DIGITAL SIGNAL PROCESSING

OPTICAL CHARACTER RECOGNITION

OCR can be used for Document Digitization, Accessibility for Visually Impaired, License Plate Recognition, Check Processing, Identity Verification, etc.

OBJECTIVE

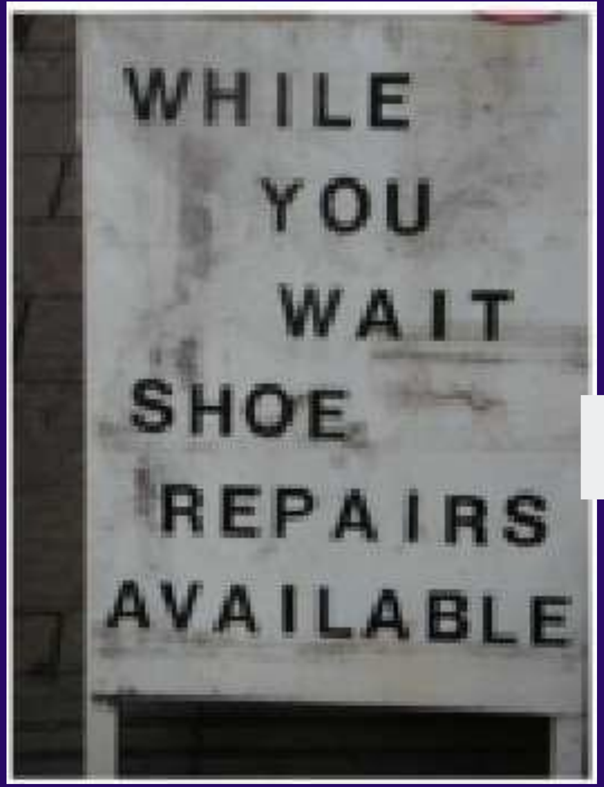
To recognise the characters from an image handwritten or

printed text into machine-encoded text.

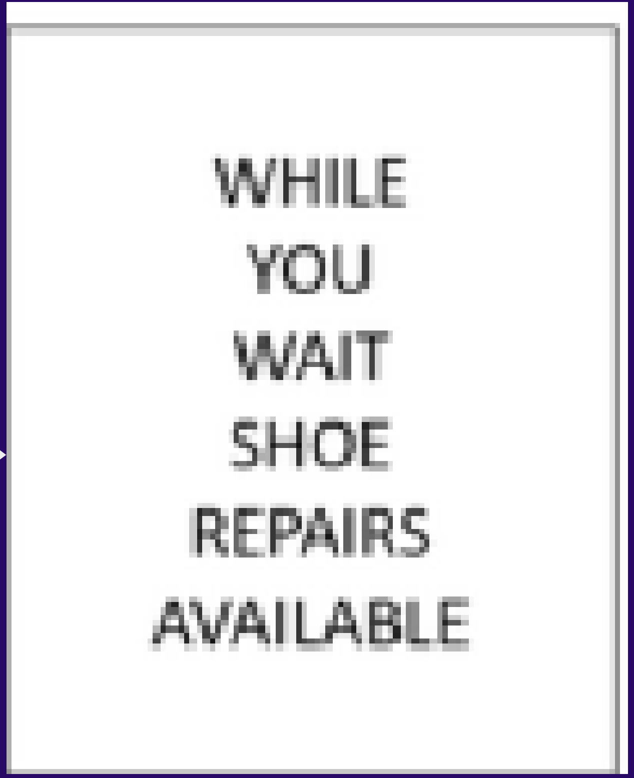
INTRODUCTION

Optical Character Recognition (OCR) is the type of annotation that allows the transcription of images of typed or handwritten messages into machine-recognizable text.

