Examination Paper for Foreign Graduates

Course: MATLAB Programming

Date: June 21st, 2018

Title of the Report: Object Removal

Name: VIN ISHIN (文山)

Student No. : LS1706203

School: School 06

Category: ☑ Master Program ☐ Doctoral Program

Nationality: Cambodian

Specialty: Computer Software and Theory

Compulsory/Elective: Elective

Score:

BEIHANG UNIVERSITY

I. Introduction

Object Removal was implemented in MATLAB Programming for removing objects from digital images and replacing them with visually reasonable backgrounds. Figure 1 shows an example of this project, where the foreground person (manually selected as the target region) is replaced by textures sampled from the remainder of the image, and the new color values for the target region is evolved in a way that looks “reasonable” to the human eye.

a **b**

Figure 1: **Removing large objects from images**. (a) Original image. (b) The region corresponding to the foreground person has been manually selected and then automatically removed.

The implementation of this project is based on the inpainting technique, “Object Removal by Exemplar-Based Inpainting”, which is proposed by Criminisi et al. in 2004 [1].

II. GUI Design

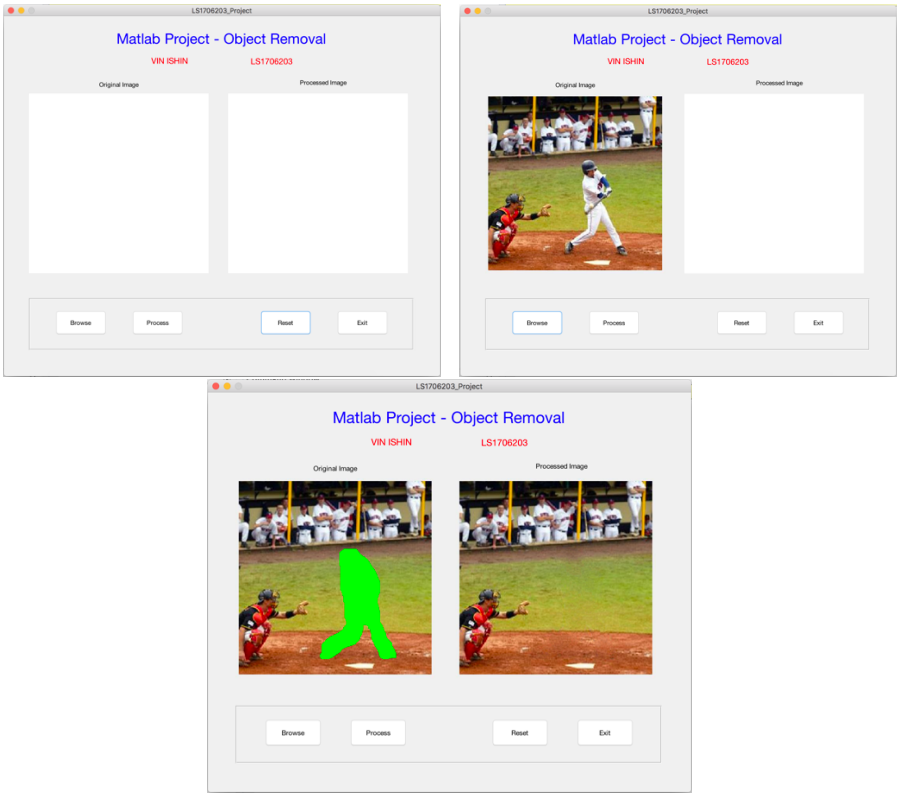

Figure 2: **GUI in action for Object Removal**: The object that is to be removed is selected using a freehand tool. Clicking on the ’Process’ button gives us the result of the implementation.

III. Testing

The following figures are the output of the MATLAB implementation.

