# ДОДАТОК А

Графічний матеріал атестаційної роботи

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

# АТЕСТАЦІЙНА РОБОТА

НА ТЕМУ: **Методи підвищення стійкості водяних знаків у цифрових зображеннях**

ВИКОНАВ:
Студент гр. КСМм-19-1 Пересада Р.А.

КЕРІВНИК:
Ільїна І.В.

ХАРКІВ
2020р.

2

## МЕТА ТА ОБ'ЄКТ ДОСЛІДЖЕННЯ

**Метою атестаційної роботи** є дослідження статистичних властивостей цифрових зображень, що забезпечують найкращу стійкість стеганографічної системи, що використовує цифрові зображення в якості контейнерів для вбудовування секретного повідомлення.

**Об'єкт дослідження** - цифрова стеганографічна система на основі цифрових статичних зображень в якості контейнера.

**Предмет дослідження** - статистичні властивості цифрових зображень BMP формату.

## ВИРІШЕНІ ЗАВДАННЯ

3

1. Зробити короткий огляд методів вбудови інформації в просторові області цифрових зображень. Дати коротка характеристика існуючих методів.

2. Розглянути принцип дії LSB методу для стеганографічної системи на основі цифрових зображень. Сформовати загальний список вимог і критеріїв вибору зображень-контейнерів для алгоритмів стеганографічного приховування інформації на основі методу LSB.

3. Провести дослідження статистичних характеристик цифрових зображень BMP формату, зображення отримані за допомогою цифрового фотоапарата шляхом конвертації з RAW формату.

4. Провести аналіз дисперсії бітових площин.

5. Провести аналіз даних про ентропії молодших бітових площин

## Критерії ефективності стеганографічних алгоритмів зображень

4

- стеганосистема повинна мати прийнятну обчислювальну складність реалізації;
- заповнений контейнер повинен бути візуально однаковим з незаповненим;
- повинна забезпечуватися необхідна пропускна здатність (що особливо актуально для стеганосистем прихованої передачі даних);
- методи приховування повинні забезпечувати автентичність і цілісність секретної інформації для авторизованої особи;
- потенційний порушник має повне уявлення про стеганосистеми і деталі її реалізації, єдине, що йому невідомо, - це ключ, за допомогою якого тільки його власник може встановити факт наявності і зміст прихованого повідомлення;
- якщо факт існування прихованого повідомлення стає відомим порушнику, це не повинно дозволити останньому витягти його до тих пір, поки ключ зберігається в таємниці;
- порушник повинен бути позбавлений будь-яких технічних та інших переваг в розпізнанні або, по крайній мірі, розкритті змісту секретних повідомлень.

5

## Методи вбудовування в просторові області зображень

- Метод Kutter
- Метод Bruyndonckx
- Метод Langelaar
- Метод Pitas
- Метод Rongen
- Метод Patchwork
- Метод Bender

6

## Аналіз методу заміни найменш значущих біт

```
Pixels: (00100111 11101001 11001000)
(00100111 11001000 11101001)
(11001000 00100111 11101001)
A: 01000001
Result: (00100110 11101001 11001000)
(00100110 11001000 11101000)
(11001000 00100111 11101001) .
```

Приклад 1 – Приховане повідомлення

7

## Аналіз методу заміни найменш значущих біт

Переваги методу:

- розмір файлу-контейнера залишається незмінним;
- при заміні одного біта в каналі синього кольору впровадження неможливо помітити візуально;
- можливість варіювати пропускну здатність, змінюючи кількість замінних біт.

Недоліки методу:

- приховане повідомлення легко зруйнувати, наприклад, при стисненні або відображенні.
- не забезпечено таємність вбудовування інформації. Точно відомо місце розташування зашифрованої інформації. Для подолання цього недоліку можна вбудовувати інформацію не в усі пікселі зображення, а лише до деяких з них, що визначаються за псевдовипадковому закону відповідно до ключа, відомого тільки законному користувачеві. Пропускна здатність при цьому зменшується [7].

8

## Загальні критерії вибору контейнерів

Можливі такі варіанти контейнерів:

- контейнер генерується самою стегосистемою.
- контейнер вибирається з деякого множини контейнерів.
- контейнер надходить ззовні.

9

# Загальні критерії вибору контейнерів

Класифікація критеріїв вибору контейнера:

- відмова від загальновідомих зображень в якості контейнера, як, наприклад, зображення «Джоконда»;
- відмова від використання в якості контейнера зображень, конвертованих з JPEG-формату в формат BMP;
- отримання зображення за допомогою фотоапарата або сканера, а не за допомогою графічних редакторів;
- великий розмір контейнера;
- відсутність корисної складової на молодших бітових площинах зображення;
- зашумленность;
- відсутність плавних переходів і монотонних областей;
- «строкатість»;
- велике число перепадів яскравості;
- наявність великої кількості пікселів, відтінки кольорів яких погано розрізняються оком людини (зелений, жовтий).

10

# Атака на основі аналізу статистики Хі - квадрат

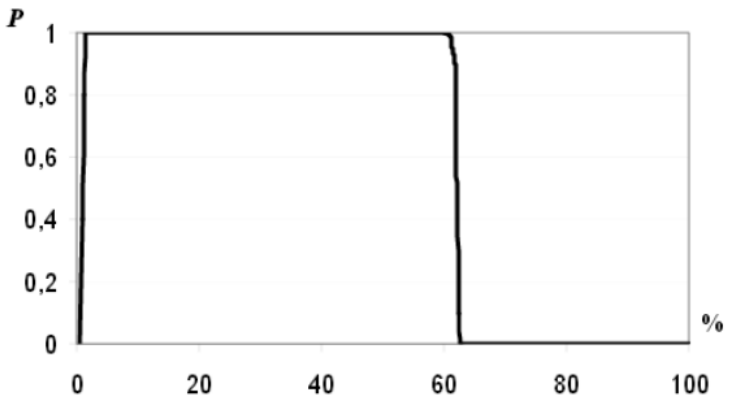



Рисунок 1 – Імовірність вбудови за критерієм Хі - квадрат при аналізі стегоконтейнера, отриманого методом послідовної заміни.