SOFTWARE REQUIRED: MATLAB



OCR



```
WHILE
YOU
WAIT
SHOE
REPAIRS
AVAILABLE
```

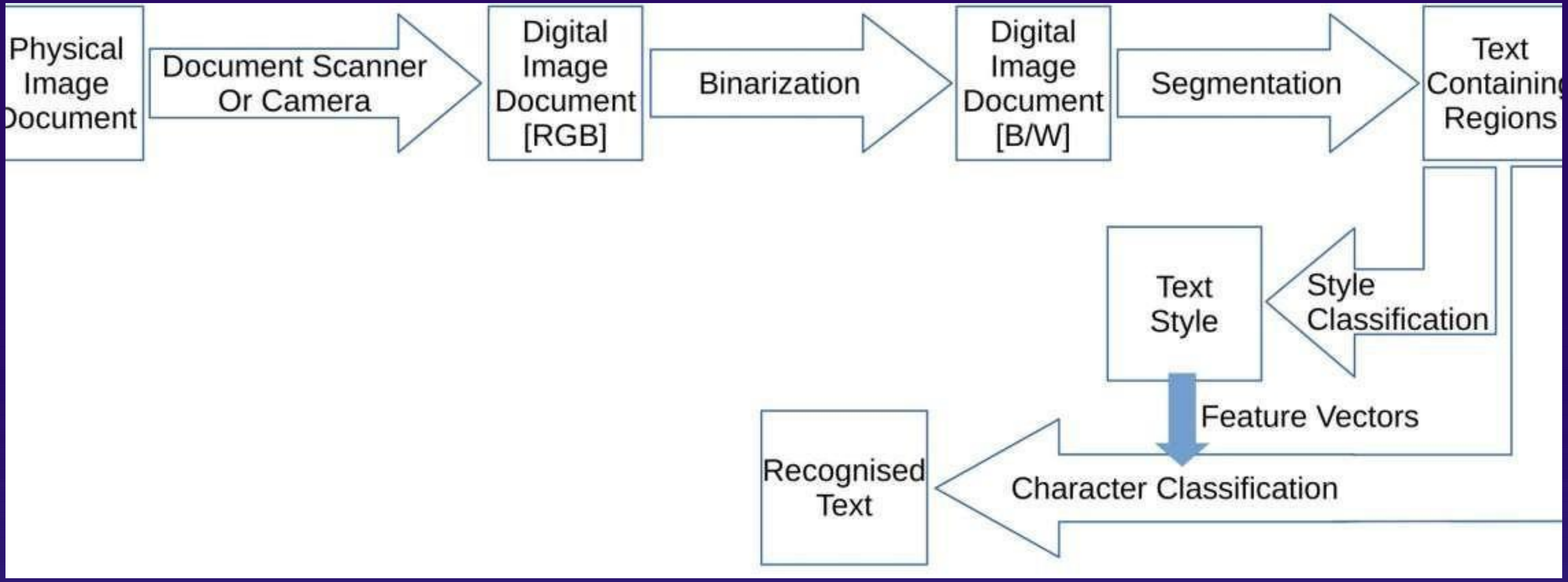
INPUT THE IMAGE TO RECOGNISE TEXT FROM

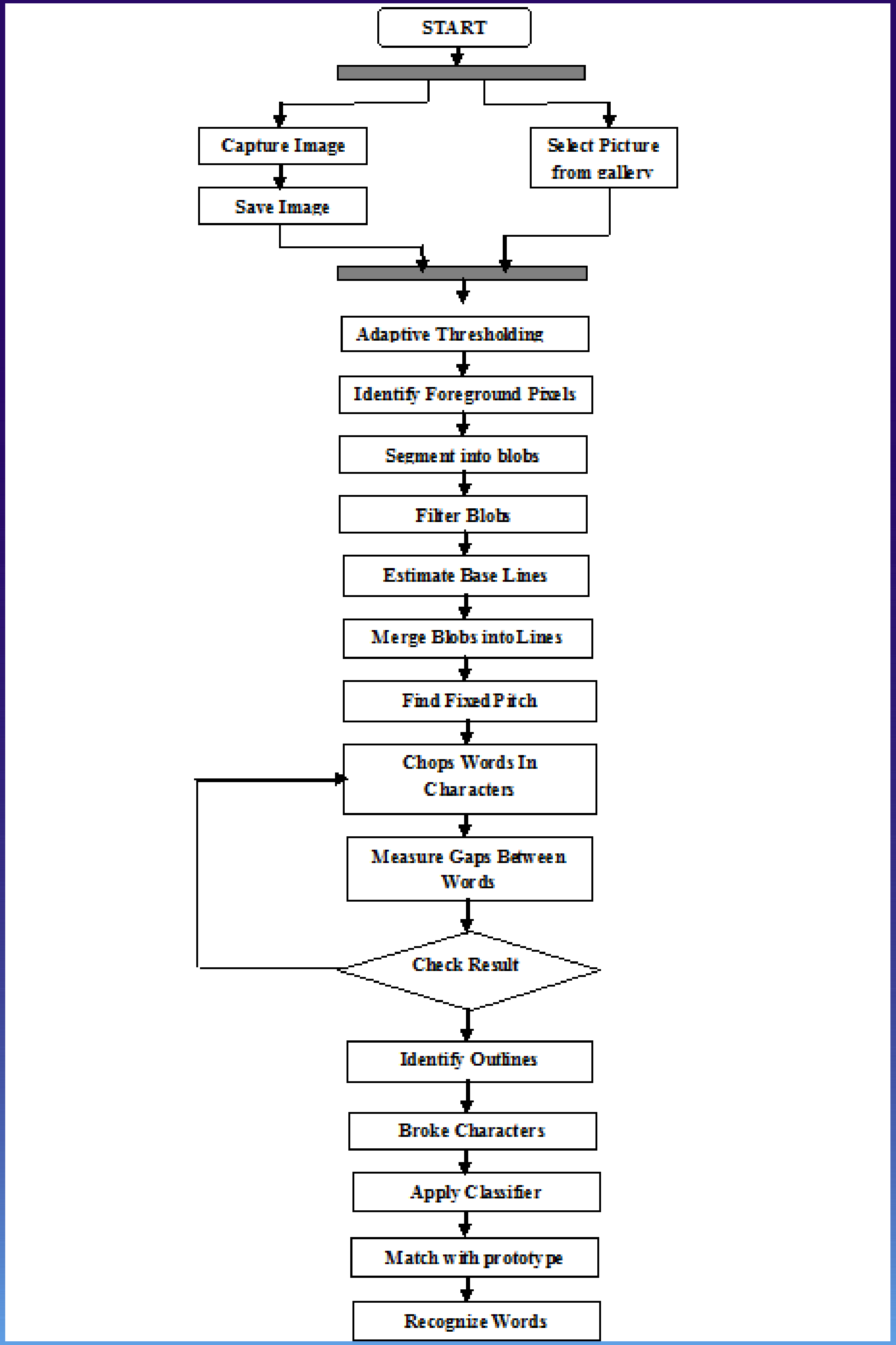
To perform optical character recognition (OCR) using Python, we can use the Tesseract OCR library.