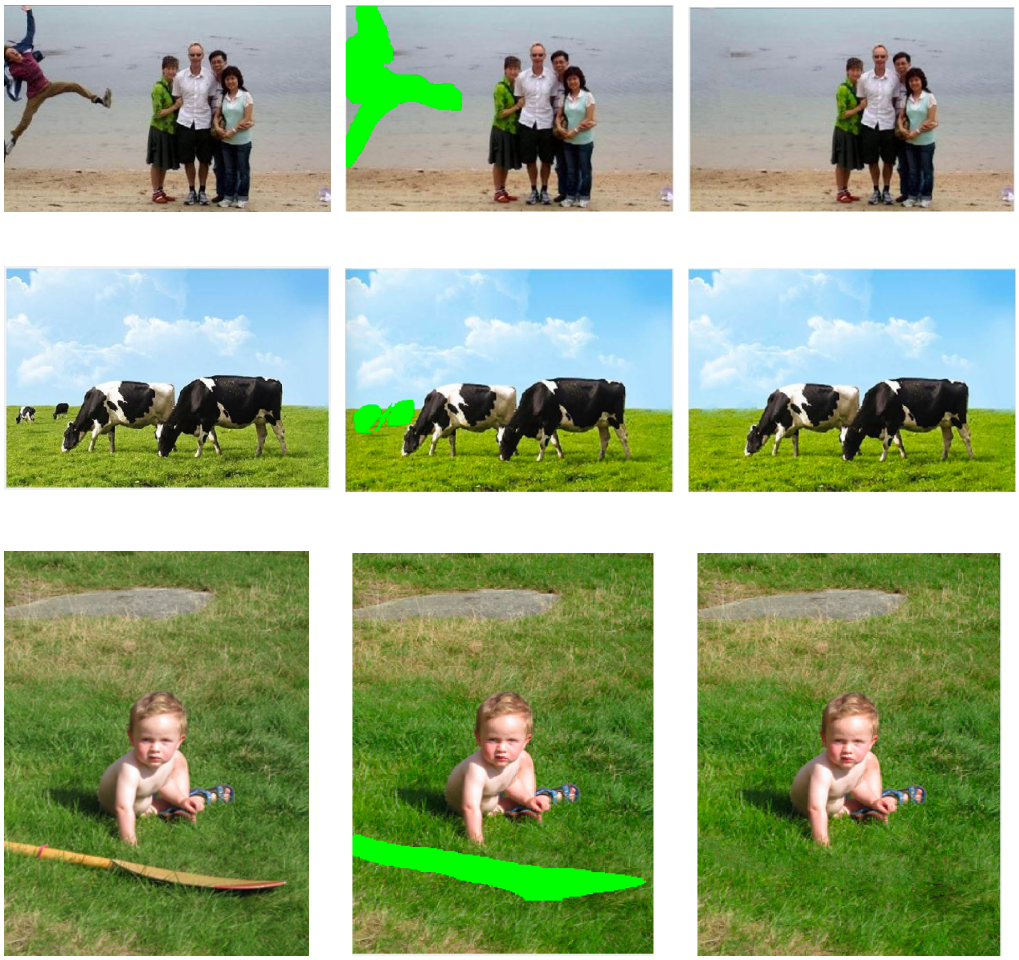

Figure 3: **The output example of Object Removal**

References:

1. Criminisi, Antonio, Patrick Perez, and Kentaro Toyama. “Object removal by exemplar-based inpainting.” Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on. Vol.

Appendix:

File: inpainting.m (for inpainting function)

```
function [inpaintedImg,C,D,fillMovie] = inpainting(origImg,mask,psz)
%INPAINTING The MATLAB implementation of inpainting algorithm by A. Criminisi (2004)
% Inputs:
%   - origImg      original image or corrupted image
%   - mask         implies target region (1 denotes target region)
%   - psz:         patch size (odd scalar). If psz=5, patch size is 5x5.
%
% Outputs:
%   - inpaintedImg   The inpainted image; an MxNx3 matrix of doubles.
%   - C              MxN matrix of confidence values accumulated over all iterations.
%   - D              MxN matrix of data term values accumulated over all iterations.
%   - fillMovie      A Matlab movie struct depicting the fill region over time.

% error check
if ~ismatrix(mask)
    error('Invalid mask');
end
if sum(sum(mask~=0 & mask~=1))>0
    error('Invalid mask');
end
if mod(psz,2)==0
    error('Patch size psz must be odd.');
end

fillRegion = mask;

origImg = double(origImg);
img = origImg;
ind = img2ind(img);
sz = [size(img,1) size(img,2)];
sourceRegion = ~fillRegion;

% Initialize isophote values
[Ix(:,:,3), Iy(:,:,3)] = gradient(img(:,:,3));
[Ix(:,:,2), Iy(:,:,2)] = gradient(img(:,:,2));
[Ix(:,:,1), Iy(:,:,1)] = gradient(img(:,:,1));
Ix = sum(Ix,3)/(3*255); Iy = sum(Iy,3)/(3*255);
temp = Ix; Ix = -Iy; Iy = temp;  % Rotate gradient 90 degrees
```

```
% Initialize confidence and data terms
C = double(sourceRegion);
D = repmat(-.1,sz);
iter = 1;
% Visualization stuff
if nargout==4
    fillMovie(1).cdata=uint8(img);
    fillMovie(1).colormap=[];
    origImg(1,1,:) = [0, 255, 0];
    iter = 2;
end

% Seed 'rand' for reproducible results (good for testing)
rand('state',0);

% Loop until entire fill region has been covered
var = 1;
while any(fillRegion(:))
    var = var + 1;
    % Find contour & normalized gradients of fill region
    fillRegionD = double(fillRegion); % Marcel 11/30/05
    dR = find(conv2(fillRegionD,[1,1,1;1,-8,1;1,1,1],'same')>0);

    [Nx,Ny] = gradient(double(~fillRegion));
    %[Nx,Ny] = gradient(~fillRegion);
    N = [Nx(dR(:)) Ny(dR(:))];
    N = normr(N);
    N(~isfinite(N))=0; % handle NaN and Inf

    % Compute confidences along the fill front
    for k=dR'
        Hp = getpatch(sz,k,psz);
        q = Hp(~(fillRegion(Hp)));
        C(k) = sum(C(q))/numel(Hp);
    end

    % Compute patch priorities = confidence term * data term
    D(dR) = abs(Ix(dR).*N(:,1)+Iy(dR).*N(:,2)) + 0.001;
    priorities = C(dR).* D(dR);

    % Find patch with maximum priority, Hp
    [~,ndx] = max(priorities(:));
    p = dR(ndx(1));
    [Hp,rows,cols] = getpatch(sz,p,psz);
```

```
    toFill = fillRegion(Hp);

    % Find exemplar that minimizes error, Hq
    Hq = bestexemplar(img,img(rows,cols,:),toFill',sourceRegion);

    % Update fill region
    toFill = logical(toFill);
    fillRegion(Hp(toFill)) = false;

    % Propagate confidence & isophote values
    C(Hp(toFill))  = C(p);
    Ix(Hp(toFill)) = Ix(Hq(toFill));
    Iy(Hp(toFill)) = Iy(Hq(toFill));

    % Copy image data from Hq to Hp
    ind(Hp(toFill)) = ind(Hq(toFill));
    img(rows,cols,:) = ind2img(ind(rows,cols),origImg);

    % Visualization stuff
    if nargout==4
        ind2 = ind;
        ind2(logical(fillRegion)) = 1;
        fillMovie(iter).cdata=uint8(ind2img(ind2,origImg));
        fillMovie(iter).colormap=[];
    end
    iter = iter+1;
end
inpaintedImg = img;

%-------------------------------------------------------------------------
% Scans over the entire image (with a sliding window)
% for the exemplar with the lowest error. Calls a MEX function.
%-------------------------------------------------------------------------
function Hq = bestexemplar(img,Ip,toFill,sourceRegion)
m=size(Ip,1); mm=size(img,1); n=size(Ip,2); nn=size(img,2);
best = bestexemplarhelper(mm,nn,m,n,img,Ip,toFill,sourceRegion);
Hq = sub2ndx(best(1):best(2),(best(3):best(4))',mm);

%-------------------------------------------------------------------------
% Returns the indices for a 9x9 patch centered at pixel p.
%-------------------------------------------------------------------------
function [Hp,rows,cols] = getpatch(sz,p,psz)
% [x,y] = ind2sub(sz,p);  % 2*w+1 == the patch size
w=(psz-1)/2; p=p-1; y=floor(p/sz(1))+1; p=rem(p,sz(1)); x=floor(p)+1;
```

```
rows = max(x-w,1):min(x+w,sz(1));
cols = (max(y-w,1):min(y+w,sz(2)))';
Hp = sub2ndx(rows,cols,sz(1));

%-----------------------------------------------------------------------
% Converts the (rows,cols) subscript-style indices to Matlab index-style
% indices.  Unforunately, 'sub2ind' cannot be used for this.
%-----------------------------------------------------------------------
function N = sub2ndx(rows,cols,nTotalRows)
X = rows(ones(length(cols),1),:);
Y = cols(:,ones(1,length(rows)));
N = X+(Y-1)*nTotalRows;

%-----------------------------------------------------------------------
% Converts an indexed image into an RGB image, using 'img' as a colormap
%-----------------------------------------------------------------------
function img2 = ind2img(ind,img)
for i=3:-1:1
    temp=img(:,:,i);
    img2(:,:,i)=temp(ind);
end

%-----------------------------------------------------------------------
% Converts an RGB image into a indexed image, using the image itself as
% the colormap.
%-----------------------------------------------------------------------
function ind = img2ind(img)
s=size(img);
ind=reshape(1:s(1)*s(2),s(1),s(2));
```