# RS-діаграма типового зображення

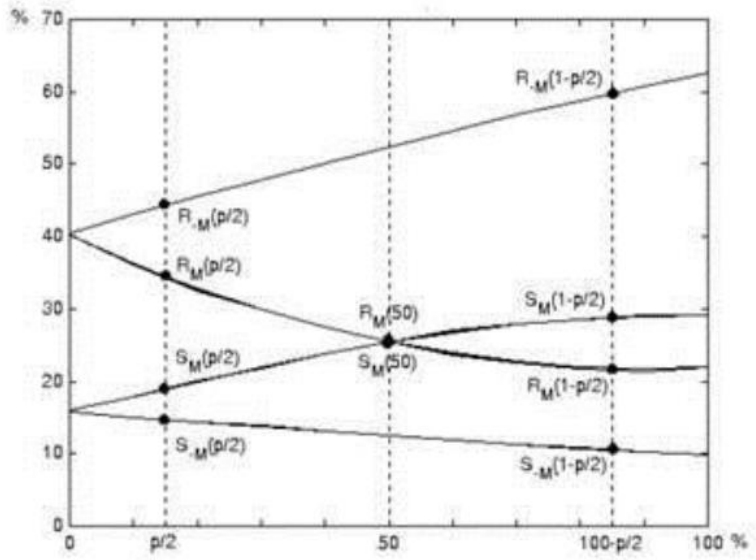


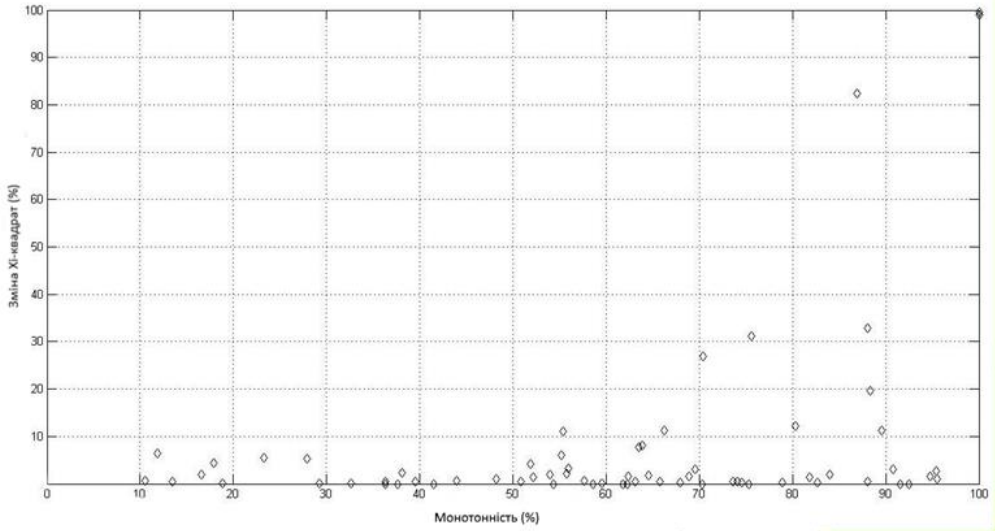

Рисунок 2 – RS-діаграма типового зображення

# Вимоги до зображень

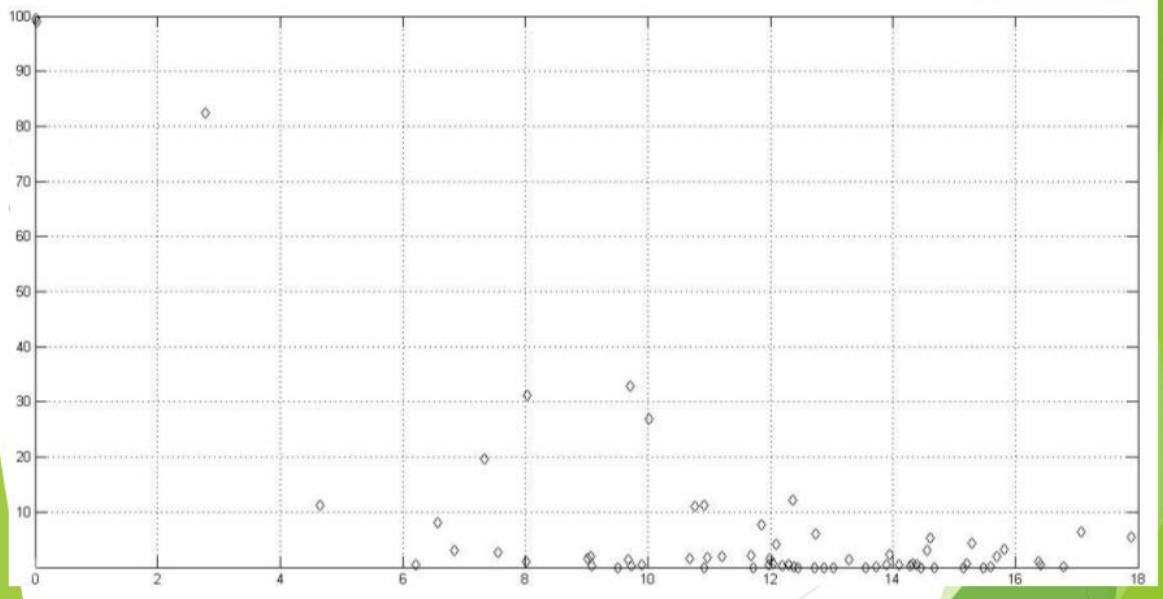
До всіх досліджуваних зображень пред'являлися наступні вимоги:

1. зображення повинні бути вихідним файлом, а не бути отриманими шляхом конвертації інших цифрових форматів зображень у формат BMP;
2. зображення не повинні бути створені з використанням будь-яких графічних редакторів;
3. всі зображення повинні мати однаковий розмір, що виключить вплив розміру зображення на отримані результати.