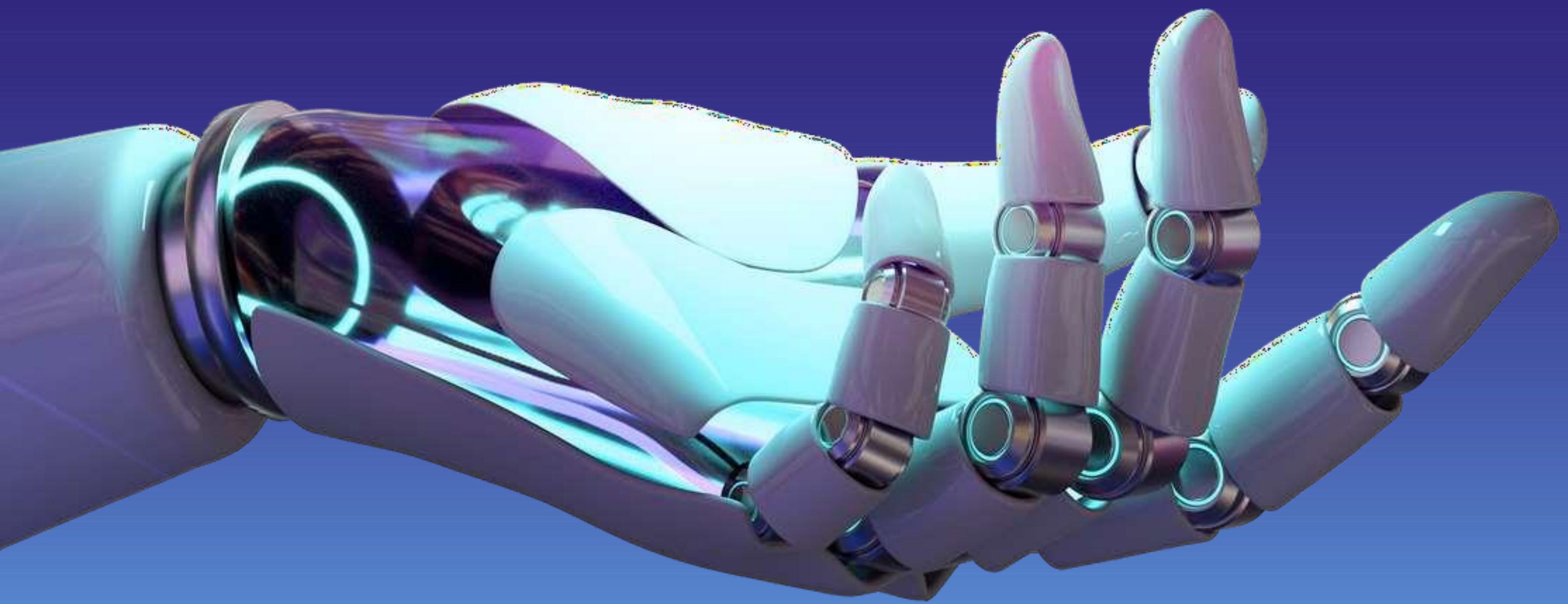
OUTPUT THE RECOGNISED TEXT

BLOCK DIAGRAM

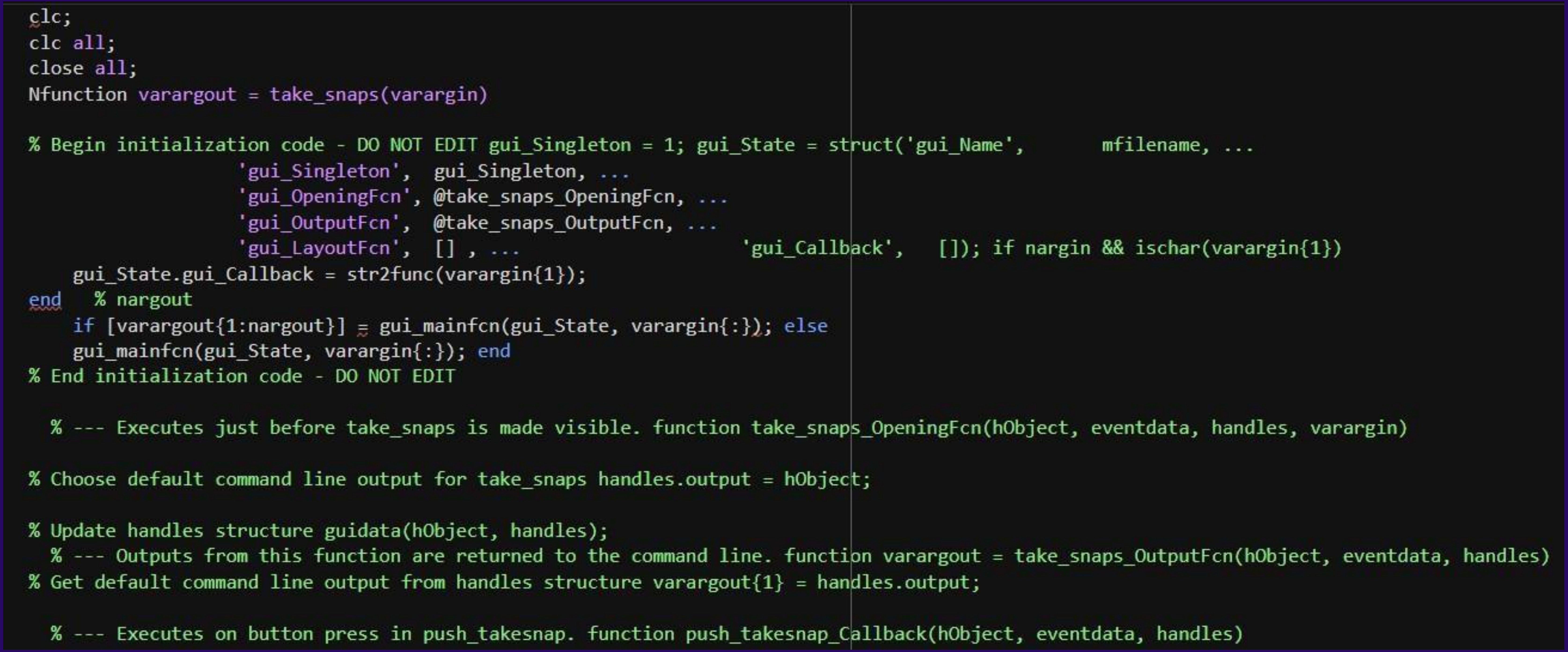
FLOW CHART



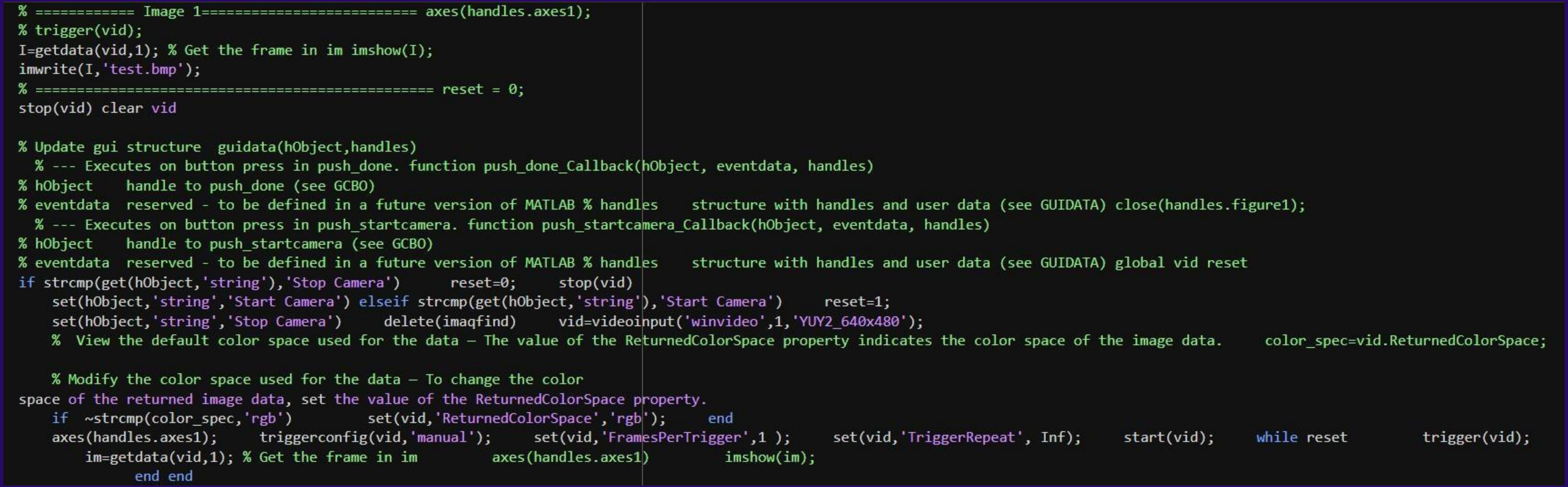




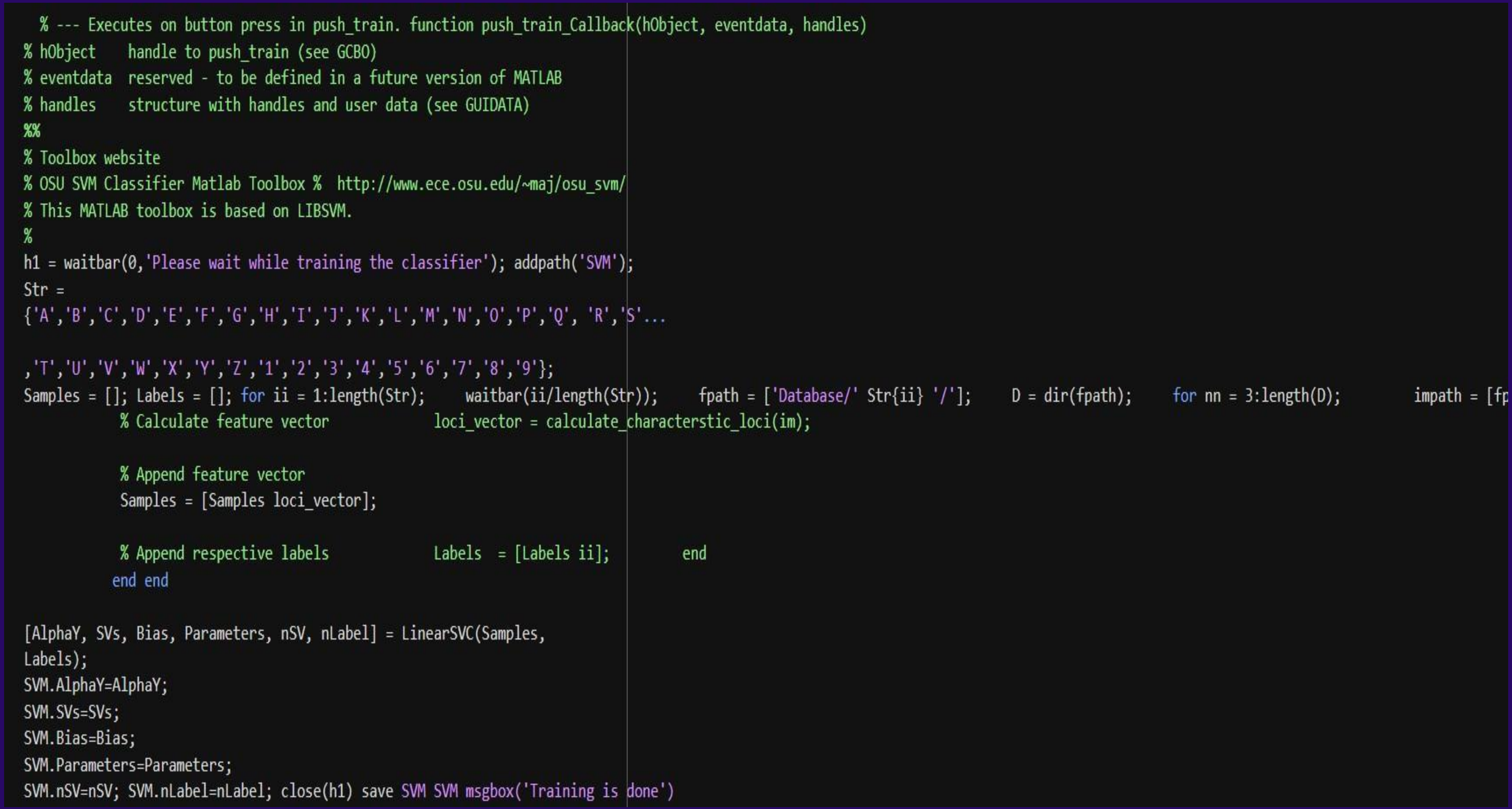
CODE:



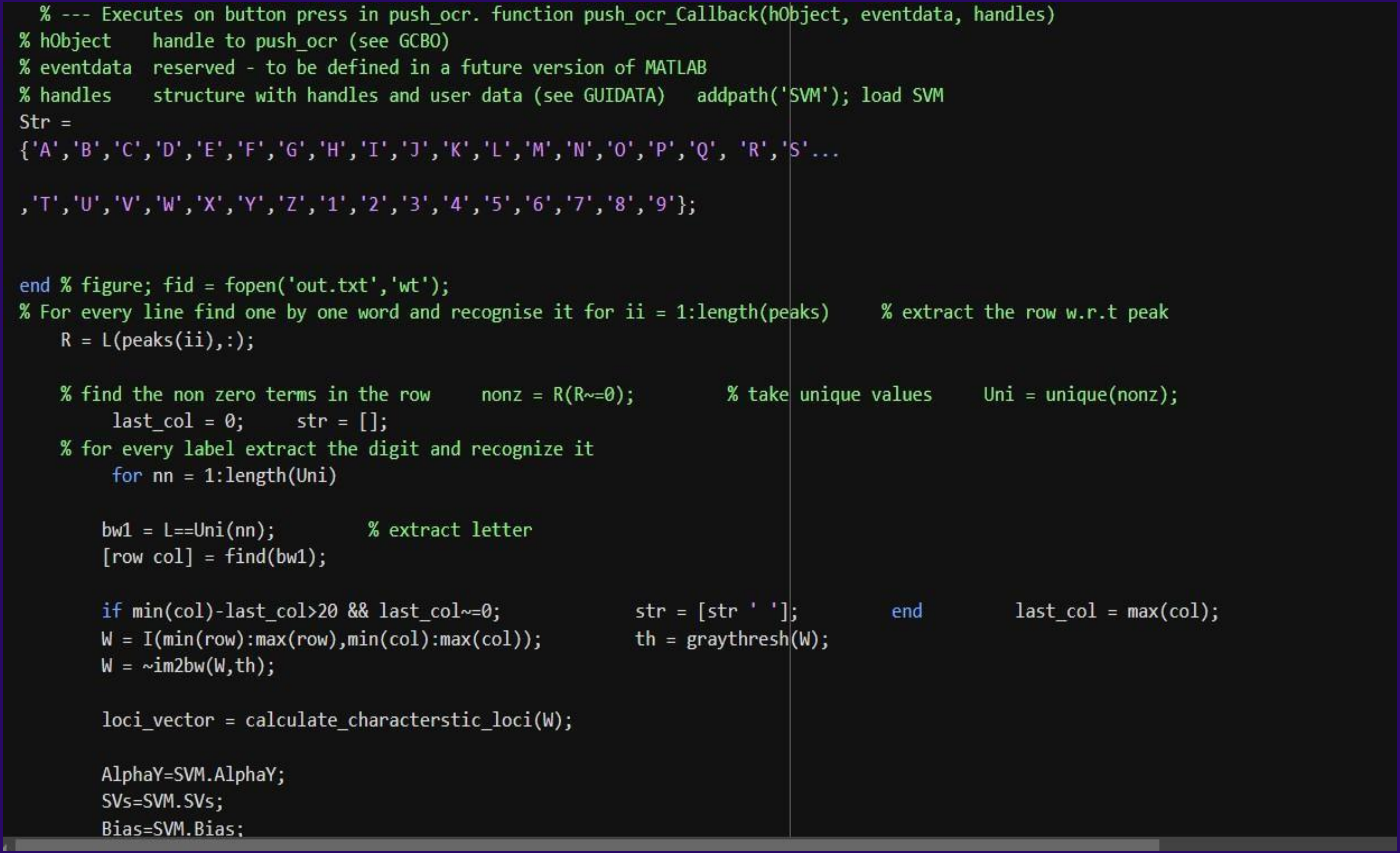
```
clc;
clc all;
close all;
Nfunction varargout = take_snaps(varargin)

% Begin initialization code - DO NOT EDIT gui_Singleton = 1; gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @take_snaps_OpeningFcn, ...
                   'gui_OutputFcn',  @take_snaps_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...                     'gui_Callback',   []); if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end   % nargout
    if [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:}); else
    gui_mainfcn(gui_State, varargin{:}); end
% End initialization code - DO NOT EDIT

  % --- Executes just before take_snaps is made visible. function take_snaps_OpeningFcn(hObject, eventdata, handles, varargin)

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

% Update handles structure guidata(hObject, handles);
  % --- Outputs from this function are returned to the command line. function varargout = take_snaps_OutputFcn(hObject, eventdata, handles)
% Get default command line output from handles structure varargout{1} = handles.output;

  % --- Executes on button press in push_takesnap. function push_takesnap_Callback(hObject, eventdata, handles)
```



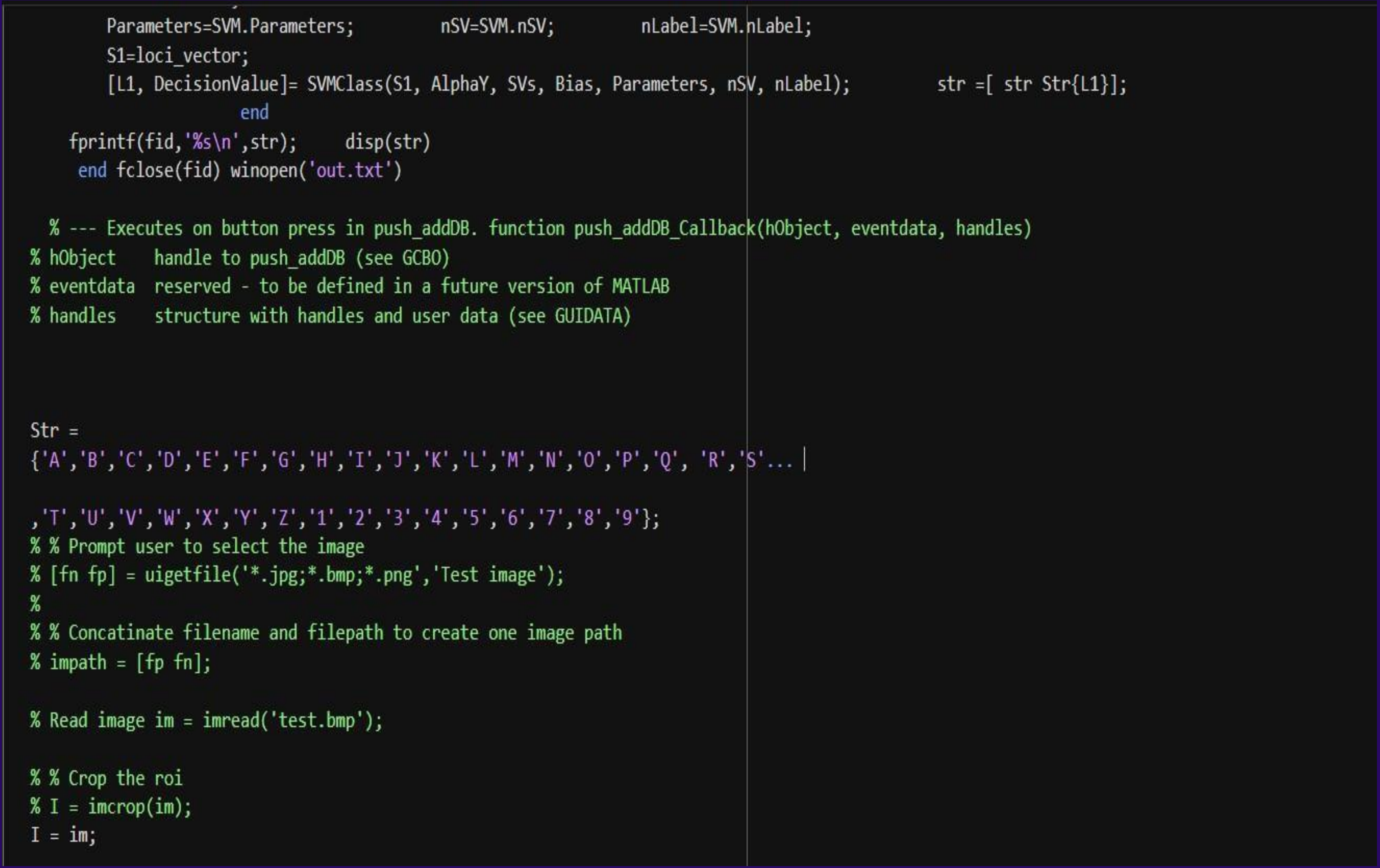
```
% =========== Image 1======================== axes(handles.axes1);
% trigger(vid);
I=getdata(vid,1); % Get the frame in im imshow(I);
imwrite(I,'test.bmp');
% ================================================ reset = 0;
stop(vid) clear vid

% Update gui structure  guidata(hObject,handles)
  % --- Executes on button press in push_done. function push_done_Callback(hObject, eventdata, handles)
% hObject    handle to push_done (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB % handles    structure with handles and user data (see GUIDATA) close(handles.figure1);
  % --- Executes on button press in push_startcamera. function push_startcamera_Callback(hObject, eventdata, handles)
% hObject    handle to push_startcamera (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB % handles    structure with handles and user data (see GUIDATA) global vid reset
if strcmp(get(hObject,'string'),'Stop Camera')      reset=0;     stop(vid)
    set(hObject,'string','Start Camera') elseif strcmp(get(hObject,'string'),'Start Camera')     reset=1;
    set(hObject,'string','Stop Camera')     delete(imaqfind)     vid=videoinput('winvideo',1,'YUY2_640x480');
    %  View the default color space used for the data – The value of the ReturnedColorSpace property indicates the color space of the image data.     color_spec=vid.ReturnedColorSpace;

    % Modify the color space used for the data – To change the color
space of the returned image data, set the value of the ReturnedColorSpace property.
    if  ~strcmp(color_spec,'rgb')         set(vid,'ReturnedColorSpace','rgb');     end
    axes(handles.axes1);     triggerconfig(vid,'manual');     set(vid,'FramesPerTrigger',1 );     set(vid,'TriggerRepeat', Inf);     start(vid);     while reset         trigger(vid);
        im=getdata(vid,1); % Get the frame in im         axes(handles.axes1)         imshow(im);
              end end
```