File: LS1706203_Project.m (for GUI callback function)

** This part shows only the important functions in this project.

```
% --- Executes just before LS1706203_Project is made visible.
function LS1706203_Project_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to LS1706203_Project (see VARARGIN)

% Choose default command line output for LS1706203_Project
handles.output = hObject;
```

```
a = ones(256, 256);
axes(handles.axes1);
imshow(a);
axes(handles.axes2);
imshow(a);

% Update handles structure
guidata(hObject, handles);
```

** Code for “Browse” button

```
% --- Executes on button press in btnBrowse.
function btnBrowse_Callback(hObject, eventdata, handles)
% hObject    handle to btnBrowse (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

[filename, pathname] = uigetfile({'*.jpg;*.tif;*.png;*.gif;*.bmp;*.jpeg'},'File
Selector');
if isequal(filename, 0) || isequal(pathname, 0)
    disp('User pressed cancel');
    return;
end
handles.myImage = strcat(pathname, filename);
axes(handles.axes1);
handles.myImage = imread(handles.myImage);
imshow(handles.myImage);
h = imfreehand(gca);
xy = h.getPosition;
xCoordinates = xy(:, 1);
yCoordinates = xy(:, 2);
handles.xCoordinates = xCoordinates;
handles.yCoordinates = yCoordinates;

% Changing value at those pixels which have value as 255 to have pixel value as 254.
% This is being done to ensure that the region that is to be inpainted has
% not pixel value overlap with any of the pixels in the original region
handles.myImage(handles.myImage == 255) = 254;
uploadedImage = handles.myImage;
handles.uploadedImage = uploadedImage;

numberOfCoordinates=(length(handles.myImage(:,1,1)))*(length(handles.myImage(1,:,1)));
imageCoordinate = zeros(numberOfCoordinates, 2);
var = 1;
```

```
for i = 1:length(handles.myImage(:, 1, 1))
    for j = 1:length(handles.myImage(1, :, 1))
        imageCoordinate(var, 2) = i;
        imageCoordinate(var, 1) = j;
        var = var + 1;
    end
end

inPolygonOrNot=inpolygon(imageCoordinate(:,1),imageCoordinate(:,2),xCoordinates,yCoordinates);
for someVar = 1: numberOfCoordinates
    if inPolygonOrNot(someVar) == 1
        handles.myImage(imageCoordinate(someVar,2),imageCoordinate(someVar,1),1) = 0;
        handles.myImage(imageCoordinate(someVar,2),imageCoordinate(someVar,1),2) = 255;
        handles.myImage(imageCoordinate(someVar,2),imageCoordinate(someVar,1),3) = 0;
    end
end

%Let's construct the mask that we are going to use later
mask = zeros(length(handles.myImage(:, 1, 1)), length(handles.myImage(1, :, 1)));
for someVar = 1:numberOfCoordinates
    if inPolygonOrNot(someVar) == 1
        mask(imageCoordinate(someVar, 2), imageCoordinate(someVar, 1)) = 255;
    end
end
handles.mask = mask;
imshow(handles.myImage)

% save the updated handles object
guidata(hObject,handles);
```