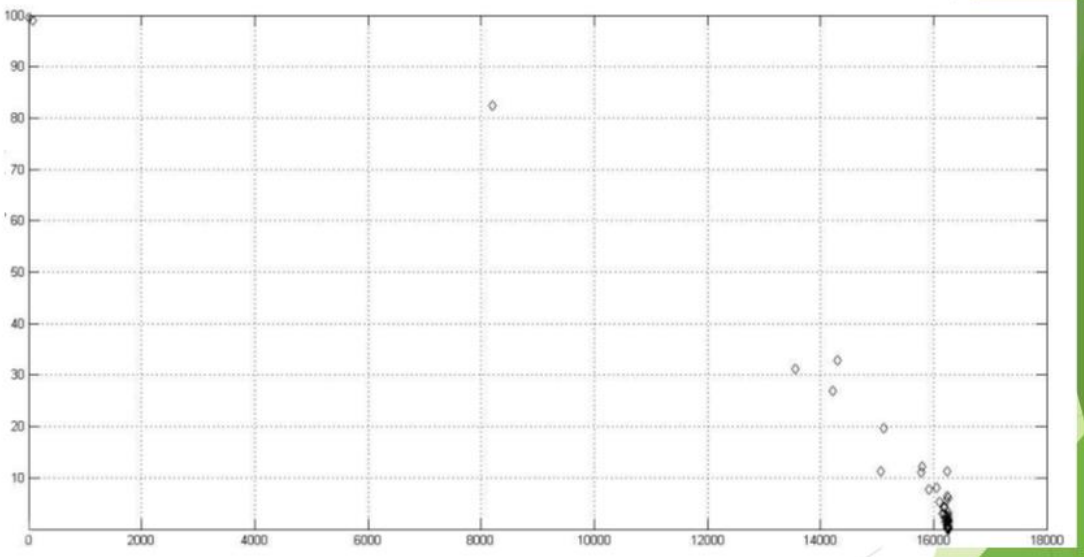
# Залежність зміни статистики Хі-квадрат від монотонності зображення




# Залежність зміни Хі-квадрат від ентропії молодшої бітової площини



# Залежність зміни Хі-квадрат від дисперсії молодшої бітової площини






# ВИСНОВКИ

Мета цієї роботи полягає у визначенні характеристик цифрових зображень-контейнерів, які забезпечують найбільшу стійкість стеганографічної системи. Для досягнення зазначеної мети, було вирішено ряд теоретичних і практичних завдань. В ході проведених досліджень отримані наступні основні результати.

1. Зроблено короткий огляд методів вбудови інформації в просторові області цифрових зображень. Дана коротка характеристика існуючих методів.

2. Розглянуто принцип дії LSB методу для стеганографічної системи на основі цифрових зображень. Сформовано загальний список вимог і критеріїв вибору зображень-контейнерів для алгоритмів стеганографічного приховування інформації на основі методу LSB. На основі сформульованих вимог додосліджуваних зображень-контейнерів обрані зображення формату BMP.

3. Проведено дослідження статистичних характеристик цифрових зображень BMP формату, зображення отримані за допомогою цифрового фотоапарата шляхом конвертації з RAW формату. Конвертація з RAW формату, не призначеного для безпосередньої візуалізації, в формат BMP відбувається без втрати якості зображення. В якості досліджуваних характеристик були обрані монотонність, ентропія і дисперсія зображень. Дослідження зміни статистичних характеристик зображень проведено методами оцінки критерію Хі-квадрат і RS методом стегоаналіза. На основі отриманих даних можна зробити висновок, що найбільшою стеганографічної стійкістю володіють зображень з найбільшою ентропією і дисперсією молодшої бітової площини, з підвищенням монотонності зображення стійкість стеганографічної системи падає, погіршується.

# ВИСНОВКИ



4. Аналіз дисперсії бітових площин показав, що молодші бітові площини тестованих зображення володіють досить великими близькими показниками дисперсії порядку 16000. Мінімальне значення дисперсії молодших бітових площин становить близько 6000, що відповідає ймовірності виявлення повідомлення методу RS аналізу в 60 відсотків. Зі збільшенням біта вбудовування збільшується розкид значень дисперсії досліджуваних зображень. RS метод стегоаналіза показує, що при показниках дисперсії прагнуть до 0, ймовірність виявлення прихованої інформації, що міститься в зображенні, прагне до 100 відсотків. Таким чином, наведені дані про дисперсії підтверджують теорію випадковості молодших біт і показують їх перевагу найбільш старшим бітам при встановленні інформації методом LSB. При виникненні необхідності задіяння старших біт при LSB стеганографії, слід більш уважно ставитися до вибору зображення в силу великих разбросів показників дисперсії різних зображень і віддавати переваги зображенням, що володіють найбільшою дисперсією в цих бітах.

5. Аналіз даних про ентропії показує, що її відносна величина молодших бітових площин є більшою у порівнянні зі старшими бітами. Максимальна відносне значення молодших бітових площин становить близько 18, в свою чергу максимальне значення ентропії другий бітової площини зменшилася більш ніж в два рази. Максимальна ймовірність виявлення впровадження повідомлення в молодший біт зображення становить близько 60 відсотків при значенні ентропії дорівнює 3. При залученні найбільш старших біт зображень, ймовірність виявлення в 60 відсотків досягається при відносному значенні ентропії близько 6. Необхідно відзначити, що зі збільшенням біта вбудовування, відносне значення ентропії бітових площин тестованих зображень прагне до максимального значення. Підводячи підсумок, можна зробити висновок, що, як і в випадку з дисперсією зображень, при виборі зображення в якості контейнера для стеганографічного приховування даних методом LSB, слід вибирати зображення з великими значеннями ентропії в бітових площинах.