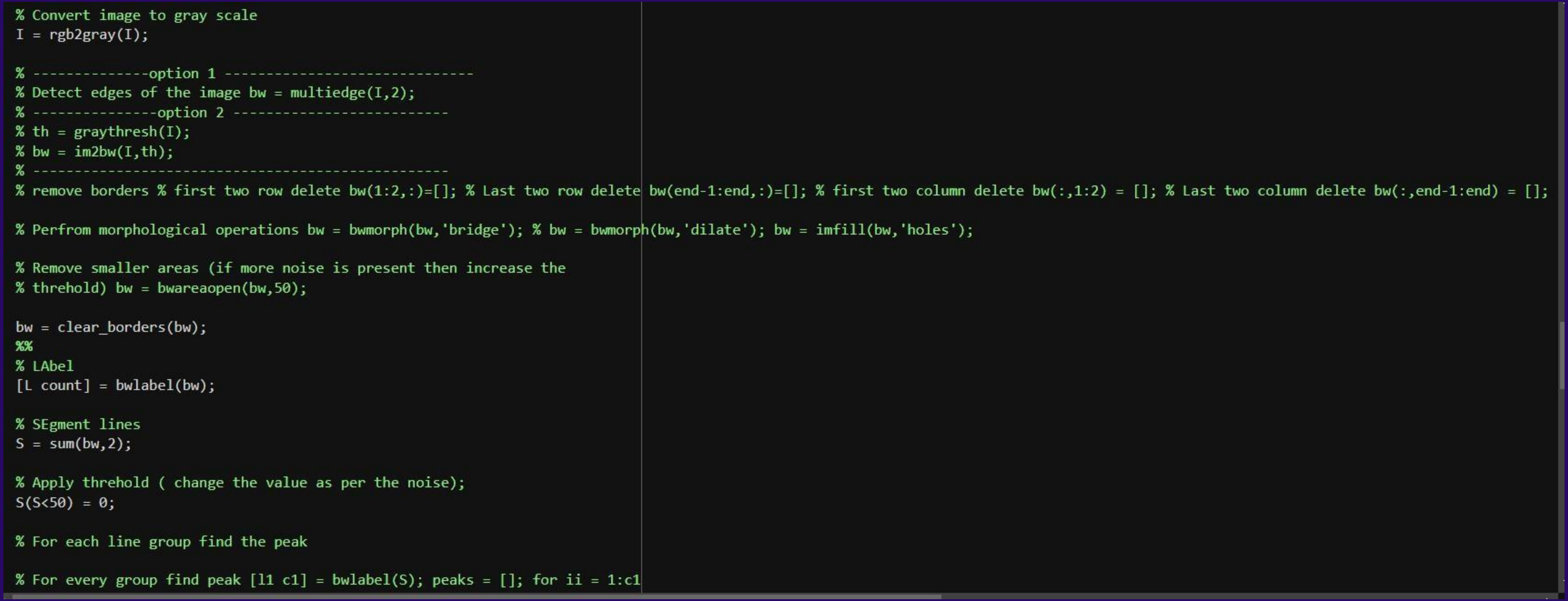
```
  % --- Executes on button press in push_train. function push_train_Callback(hObject, eventdata, handles)
% hObject    handle to push_train (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%%
% Toolbox website
% OSU SVM Classifier Matlab Toolbox %  http://www.ece.osu.edu/~maj/osu_svm/
% This MATLAB toolbox is based on LIBSVM.
%
h1 = waitbar(0,'Please wait while training the classifier'); addpath('SVM');
Str =
{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q', 'R','S'...

,'T','U','V','W','X','Y','Z','1','2','3','4','5','6','7','8','9'};
Samples = []; Labels = []; for ii = 1:length(Str);     waitbar(ii/length(Str));     fpath = ['Database/' Str{ii} '/'];     D = dir(fpath);     for nn = 3:length(D);         impath = [fp
            % Calculate feature vector             loci_vector = calculate_characterstic_loci(im);

            % Append feature vector
            Samples = [Samples loci_vector];

            % Append respective labels             Labels  = [Labels ii];         end
           end end

[AlphaY, SVs, Bias, Parameters, nSV, nLabel] = LinearSVC(Samples,
Labels);
SVM.AlphaY=AlphaY;
SVM.SVs=SVs;
SVM.Bias=Bias;
SVM.Parameters=Parameters;
SVM.nSV=nSV; SVM.nLabel=nLabel; close(h1) save SVM SVM msgbox('Training is done')
```



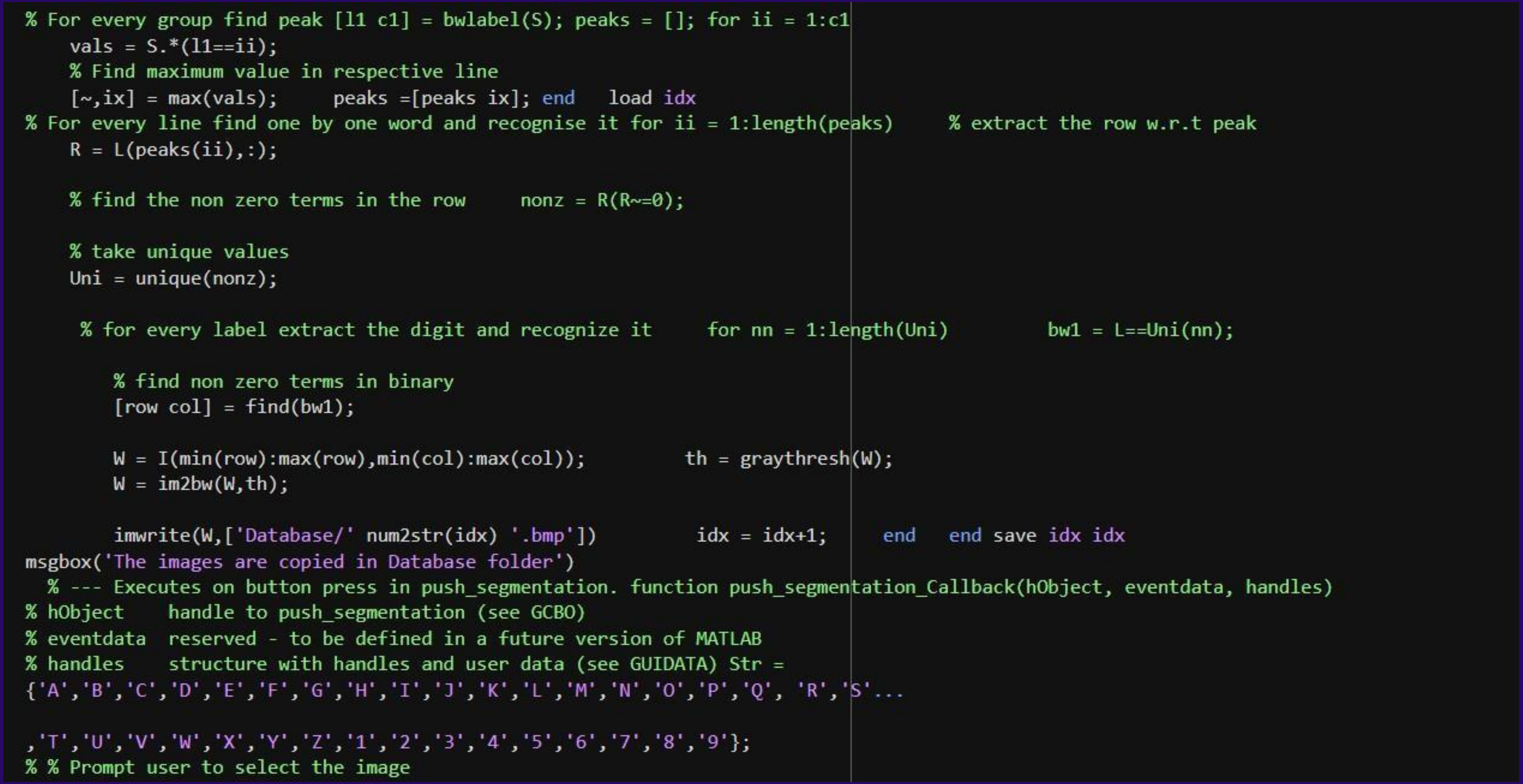
```
  % --- Executes on button press in push_ocr. function push_ocr_Callback(hObject, eventdata, handles)
% hObject    handle to push_ocr (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)   addpath('SVM'); load SVM
Str =
{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q', 'R','S'...

,'T','U','V','W','X','Y','Z','1','2','3','4','5','6','7','8','9'};


end % figure; fid = fopen('out.txt','wt');
% For every line find one by one word and recognise it for ii = 1:length(peaks)     % extract the row w.r.t peak
    R = L(peaks(ii),:);

    % find the non zero terms in the row     nonz = R(R~=0);         % take unique values     Uni = unique(nonz);
         last_col = 0;     str = [];
    % for every label extract the digit and recognize it
         for nn = 1:length(Uni)

        bw1 = L==Uni(nn);         % extract letter
        [row col] = find(bw1);

        if min(col)-last_col>20 && last_col~=0;             str = [str ' '];         end         last_col = max(col);
        W = I(min(row):max(row),min(col):max(col));         th = graythresh(W);
        W = ~im2bw(W,th);

        loci_vector = calculate_characterstic_loci(W);

        AlphaY=SVM.AlphaY;
        SVs=SVM.SVs;
        Bias=SVM.Bias;
```



