Kortyak, A. Liashova // Proceedings of Fourth International Scientific and Technical Conference on COMPUTER AND INFORMATION SYSTEMS AND TECHNOLOGIES. – Kharkiv, April 22-23, 2020. – P.12–13.

12. Bolohova N. Method for evaluating the effectiveness of methods for embedding digital watermarks / N. Bolohova, V. Martovytskyi, V. Diachenko, O. Kolomiitsev, V. Fedorchenko // Матеріали XX міжнародної науково-практичної конференції «Інформаційні технології і безпека». Київ. – 2020. С. 75–83.

13. Пересада Р.А. Аналіз методів підвищення стійкості водяних знаків у цифрових зображеннях / Р.А. Пересада, Н.М. Бологова // Тези доповідей восьмої міжнародної науково-технічної

конференції «Проблеми інформатизації». 26 – 27 листопада 2020 р., . Черкаси, Харків, Баку, Бельсько-Бяла. – 2020. – С. 53.

14. Бологова Н.М. Модель автентифікації цифрового зображення / Н.М. Бологова // Тези доповідей десятої міжнародної науково-практичної конференції «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління». 8 – 9 квітня 2021р., – Баку, Харків, Жиліна. – С. 41.

Додаток Г

Акт впровадження

«ЗАТВЕРДЖУЮ»
Проректор з наукової роботи
Харківського національного
університету радіоелектроніки
_____ Юрій РОМАНЕНКОВ
14 листопада 2023 р.

АКТ
про використання результатів дисертаційної роботи на тему:
«Модель, методи та інформаційна технологія автентифікації цифрових зображень у прикладних системах користувача»»
здобувача кафедри електронних обчислювальних машин
Харківського національного університету радіоелектроніки
Бологової Наталії Миколаївни

Ми, що нижче підписалися, начальник Навчального відділу ХНУРЕ к.т.н., доц. Міхнова А.В., декан факультету Комп'ютерної інженерії та управління к.т.н., доц. Ляшенко О.С., завідувач кафедри Електронних обчислювальних машин д.т.н., проф. Коваленко А.А., склали цей акт про те, що результати дисертаційної роботи на здобуття ступеня доктора філософії Бологової Наталії Миколаївни реалізовано в навчальному процесі Харківського національного університету радіоелектроніки на кафедрі електронних обчислювальних машин, а саме: метод надійної перевірки справжності цифрового зображення з високим ступенем захисту при передачі інформації, та вдосконалено метод підвищення стійкості стегасистеми, впроваджені в навчальний процес на кафедрі електронних обчислювальних машин в курсовому та дипломному проектуванні, та в дисципліні «Системне програмне забезпечення» відповідно до освітньо-професійної програми «Комп'ютерна інженерія» для здобувачів першого (бакалаврського) рівня вищої освіти.

Начальник навчального відділу _____ Аліна МІХНОВА

Декан факультету Комп'ютерної інженерії та управління _____ Олексій ЛЯШЕНКО

Завідувач кафедри електронних обчислювальних машин _____ Андрій КОВАЛЕНКО

Додаток Д

Акт впровадження

ЗАТВЕРДЖУЮ

ТВО Директора
Товариства з обмеженою
відповідальністю
«ОБОРОННІ ТЕХНОЛОГІЇ»
_____ Василь МИХИДЕНКО
“ 25 ” 04 2023 р.

АКТ

впровадження результатів дисертаційної роботи

БОЛОГОВОЇ Наталії Миколаївни

Комісія у складі:

голови – *директора комерційного КУЛИКА Ю.В.*,

членів комісії: *керівника проектів, ктн ДАВИКОЗИ О.П., головного інженера проектів ЗАТРАВКІНА С.В.*, підтверджує, що наступні результати наукових досліджень БОЛОГОВОЇ Наталії Миколаївни, а саме:

- модель надійної перевірки справжності цифрового зображення з високим ступенем захисту від спотворення та підробки.

Використання зазначених результатів дозволяє здійснювати вбудову інформації в цифровий контент, в даному випадку цифрове зображення.

Таким чином, застосування запропонованої моделі дозволяє забезпечити справжність цифрового зображення на основі застосування прихованого водяного знаку.

Результати проведених випробувань підтвердили стійкість даної конструкції до геометричних і не геометричних атак.

Голова комісії:

Директор комерційний _____ Юрій КУЛИК

Члени комісії:

Керівник проектів _____ Олександр ДАВИКОЗА

Головний інженер проектів _____ Сергій ЗАТРАВКІН