# ДОДАТОК Б

## Лістинг вихідного коду програми

```
close all;
clear all;
clc;
pause(0.1);
%% config
bit = 1;
%% import lib
%%path('stego',path);
%%path('analyze',path);
imageFolder = 'images';
saveFile_old = 'RESULT3.mat’;
saveFile = 'RESULT4.mat';
forceBlackWhite = 0; % convert images to gray !
graph3d = 0; %% plot 3d graph instead of 2d
%% console output:
%% OX line:
%% 1-monotone
%% 2-dispersion
%% 3-entropy
%% OY line:
%% x – chi-square
%% r – regular group (RS analysis)
%% s – singular group (RS analysis)
graph = '123xrs';
%%
analyzeResult = [];
path = [pwd,'\',imageFolder,'\'];
%% scan folder
if (exist(saveFile,'file'))
%% load calculated data
disp('Previous calculation loaded); ’
load(saveFile,'analyzeResult');
else if(exist(saveFile_old,'file'))
%% load calculated data
disp(['Previous calculation loaded (', saveFile_old, ' now deprecated.
New calculations will be saved as ',saveFile, ')'] );
loadedData = load(saveFile_old,'analyzeResult');
analyzeResult_old = loadedData.analyzeResult;
analyzeResult = zeros(6, size(analyzeResult_old, 2));
analyzeResult(1,:) = analyzeResult_old(1,:)*100; %% immonot
analyzeResult(2,:) = analyzeResult_old(2,:); %% d
analyzeResult(3,:) = analyzeResult_old(3,:); %% ent
analyzeResult(4,:)            =            100*(analyzeResult_old(5,:)-
analyzeResult_old(4,:)); %% chi2 (take the difference)
analyzeResult(5,:)                                                    =
100*(analyzeResult_old(6,:)./analyzeResult_old(8,:));    %%    ps_reg    (the
difference)
analyzeResult(6,:)                                                    =
100*(analyzeResult_old(7,:)./analyzeResult_old(8,:));    %%    ps_sing    (the
difference)
%% v3: analyzeResult = [analyzeResult,[immonot; d; ent; chi2_orig;
chi2_enc; dR; dS; N]]; %#ok<AGROW>
%%   v4:   analyzeResult   =   [analyzeResult,[immonot*100;   d;   ent;
100*(chi2_enc-chi2_orig); 100*dR./N; 100*dS./N ]]; %#ok<AGROW>
else
```

```
      if(forceBlackWhite)
      disp(['Colored images will be converted to gray. It can save time']);
      else
      disp(['Working with colored images. Take a quite, it will take much more
time...']);
      end;
      %% calculate data
      filename_arr = dir(path);
      n = length(filename_arr);
      for i=1:n
      filedata = filename_arr(i);
      if filedata.isdir>0 continue; end;
      %% skip folder
      [pathstr,name,ext] = fileparts([filedata.name]);
      if(strcmp(ext,'.png') && strcmp(ext,'.bmp') ) continue; end;
      %% skip non-images
      disp('------------------------------');
      disp(['Loading    image:    ',    name,ext,    '    (',num2str(i),'    of
',num2str(n),')']);
      image = imread([path,'\',name,ext]);
      if(length(size(image))==3 && forceBlackWhite)
      %%make it black & white
      image = rgb2gray(image);
      end;
      %% resize image to make size equivalent
      %% image = imresize(image,[480 640]);
      imshow(image);
      disp(['Analyze image...']);
      pause(0.1);
      [immonot,    d,    ent,    chi2_orig,    chi2_enc,    dR,    dS,    N]    =
image_analyze_complex(image, bit);
      %% chi2_enc-chi2_orig (take the difference)
      %% dR, dS (the difference)
      analyzeResult  =  [analyzeResult,[immonot*100;  d;  ent;  100*(chi2_enc-
chi2_orig); 100*dR./N; 100*dS./N ]]; %#ok<AGROW>
      disp(['monotone: ',num2str(immonot), ' (', num2str( ceil(immonot*100)),
'%)']);
      disp(['dispersion: ', num2str(d)]);
      disp(['entropy: ', num2str(ent) ]);
      disp(['’and: ', num2str(chi2_orig), ' -> ', num2str(chi2_enc), ' (',
num2str( ceil(chi2_orig*100)), '% -> ', num2str( ceil(chi2_enc*100)), '%)']);
      disp(['regular   group   changes:   ',num2str(   dR   ),   '   (',num2str(
round(N\dR*10000)/100),'%)']);
      disp(['singular   group   changes:   ',num2str(   dS   ),   '   (',num2str(
round(N\dS*10000)/100),'%)']);
      pause(0.1);
      end; end;
      close all;
      %% save result
      save(saveFile,'analyzeResult');
      end;
      %% plot graphics
      disp('------------------------------');
      disp(' construct graphs...');
      if (graph3d)
      error('not implemented');
      else
      param_map = struct('f1', 1, 'f2', 2, 'f3', 3, 'x', 4, 'r', 5, 's', 6);
      param_label = struct('f1', 'Monotone (%)',...
      'f2', 'Dispersion',...
      'f3','Entropy',...
      'x', 'Chi-square changes (%)',...
      'r', 'regular group changes(%)',...
      's', ' regular group changes (%)');
```

```
param_lim = struct('f1', [0, 100],...
'x', [0, 100],...
'r', [0, 100],...
's', [0, 100]);
graph_style = 'd';
%% only for OY
param_color = struct( 'x', 'k',...
's', 'r');
pairs = l_pairs(graph);
for i=1:length(pairs)
%% parse config
graph_config = pairs(i,:);
ox_p = ['f', graph_config(1)]; %% digit
oy_p = graph_config(2); %% letter
%% verify reqirements
if(~isfield(param_map, ox_p))
error(['Invalid graph param: ',ox_p]);
end;
if(~isfield(param_map, oy_p))
error(['Invalid graph param: ',oy_p]);
end;
%% color
color = graph_style;
if(isfield(param_color, oy_p))
color = [color, getfield(param_color,oy_p)]; %#ok<AGROW,GFLD>
end;
%% get values
ox_values = analyzeResult( getfield(param_map,ox_p) ,:); %#ok<GFLD>
oy_values = analyzeResult( getfield(param_map,oy_p) ,:); %#ok<GFLD>
%% PLOT
figure();
plot(ox_values,oy_values, color);
grid on;
%% labels
if(isfield(param_label, ox_p))
xlabel( getfield(param_label,ox_p) ); %% #ok<GFLD>
end;
if(isfield(param_label, oy_p))
ylabel( getfield(param_label,oy_p) ); %% #ok<GFLD>
end;
% limits
if(isfield(param_lim, ox_p))
xlim( getfield(param_lim,ox_p) ); %% #ok<GFLD>
end;
if(isfield(param_lim, oy_p))
ylim( getfield(param_lim,oy_p) ); %% #ok<GFLD>
end;
end;
end
disp('Done');
%%
disp(['[', datestr(clock), ']',' Done!']);
function new_str = lorem_ipsum(str, charlength)
%% new_str = lorem_ipsum(str, charlength)
%% more description will be later...
new_str = '';
repeat = ceil(charlength/length(str));
for i=1:repeat
new_str = [new_str,str];
end;
%%cut unnecessary symbols
new_str = new_str(1:charlength);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function pairs = l_pairs(str)
%% pairs = l_pairs(str)
%% split string into pairs of letter+digit
pairs = [];
char_offset = 1;
while(1==1)
char_pos = lookup_char(str, char_offset);
if (char_pos>0)
digit_offset = 1;
while(1==1)
digit_pos = lookup_digit(str, digit_offset);
if(digit_pos>0)
%% char and digit found
pairs = [pairs; [str(digit_pos), str(char_pos)]];
digit_offset = digit_pos+1;
else
break;
end;
end;
char_offset = char_pos+1;
else
break;
end;
end;
end
function pos = lookup_char(str, offset)
for pos=offset:length(str)
letter = str(pos)*1-'0';
if(letter>=0 && letter<=9)
%% digit
else
%% char
return;
end;
end;
pos = -1;
end
%% induss code :)
function pos = lookup_digit(str, offset)
for pos=offset:length(str)
letter = str(pos)*1-'0';
if(letter>=0 && letter<=9)
%% digit
return;
else
%% char
end;
end;
pos = -1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [H] = imhist3(image)
%% H = imhist3(image)
%% imhist for colour images. return array of hist
%% s = size(image);
%% H = [];
%% for i=1:s(3)
%% H = [H, imhist(image(:,:,i))];
%% end;
if(length(size(image))~=3)
error('not implemented');
end;
H = int32(zeros(256,256,256));
for i=1:size(image,1)
```

```
      for j=1:size(image,2)
      pixel = image(i,j,:);
      H(pixel(1)+1,  pixel(2)+1,  pixel(3)+1  )  =  H(pixel(1)+1,  pixel(2)+1,
pixel(3)+1 ) + 1;
      end;
      end;
      %% if (nargout == 0)
      %% plot_result(x, y, map, isScaled, class(a), range);
      %% else
      %% yout = y;
      %% end
      end
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Исходный код image_ps.m:

```
      function [dR, dS, N] = image_ps(image, blocksize)
      %%%  function  [R_plus,  R_minus,  S_plus,  S_minus,  U_plus,  U_minus,
N_group] = image_ps(image, blocksize)
      %% [R_plus, R_minus, S_plus, S_minus, U_plus, U_minus] = image_ps(image,
blocksize)
      %% image - image
      %% R_plus, R_minus – regular groups
      %% S_plus, S_minus - — singular groups
      %% U_plus, U_minus – not used groups
      %% N_group – common number of groups
      %% blocksize - size of block. must divide on 2
      if(blocksize<=0 || mod(blocksize, 2))
      error('Block size must divide on 2 (like 2*n)');
      end;
      if(mod( size(image, 1), blocksize) ~=0)
      error(['You must block size aliquot to image width (n*blocksize=width).
Cannot  use  blocksize  ',  num2str(blocksize),  '  for  image  with  width  ',
num2str(size(image, 1))] );
      end;
      %% theoretical values (average)
      if(length(size(image))==2)
      [dR, dS, N] = image_ps_layer(image, blocksize);
      else
      res_r = zeros(1,3);
      res_s = zeros(1,3);
      [res_r(1), res_s(1), N] = image_ps_layer(image(:,:,1), blocksize); % R
      [res_r(2), res_s(2), N] = image_ps_layer(image(:,:,2), blocksize); % G
      [res_r(3), res_s(3), N] = image_ps_layer(image(:,:,3), blocksize); % B
      dR = sum(res_r);
      dS = sum(res_s);
      N = N*3;
      %p = p/3; % because 3 layer
      %histval = imhist3(image);
      end;
      end
      function [dR, dS, N] = image_ps_layer(image, blocksize)
      if(length(size(image))~=2)
      error(['You must pass only one layer (R, G or B) for colored images']);
      end;
      %% invert last bit
      image_one = invert_bit(image);
      %% invert +1
      image_minus_one = invert_bit_plus_one(image);
      %% select sum of changes
      %%              more:             http://network-journal.mpei.ac.ru/cgi-
bin/main.pl?l=ru&n=11&pa=13&ar=4
      s = size(image);
      L = floor( s(1)/blocksize); % count block in one row
      N = L*s(2); %result block count
```