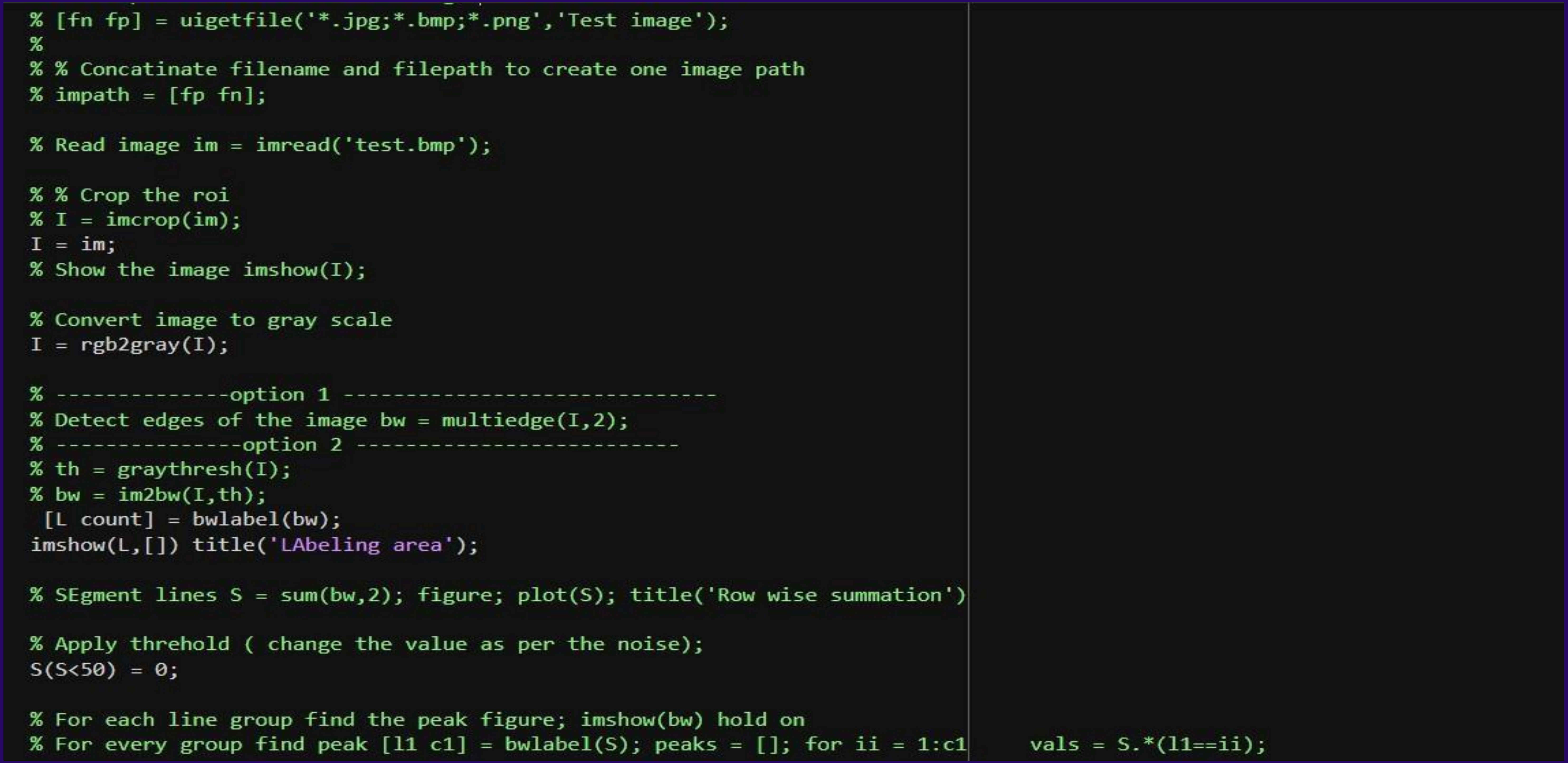
```
        Parameters=SVM.Parameters;        nSV=SVM.nSV;        nLabel=SVM.nLabel;
        S1=loci_vector;
        [L1, DecisionValue]= SVMClass(S1, AlphaY, SVs, Bias, Parameters, nSV, nLabel);        str =[ str Str{L1}];
                     end
    fprintf(fid,'%s\n',str);     disp(str)
     end fclose(fid) winopen('out.txt')

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



Str =
{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q', 'R','S'...

,'T','U','V','W','X','Y','Z','1','2','3','4','5','6','7','8','9'};
% % Prompt user to select the image
% [fn fp] = uigetfile('*.jpg;*.bmp;*.png','Test image');
%
% % Concatinate filename and filepath to create one image path
% impath = [fp fn];

% Read image im = imread('test.bmp');

% % Crop the roi
% I = imcrop(im);
I = im;
```