** Code for “Process” button

```
% --- Executes on button press in btnProcess.
function btnProcess_Callback(hObject, eventdata, handles)
% hObject    handle to btnProcess (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

if isfield(handles,'uploadedImage')
    originalImage = handles.uploadedImage;
    imageFilled1=regionfill(originalImage(:,:,1),handles.xCoordinates,handles.yCoordinates);
    imageFilled2=regionfill(originalImage(:,:,2),handles.xCoordinates,handles.yCoordinates);
    imageFilled3=regionfill(originalImage(:,:,3),handles.xCoordinates,handles.yCoordinates);
    imageFilled(:, :, 1) = imageFilled1;
```

```
    imageFilled(:, :, 2) = imageFilled2;
    imageFilled(:, :, 3) = imageFilled3;
    mask = handles.mask;
    mask = mat2gray(mask);
    psz = 15;
    [inpaintedImage, C, D, fillMovie] = inpainting(imageFilled, mask, psz);
    inpaintedImage = uint8(inpaintedImage);
    handles.modifiedImage1 = inpaintedImage;
    axes(handles.axes2)
    imshow(handles.modifiedImage1)
    implay(fillMovie);

    % save the updated handles object
    guidata(hObject,handles);
else
    disp('No input');
end
```

** Code for “Reset” button

```
% --- Executes on button press in btnReset.
function btnReset_Callback(hObject, eventdata, handles)
% hObject    handle to btnReset (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

a = ones(256, 256);
axes(handles.axes1);
imshow(a);
axes(handles.axes2);
imshow(a);
handles = rmfield(handles, 'uploadedImage');

% save the updated handles object
guidata(hObject,handles);
```

** Code for “Exit” button

```
% --- Executes on button press in btnExit.
function btnExit_Callback(hObject, eventdata, handles)
% hObject    handle to btnExit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close
```

File: bestexemplarhelper.c (for exemplar-based inpainting algorithm)

```
#include "mex.h"
#include <limits.h>

void bestexemplarhelper(const int mm, const int nn, const int m, const int n,
       const double *img, const double *Ip, const mxLogical *toFill,
       const mxLogical *sourceRegion, double *best) {
    register int i,j,ii,jj,ii2,jj2,M,N,I,J,ndx,ndx2,mn=m*n,mmnn=mm*nn;
    double patchErr=0.0,err=0.0,bestErr=1000000000.0;

    /* foreach patch */
    N=nn-n+1;  M=mm-m+1;
    for(j=1; j<=N; ++j) {
       J=j+n-1;
       for(i=1; i<=M; ++i) {
          I=i+m-1;
          /*** Calculate patch error ***/
          /* foreach pixel in the current patch */
          for(jj=j,jj2=1; jj<=J; ++jj,++jj2) {
             for(ii=i,ii2=1; ii<=I; ++ii,++ii2) {
                ndx=ii-1+mm*(jj-1);
                if(!sourceRegion[ndx]) goto skipPatch;
                ndx2=ii2-1+m*(jj2-1);
                if(!toFill[ndx2]) {
                    err=img[ndx     ] - Ip[ndx2    ]; patchErr += err*err;
                    err=img[ndx+=mmnn] - Ip[ndx2+=mn]; patchErr += err*err;
                    err=img[ndx+=mmnn] - Ip[ndx2+=mn]; patchErr += err*err;
                }
             }
          }
          /*** Update ***/
          if(patchErr < bestErr) {
             bestErr = patchErr;
             best[0] = i; best[1] = I;
             best[2] = j; best[3] = J;
          }
          /*** Reset ***/
          skipPatch:
             patchErr = 0.0;
       }
    }
}
```

```
/* best = bestexemplarhelper(mm,nn,m,n,img,Ip,toFill,sourceRegion); */
void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[]) {
    int mm,nn,m,n;
    double *img,*Ip,*best;
    mxLogical *toFill,*sourceRegion;

    /* Extract the inputs */
    mm = (int)mxGetScalar(prhs[0]);
    nn = (int)mxGetScalar(prhs[1]);
    m  = (int)mxGetScalar(prhs[2]);
    n  = (int)mxGetScalar(prhs[3]);
    img = mxGetPr(prhs[4]);
    Ip  = mxGetPr(prhs[5]);
    toFill = mxGetLogicals(prhs[6]);
    sourceRegion = mxGetLogicals(prhs[7]);

    /* Setup the output */
    plhs[0] = mxCreateDoubleMatrix(4,1,mxREAL);
    best = mxGetPr(plhs[0]);
    best[0]=best[1]=best[2]=best[3]=0.0;

    /* Do the actual work */
    bestexemplarhelper(mm,nn,m,n,img,Ip,toFill,sourceRegion,best);
}
```

** NOTE: To make the “bestexemplarhelper” function works, execute a command “mex bestexemplarhelper.c”.