```
      f_original = zeros(L, s(2) );
      f_one = zeros(L, s(2) );
      f_m_one = zeros(L, s(2) );
      for i=1:L
      diapason = (i-1)*blocksize+1:i*blocksize;
      %% SUM HERE. can be dispersion
      f_original(i,:) = sum( image(diapason, :),1);
      f_one(i,:) = sum( image_one(diapason, :),1);
      f_m_one(i,:) = sum( image_minus_one(diapason, :),1);
      end;
      %% count groups
      R_plus = sum(sum(uint8( f_one>f_original ))); %% regular groups
      S_plus = sum(sum(uint8( f_one<f_original ))); %% singular groups
      %% U_plus = sum(sum(uint8( f_one==f_original ))); %% not used groups
      R_minus = sum(sum(uint8( f_m_one>f_original ))); %% regular groups
      S_minus = sum(sum(uint8( f_m_one<f_original ))); %% singular groups
      %% U_minus = sum(sum(uint8( f_one==f_original ))); %% not used
      %% difference on level of direct and indirect groups
      dR = -(R_plus-R_minus);
      dS = S_plus-S_minus;
      %dU = U_plus-U_minus;
      %% disp([num2str(dR), ' / ', num2str(dS), ' / ', num2str(N)]);
      end
      function image_one = invert_bit(image)
      %% invert last bit
      image_one = bitset(image, 1, 1-bitget(image, 1));
      end
      function image_minus_one = invert_bit_plus_one(image)
      %% invert +1
      image = uint8(image); %% VERY important
      %% e bit plus one
      ones_layer = image.*bitget(image, 1);
      %% convert 1->2, 3->4, ..., 253->254, 255->255 (! we cannot exceed upper
255!!!)
      ones_layer = ones_layer+1;
      %% convert 255->0,
      ones_layer = ones_layer.*(1-bitget(ones_layer, 1));
      %% convert 0->255,
      zeros_layer = 255.*uint8(image==0);
      %% convert 2->1, 4->3, ..., 254->253, 0->0 (! we already convert it, so
- just ignore)
      odd_layer = image.*(1-bitget(image, 1));
      odd_layer = odd_layer-1;
      %%combine layers
      image_minus_one = ones_layer+zeros_layer+odd_layer;
      end
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      function [monoton] = image_monotonn(image, histval)
      %% calculate image monotonnost'
      %% histval - calculated image histogramm. can save a bit processor time.
You can simply pass [] (empty array) to calculate it
      %% [monoton] = image_monotonn(image)
      if(numel(histval)==0)
      %% recalc histogramm
      if(length(size(image))==2)
      %% grayscale image is easier
      histval = imhist(image);
      else
      %% colour image is more difficult
      histval = imhist3(image);
      end;
      end;
      if ( length(size(image))~=length(size(histval)) )
      error('histval must be calculated from image via hist or hist3');
```

```
      end;
      if (length(size(image))==2)
      %% grayscale image is easier
      %% histval = imhist(image);
      %% perfomance trick: value passed here
      %% monoton = max(histval)/numel(image);
      %% get index of max element
      maxval = max(max(max(histval)));
      imax = find(maxval==histval, 1); % much faster than for-end loop
      scale = ones(256, 1);
      for i=1:256
      scale(i) = 1 - exp( abs(imax-i)/256 )/exp(1); % Gaussian attenuation
      end;
      monoton = sum(histval .* scale)/numel(image);
      %% disp([' Monochrome image']);
      else
      %% colour image is more difficult
      %% histval = imhist3(image);
      %% perfomance trick: value passed here
      %% get index of max element
      maxval = max(max(max(histval)));
      indexmax = find(maxval==histval, 1); %% much faster than for-end loop
      kMax = floor(indexmax/(256*256))+1;
      jMax = floor( (indexmax-(kMax-1)*256*256) /256)+1;
      iMax = indexmax-(kMax-1)*256*256 - (jMax-1)*256;
      [iMax, jMax, kMax]
      %% too hard to calculate. load preset
      %% on recalc - it will recalculate about 1-3 minute
      %% scale = ones(256, 256, 256);
      %% for i=1:256
      %% for j=1:256
      %% for k=1:256
      %% scale(i,j,k) = ( 1- sqrt( (i-1)^2+(j-1)^2+(k-1)^2)/(255*sqrt(3)) )^3
; %% parabolic attenuation
      %%  scale(i,j,k) = exp(-( sqrt( (i-1)^2+(j-1)^2+(k-1)^2)/(255*sqrt(3))
)^2 ) ; %% gaussian attenuation
      %% end;
      %% end;
      %% end;
      %% figure; surf(1:256, 1:256, scale(:,:,1))
      %% figure; surf(1:256, 1:256, scale(:,:,128))
      %% scaleInt = int8(scale*127); % save a lot of memory
      %% save('gaussi3d.mat', 'scaleInt')
      load('gaussi3d.mat','scaleInt');
      scale = int32(scaleInt);
      histval = int32(histval);
      %% cube sum
      monoton = 0;
      monoton   =   monoton   +   sum(sum(sum(   histval(iMax:256,   jMax:256,
kMax:256).*scale(1:(256-iMax+1), 1:(256-jMax+1), 1:(256-kMax+1)) )));
      monoton   =   monoton   +   sum(sum(sum(   histval(iMax:-1:1,   jMax:256,
kMax:256).*scale(1:iMax, 1:(256-jMax+1), 1:(256-kMax+1)) )));
      monoton   =   monoton   +   sum(sum(sum(   histval(iMax:256,   jMax:-1:1,
kMax:256).*scale(1:(256-iMax+1), 1:jMax, 1:(256-kMax+1)) )));
      monoton   =   monoton   +   sum(sum(sum(   histval(iMax:-1:1,   jMax:-1:1,
kMax:256).*scale(1:iMax, 1:jMax, 1:(256-kMax+1)) )));
      monoton = monoton + sum(sum(sum( histval(iMax:256, jMax:256, kMax:-
1:1).*scale(1:(256-iMax+1), 1:(256-jMax+1), 1:kMax) )));
      monoton = monoton + sum(sum(sum( histval(iMax:-1:1, jMax:256, kMax:-
1:1).*scale(1:iMax, 1:(256-jMax+1), 1:kMax) )));
      monoton = monoton + sum(sum(sum( histval(iMax:256, jMax:-1:1, kMax:-
1:1).*scale(1:(256-iMax+1), 1:jMax, 1:kMax) )));
      monoton = monoton + sum(sum(sum( histval(iMax:-1:1, jMax:-1:1, kMax:-
1:1).*scale(1:iMax, 1:jMax, 1:kMax) )));
```