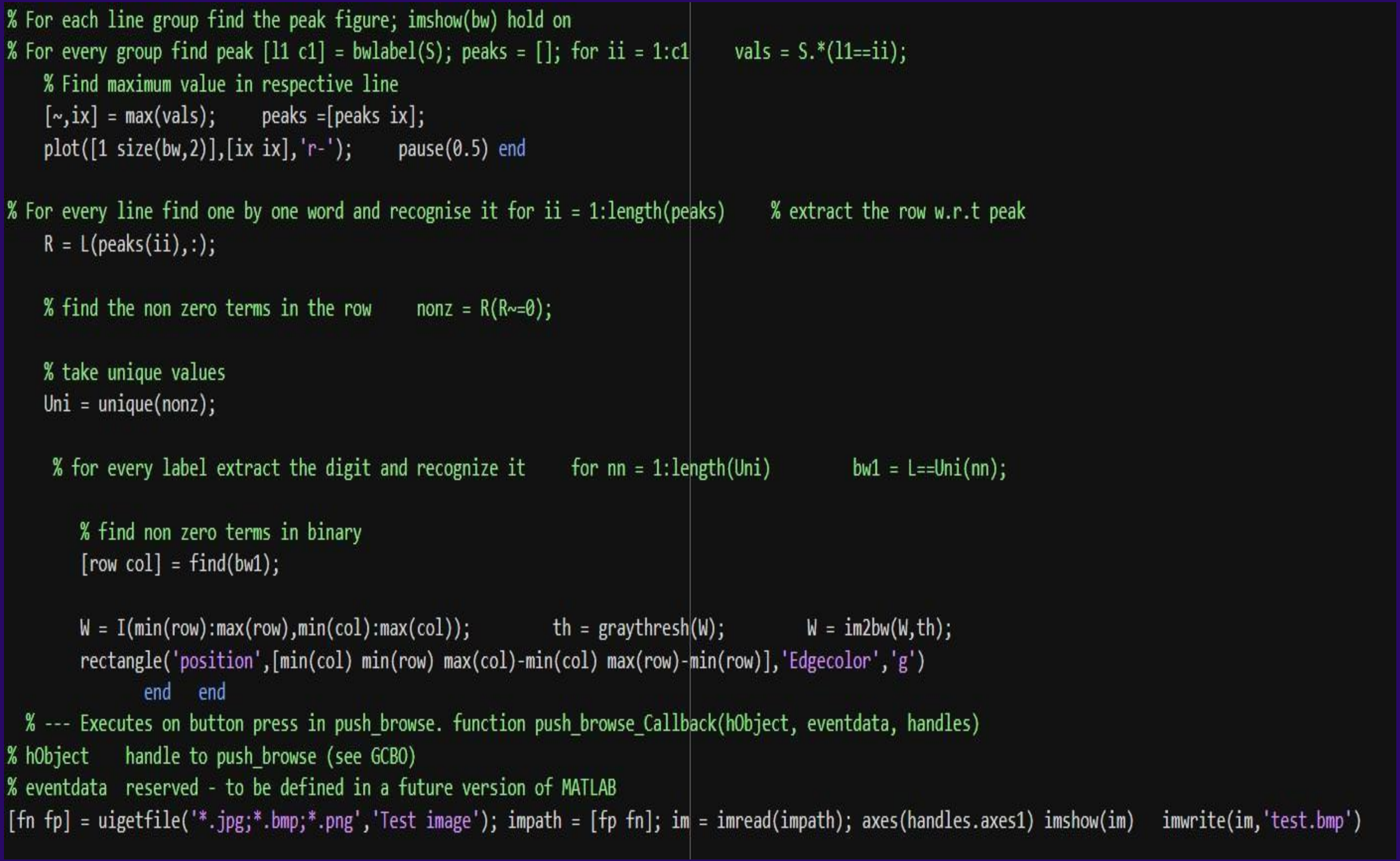
```
% Convert image to gray scale
I = rgb2gray(I);

% --------------option 1 ------------------------------
% Detect edges of the image bw = multiedge(I,2);
% ---------------option 2 --------------------------
% th = graythresh(I);
% bw = im2bw(I,th);
% --------------------------------------------------
% remove borders % first two row delete bw(1:2,:)=[]; % Last two row delete bw(end-1:end,:)=[]; % first two column delete bw(:,1:2) = []; % Last two column delete bw(:,end-1:end) = [];

% Perfrom morphological operations bw = bwmorph(bw,'bridge'); % bw = bwmorph(bw,'dilate'); bw = imfill(bw,'holes');

% Remove smaller areas (if more noise is present then increase the
% threhold) bw = bwareaopen(bw,50);

bw = clear_borders(bw);
%%
% LAbel
[L count] = bwlabel(bw);

% SEgment lines
S = sum(bw,2);

% Apply threhold ( change the value as per the noise);
S(S<50) = 0;

% For each line group find the peak

% For every group find peak [l1 c1] = bwlabel(S); peaks = []; for ii = 1:c1
```



```
% For every group find peak [l1 c1] = bwlabel(S); peaks = []; for ii = 1:c1
    vals = S.*(l1==ii);
    % Find maximum value in respective line
    [~,ix] = max(vals);     peaks =[peaks ix]; end   load idx
% For every line find one by one word and recognise it for ii = 1:length(peaks)     % extract the row w.r.t peak
    R = L(peaks(ii),:);

    % find the non zero terms in the row     nonz = R(R~=0);

    % take unique values
    Uni = unique(nonz);

     % for every label extract the digit and recognize it     for nn = 1:length(Uni)         bw1 = L==Uni(nn);

        % find non zero terms in binary
        [row col] = find(bw1);

        W = I(min(row):max(row),min(col):max(col));         th = graythresh(W);
        W = im2bw(W,th);

        imwrite(W,['Database/' num2str(idx) '.bmp'])         idx = idx+1;     end   end save idx idx
msgbox('The images are copied in Database folder')
  % --- Executes on button press in push_segmentation. function push_segmentation_Callback(hObject, eventdata, handles)
% hObject    handle to push_segmentation (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA) Str =
{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q', 'R','S'...

,'T','U','V','W','X','Y','Z','1','2','3','4','5','6','7','8','9'};
% % Prompt user to select the image
```



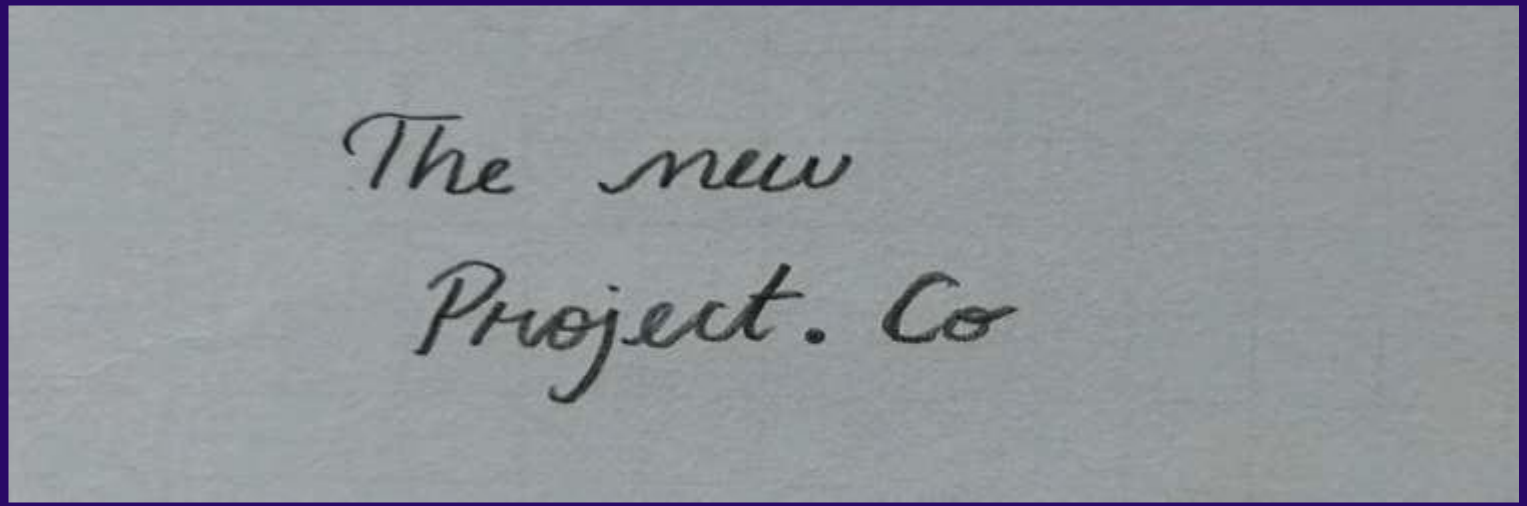
```
% [fn fp] = uigetfile('*.jpg;*.bmp;*.png','Test image');
%
% % Concatinate filename and filepath to create one image path
% impath = [fp fn];

% Read image im = imread('test.bmp');

% % Crop the roi
% I = imcrop(im);
I = im;
% Show the image imshow(I);

% Convert image to gray scale
I = rgb2gray(I);

% --------------option 1 ------------------------------
% Detect edges of the image bw = multiedge(I,2);
% ---------------option 2 --------------------------
% th = graythresh(I);
% bw = im2bw(I,th);
 [L count] = bwlabel(bw);
imshow(L,[]) title('LAbeling area');

% SEgment lines S = sum(bw,2); figure; plot(S); title('Row wise summation')

% Apply threhold ( change the value as per the noise);
S(S<50) = 0;

% For each line group find the peak figure; imshow(bw) hold on
% For every group find peak [l1 c1] = bwlabel(S); peaks = []; for ii = 1:c1     vals = S.*(l1==ii);
```



```
% For each line group find the peak figure; imshow(bw) hold on
% For every group find peak [l1 c1] = bwlabel(S); peaks = []; for ii = 1:c1     vals = S.*(l1==ii);
    % Find maximum value in respective line
    [~,ix] = max(vals);     peaks =[peaks ix];
    plot([1 size(bw,2)],[ix ix],'r-');     pause(0.5) end

% For every line find one by one word and recognise it for ii = 1:length(peaks)     % extract the row w.r.t peak
    R = L(peaks(ii),:);

    % find the non zero terms in the row     nonz = R(R~=0);

    % take unique values
    Uni = unique(nonz);

     % for every label extract the digit and recognize it     for nn = 1:length(Uni)         bw1 = L==Uni(nn);

        % find non zero terms in binary
        [row col] = find(bw1);

        W = I(min(row):max(row),min(col):max(col));         th = graythresh(W);         W = im2bw(W,th);
        rectangle('position',[min(col) min(row) max(col)-min(col) max(row)-min(row)],'Edgecolor','g')
               end   end
  % --- Executes on button press in push_browse. function push_browse_Callback(hObject, eventdata, handles)
% hObject    handle to push_browse (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
[fn fp] = uigetfile('*.jpg;*.bmp;*.png','Test image'); impath = [fp fn]; im = imread(impath); axes(handles.axes1) imshow(im)   imwrite(im,'test.bmp')
```