```
      %% remove doubled plains
      monoton    =    monoton    -    sum(sum(    histval(iMax:256,    jMax:256,
kMax).*scale(1:(256-iMax+1), 1:(256-jMax+1), 1) ));
      monoton    =    monoton    -    sum(sum(    histval(iMax:-1:1,    jMax:256,
kMax).*scale(1:iMax, 1:(256-jMax+1), 1) ));
      monoton    =    monoton    -    sum(sum(    histval(iMax:256,    jMax:-1:1,
kMax).*scale(1:(256-iMax+1), 1:jMax, 1) ));
      monoton    =    monoton    -    sum(sum(    histval(iMax:-1:1,    jMax:-1:1,
kMax).*scale(1:iMax, 1:jMax, 1) ));
      monoton    =    monoton    -    sum(sum(    histval(iMax:256,    jMax,
kMax:256).*scale(1:(256-iMax+1), 1, 1:(256-kMax+1)) ));
      monoton    =    monoton    -    sum(sum(    histval(iMax:-1:1,    jMax,
kMax:256).*scale(1:iMax, 1, 1:(256-kMax+1)) ));
      monoton    =    monoton    -    sum(sum(    histval(iMax:256,    jMax,    kMax:-
1:1).*scale(1:(256-iMax+1), 1, 1:kMax) ));
      monoton    =    monoton    -    sum(sum(    histval(iMax:-1:1,    jMax,    kMax:-
1:1).*scale(1:iMax, 1, 1:kMax) ));
      monoton = monoton - sum(sum( histval(iMax, jMax:256, kMax:256) .*scale(
1, 1:(256-jMax+1), 1:(256-kMax+1)) ));
      monoton = monoton - sum(sum( histval(iMax, jMax:-1:1, kMax:256) .*scale(
1, 1:jMax, 1:(256-kMax+1)) ));
      monoton = monoton - sum(sum( histval(iMax, jMax:256, kMax:-1:1).*scale(
1, 1:(256-jMax+1), 1:kMax) ));
      monoton = monoton - sum(sum( histval(iMax, jMax:-1:1, kMax:-1:1).*scale(
1, 1:jMax, 1:kMax) ));
      %% add lines summed:
      monoton = monoton + sum( histval(iMax, jMax, kMax:256) .*scale( 1, 1,
1:(256-kMax+1)) );
      monoton = monoton + sum( histval(iMax, jMax, kMax:-1:1).*scale( 1, 1,
1:kMax) );
      monoton = monoton + sum( histval(iMax, jMax:256, kMax).*scale(1, 1:(256-
jMax+1), 1) );
      monoton  =  monoton  +  sum(  histval(iMax,  jMax:-1:1,  kMax).*scale(1,
1:jMax, 1) );
      monoton = monoton + sum( histval(iMax:256, jMax, kMax).*scale(1:(256-
iMax+1), 1, 1) );
      monoton = monoton + sum( histval(iMax:-1:1, jMax, kMax).*scale(1:iMax,
1, 1) );
      %% compensate central point
      monoton = monoton - sum( histval(iMax, jMax, kMax).*scale(1, 1, 1) );
      %% normalize
      monoton = monoton/(size(image,1)*size(image,2));
      monoton = monoton/127; %% see algoritm of calc scaleInt matrix
      %% monoton = sum(histval .* scale)/numel(image);
      %% disp([' Colour image']);
      end;
      end
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      function [str, newoffset] = image_extract(image, offset, bit, bytecount)
      %% [value, newoffset] = image_extract(image, offset, bit)
      %% rarray - image data is inserted
      %% value - value that is built
      %% size - type value for the size (in bytes)
      %% offset - offset from beginning
      %% bit - number of bit to be embedded
      %% wrarray - with built-in image data
      %% newoffset
      %% bitget(A,BIT) gets the bit at position BIT
      %% return 0 or 1.
      str = '';
      for i=1:bytecount
      value = 0;
      for b=1:8 %*2 %2-byte character available
      value = bitset(value, b, bitget(image(offset+b-1), bit) );
```