Output:

TEST SAMPLE: EDGE DETECTION:


The new
Project. Co

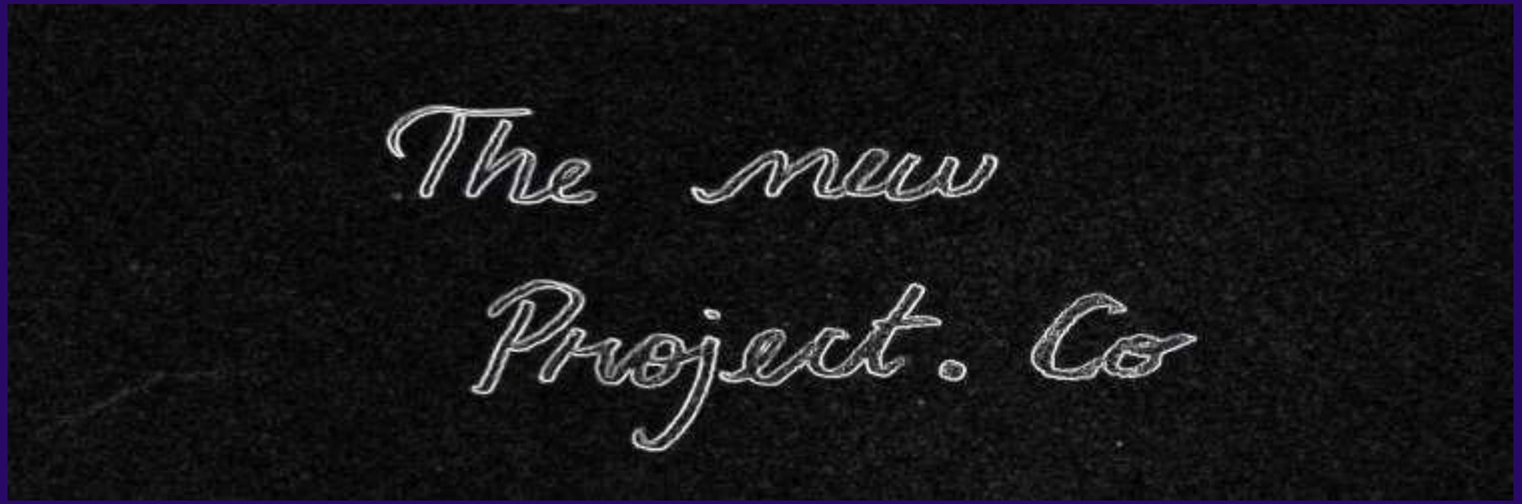
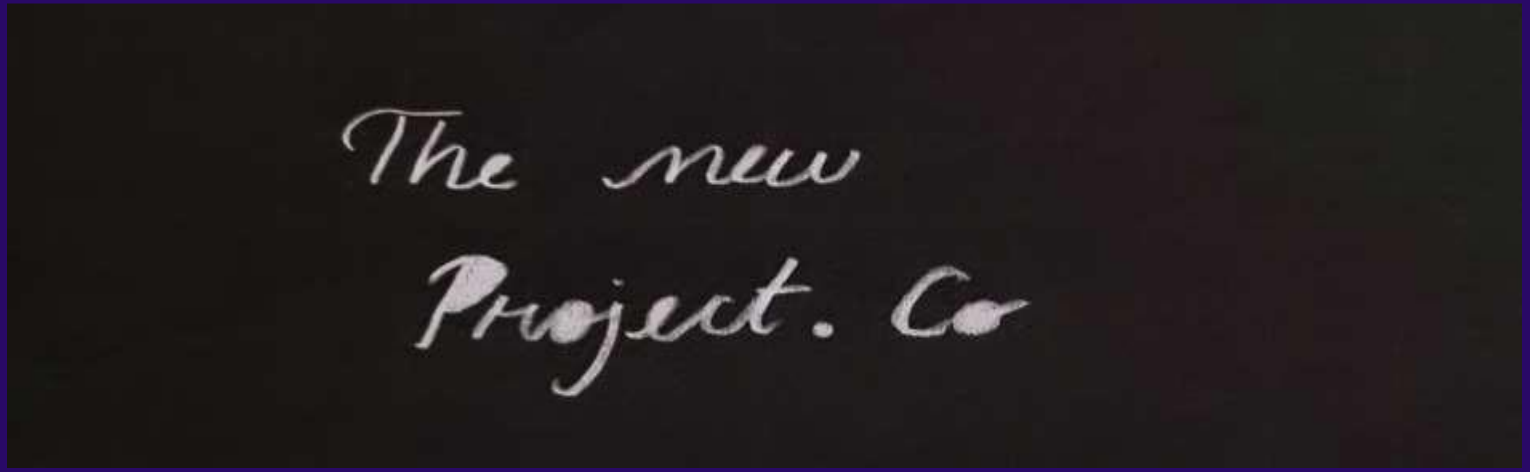

The new
Project. Co

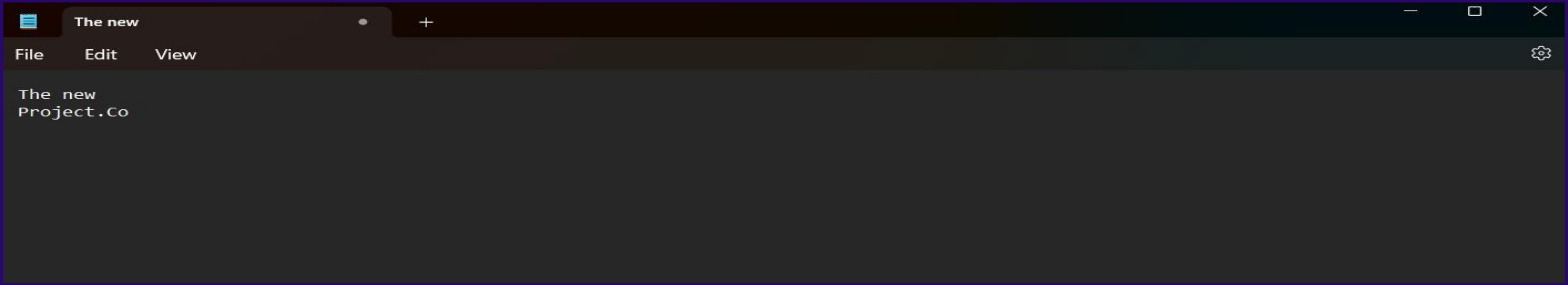
IMAGE MORPHING:



LABELLING:

BOUNDING BOX:

Final Output:



```
The new
Project.Co
```

Outcome and future Scope:

By using the methods of characterization loci and SVM toolbox we were able to achieve an accuracy in the range of 60-70 % on untrained data for online character recognition which dramatically increased over 85% when the same data was trained into the database. The coding used for noise reduction was also successful and capable of removing stray marks on the sheet of paper being used as well as the other noises that came while taking the picture were also removed automatically.

Today's uses of OCR are still somewhat limited to the scanning of the written word into useable computer text. The uses include word processing, mail delivery system scanning, ticket reading, and other such tasks.

In the future, the uses of OCR have been speculated to be far more advanced. Some of these advancements are already in the workings. Most believe that the use for reading the written text will diminish as electronic data interchange (EDI) is used more and more, while paper documents are phased out.

However, a recent AIIM study found that imaging increased paper consumption because of increased printing. While the debate carries on, it is clear that advancements will lead OCR to greater heights than before.

OCR is already used to detect viruses, or unfortunately create them, and to stop spammers. Anti- spamming applications continue to be improved, as the need for increased technology is a constant. OCR is used to prevent viruses by detecting codes hidden in images.

While the difficulty of transferring information from legacy systems to modern operating systems can be cumbersome, OCR is able to read screenshots. This can facilitate the transferring of information between incompatible technologies.

One of the current hopes for OCR is the chance of developing OCR software that can read compressed files. Text that is compressed into an image and saved as ASCII or Hexadecimal data could be read by OCR and transferred back into readable text.

Finally, it is anticipated that OCR will be used in the development of more advanced robotics. The eyes of a robot are essentially a camera meant to input information. If OCR is used to help the robot comprehend text, the uses could be almost endless. All of these advancements and more are expected to keep the technology of OCR in use and continue to expand its current capabilities.