```
      end;
      str = [str, char(value)];
      offset = offset+8;
      end
      newoffset = offset;
      end
      function [ent] = image_entropiya(image, histval)
      %% calculate image entropiya
      %% histval - calculated image histogramm. can save a bit processor time.
You can simply pass [] (empty array) to calculate it
      %% [ent] = image_entropiya(image)
      image = bitget(image, 1)*255; % make image from bitcut
      %% perfomance trick: value passed here
      if(numel(histval)==0)
      %% recalc histogramm
      if(length(size(image))==2)
      %% grayscale image is easier
      histval = imhist(image);
      else
      %% colour image is more difficult
      histval = imhist3(image);
      end;
      end;
      %% remove zeros (it will be limits to zero, but matlab cannot calculate
limits)
      nonzero_index = find(histval~=0);
      n = numel(nonzero_index);
      p = zeros(1, n);
      for i=1:n
      p(i) = histval(nonzero_index(i));
      end;
      p = p./(size(image,1)*size(image,2)); % possibilty
      ent = sum(p.*log(1./p)/log(2));
      end
      function  [image_embed,  newoffset]  =  image_embed(image,  value,  bit,
offset)
      %% [image_embed, newoffset] = image_embed(image, value, bit, offset)
      %% rarray - image data is inserted
      %% value - value that is built
      %% size - type value for the size (in bytes)
      %% offset - offset from beginning
      %% bit - number of bit to be embedded
      %% wrarray - with built-in image data
      %% newoffset
      %% BITSET(A,BIT,V) sets the bit at position BIT to the value V.
      %% V must be either 0 or 1.
      image_embed = image;
      for i=1:length(value)
      for j=1:8 %*2 % 2-byte character available
      image_embed(offset)  =  bitset(image(offset),  bit,  bitget(1*value(i),  j)
);
      offset=offset+1;
      end;
      end;
      newoffset = offset;
      end
      function [D, M] = image_dispersion(image)
      %% [D, M] = image_dispersion(image)
      %% more description will be later...
      %% M - mat.ogidanie (mean)
      %% D - dispersion (var)
      %%              more:             http://network-journal.mpei.ac.ru/cgi-
bin/main.pl?l=ru&n=11&pa=13&ar=5
      image = bitget(image, 1)*255; % make image from bitcut
```

```
      M = mean(double(image(:)));
      %% M = mean(mean_blocks);
      D = var(double(image(:)));
      %% D = var(mean_blocks);
      end
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      function p = image_chi2 (image)
      %% p = image_chi2(image)
      %%image - image
      %% histval - calculated image histogramm. can save a bit processor time.
You can simply pass [] (empty array) to calculate it
      %% p - probability of having stegocontainer, embedded by LSB method
inside image
      %% p - chi2 dispersion [0,1]
      %%  more:  http://users.ece.cmu.edu/~adrian/487-s06/westfeld-pfitzmann-
ihw99.pdf
      %% https://ru.wikipedia.org/wiki/Критерий_согласия_Пирсона
      %%theoretical values (average)
      if(length(size(image))==2)
      %% grauscale image is easier
      p = image_chi2_layer(image);
      else
      p=0;
      p = p + image_chi2_layer(image(:,:,1)); % R
      p = p + image_chi2_layer(image(:,:,2)); % G
      p = p + image_chi2_layer(image(:,:,3)); % B
      p = p/3; % because 3 layer
      %% colour image is more difficult
      %% histval = imhist3(image);
      end;
      end
      function p = image_chi2_layer(image)
      histval = imhist(image);
      hist_size = numel(histval);
      %% n = sum(h)/2; % correct, but difficult to uderstand
      %% n = numel(image)/2;
      pairs_theory = ( histval(1:2:hist_size)+histval(2:2:hist_size) )/2;
      pairs_sum = sum(pairs_theory); %% correct
      %% practical values
      pairs_practice = histval(2:2:hist_size);
      %remove zeros (it will be limits to zero, but matlab cannot calculate
limits)
      nonzero_index = find(pairs_theory~=0);
      pairs_theory_wo_zero = pairs_theory(nonzero_index);
      pairs_practice_wo_zero = pairs_practice(nonzero_index);
      %% x^2
      p       =       1       -       sum(       (pairs_practice_wo_zero-
pairs_theory_wo_zero).^2./pairs_theory_wo_zero )/pairs_sum;
      end
      function result = image_bitshift_cycle(image, n, n0)
      %% pos = e_strfind(str, find)
      %% more description will be later...
      result = image;
      if(n>=0)
      for i=1:n
      result = bitget(result(:),n0) + bitshift(result(:),1);
      end;
      else
      for i=1:-n
      result = bitshift( bitget(result(:),1), n0-1) + bitshift(result(:),-1);
      end;
      end;
      end
      image_analyze_complex(image, bit)
```

```
      %% [] = image_analyze_complex(image)
      %% more description will be later...
      encode_str = ['Steganography is the art and science of writing hidden
messages in such a way that no one, apart from the senderand intended
recipient, suspects the existence of the message, a form of security through
obscurity. The word steganography is of Greek origin and means "concealed
writing" from the Greek words steganos (????????) meaning "covered or
protected", and graphei (?????) meaning "writing". The first recorded use of
the term was in 1499 by Johannes Trithemius in his Steganographia, a treatise
on cryptography and steganography disguised as a book on magic. Generally,
messages will appear to be something else: images, articles, shopping lists,
or some other covertext and, classically, the hidden message may be in
invisible ink between the visible lines of a private letter.'];
      %% bit = 1;
      %% create string to completely fill container
      n = floor( numel(image) / 8);
      fillfile_encode_str = lorem_ipsum(encode_str, n );
      %% embed
      image_encoded = image_embed(image, fillfile_encode_str, bit, 1); %% 1 -
offset
      imshow(image_encoded);
      pause(0.1);
      %% shift img
      image = image_bitshift_cycle(image, bit-1, 8);
      %% HELPER (for faster calculating
      if(size(image,3)==1)
      %% grayscale image is easier
      histval = imhist(image);
      else
      %% colour image is more difficult
      histval = imhist3(image);
      end;
      %% monotonnost'
      immonot = image_monotonn(image, histval);
      %% meanr and dispersion
      [d, m] = image_dispersion(image);
      %% entropiya
      ent = image_entropiya(image, histval);
      %% chi2
      chi2_orig = image_chi2(image);
      chi2_enc = image_chi2(image_encoded);
      %% ps
      [dR_orig, dS_orig, N] = image_ps(image, 4);
      [dR_enc, dS_enc] = image_ps(image_encoded, 4);
      dR = abs(dR_enc - dR_orig);
      dS = abs(dS_enc - dS_orig);
      end
```