

《计算机视觉（1）》实验报告

实验七 我的拍照/扫描全能王

实验小组成员 (学号+班级+姓名)	分工及主要完成任务	成绩
201800830004 18 数科 叶家辉	全部	

山东大学

2021 年 3 月

完成综合实验，实现对相机拍摄或者扫描仪扫描的文档照片的处理，目的在于去除图像中的阴影区域或光照亮区，使得处理后的图像打印清晰，视觉显示效果大幅提升。算法利用学过的知识可自行设计，以处理效果优秀，速度快的方法为胜。具体实现方案可依托计算机实现、微信小程序或手机APP上实现（请三选二）。

请大家用相机拍摄（或者扫描仪扫描）书籍、试卷和各类证件照片，作为此实验的处理照片集合，同学们每人至少提供 3 张不同类型且具有一定处理价值的照片（也就是不能是完美的不需要处理的图像）。实验报告写在如下空白处，页数不限。

一、实验目的与原理

本实验的目的在于实现对相机拍摄或者扫描仪扫描的文档照片的处理，使得处理后的图像打印清晰，视觉显示效果大幅提升。一般来说，手机拍摄的照片都是不规则的，需要先进行一系列预处理，以提取完整且规整的文档区域，然后再进行图像增强，以锐化文字区域，消除阴影或明亮区域。本实验中，预处理按先后步骤主要包括图像的旋转、将图像转换为灰度图、图像的投影变换和图像的裁剪。

1.1 图像的旋转

图像是由像素构成的，像素是构成图像的不可分割的最小单元。在计算机中，一幅图像可以看作是由若干像素点组成的二维矩阵。因此，对图像进行的操作就可以看作是对这个二维矩阵的操作。

对于图像的旋转，我们可以推导出矩阵变换的公式。直觉上我们认为图像的旋转应该绕着中心点，然而图像默认是以左上角为原点的，所以我们首先要将图像左上角的原点移到中心点，具体方法是进行如下变换：

$$(x_1, y_1, 1) = (x_0, y_0, 1) \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ -0.5W & 0.5H & 1 \end{pmatrix}$$

其中 W 为图像的宽， H 为图像的高。原坐标为 (x_0, y_0) 的点经上述变换后得到坐标为 (x_1, y_1) 的点。假设图像旋转角度（逆时针）为 θ ，设旋转后得到的坐标为 (x_2, y_2) ，则应有如下变换：

$$(x_2, y_2, 1) = (x_1, y_1, 1) \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

假设旋转后的图像宽为 W' ，高为 H' ，那么将坐标原点变回左上角需要进行如下变换：

$$(x_3, y_3, 1) = (x_2, y_2, 1) \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0.5W' & 0.5H' & 1 \end{pmatrix}$$

最后得到的 (x_3, y_3) 即为图像 (x_0, y_0) 的点旋转（逆时针） θ 后得到的坐标。

当然，现实世界的我们认为旋转是连续的，可是计算机对图像的处理却是离散的，这就导致旋转后新图像的某些像素值会有所缺失。这是我们会用到插值的方法对缺失的像素进行赋值。三种计算方法（包括最近邻插值、双线性插值和三次线性插值）及其原理上课的时候已经详细讨论过，在此不再赘述。

1.2 图像的灰度转化

在对图像进行之后的操作之前，一般都会将其转化为灰度图像，也就是对其 RGB 三色通道经过一定运算转变为一个通道。这可以通过内部函数自动化地实现。

1.3 图像的投影变换

投影变换也叫透视变换，可以将图片投影到一个新的视平面(Viewing Plane)。通用的变换公式为：

$$[x', y', w'] = [u, v, w] \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

其中 (u, v) 是原始图片中点的坐标， (x', y') 是变换后的图片坐标。

变换矩阵 $\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$ 可以拆成四个部分。 $\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ 表示线性变换， $[a_{31} \ a_{32}]$ 表示平移，

$\begin{bmatrix} a_{13} \\ a_{23} \end{bmatrix}^T$ 产生透视变换。所以仿射变换是投影变换的特殊形式。经过投影变换之后的图片通常不是平行四边形（除非映射视平面和原来平面平行的情况）。

重写之前的变换公式可以得到：

$$x = \frac{x'}{w'} = \frac{a_{11}u + a_{12}v + a_{13}}{a_{31}u + a_{32}v + a_{33}}$$

$$y = \frac{y'}{w'} = \frac{a_{21}u + a_{22}v + a_{23}}{a_{31}u + a_{32}v + a_{33}}$$

所以，已知变换对应的几个点就可以求取变换公式。假设有一个正方形到四边形的变换：

$$(0,0) \rightarrow (x_0, y_0), (1,0) \rightarrow (x_1, y_1), (1,1) \rightarrow (x_2, y_2), (0,1) \rightarrow (x_3, y_3)$$

根据变换公式得到：

$$\begin{aligned} a_{31} &= x_0 \\ a_{11} + a_{31} - a_{13}x_1 &= x_1 \\ a_{11} + a_{21} + a_{31} - a_{13}x_2 - a_{23}x_2 &= x_2 \\ a_{21} + a_{31} - a_{23}x_3 &= x_3 \\ a_{32} &= y_0 \\ a_{12} + a_{32} - a_{13}y_1 &= y_1 \\ a_{12} + a_{22} + a_{32} - a_{13}y_2 - a_{23}y_2 &= y_2 \\ a_{22} + a_{32} - a_{23}y_3 &= y_3 \end{aligned}$$

定义几个辅助变量：

$$\begin{aligned} \Delta x_1 &= x_1 - x_2 & \Delta x_2 &= x_3 - x_2 & \Delta x_3 &= x_0 - x_1 + x_2 - x_3 \\ \Delta y_1 &= y_1 - y_2 & \Delta y_2 &= y_3 - y_2 & \Delta y_3 &= y_0 - y_1 + y_2 - y_3 \end{aligned}$$

当 $\Delta x_3, \Delta y_3$ 不为 0 时，得到：

$$\begin{aligned}a_{11} &= x_1 - x_0 + a_{12}x_1 \\a_{21} &= x_3 - x_0 + a_{12}x_2 \\a_{31} &= x_0 \\a_{12} &= y_1 - y_0 + a_{13}y_1 \\a_{22} &= y_3 - y_0 + a_{23}y_3 \\a_{32} &= y_0 \\a_{13} &= \begin{vmatrix} \Delta x_3 & \Delta x_2 \\ \Delta y_3 & \Delta y_2 \end{vmatrix} / \begin{vmatrix} \Delta x_1 & \Delta x_2 \\ \Delta y_1 & \Delta y_2 \end{vmatrix} \\a_{12} &= \begin{vmatrix} \Delta x_1 & \Delta x_3 \\ \Delta y_1 & \Delta y_3 \end{vmatrix} / \begin{vmatrix} \Delta x_1 & \Delta x_2 \\ \Delta y_1 & \Delta y_2 \end{vmatrix}\end{aligned}$$

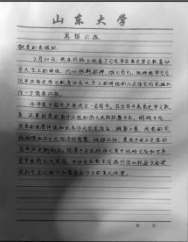


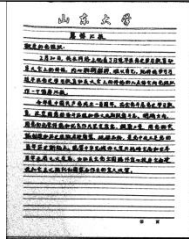


求解出的变换矩阵就可以将一个正方形变换到四边形。反之，四边形变换到正方形也是一样的。

1.4 图像的裁剪

我们知道，如果要裁剪一块矩形区域，则只需要确定图像任意一条对角线上的两点即可。我们可以通过图形化交互界面来确定裁剪区域的具体位置，然后调用 Matlab 内置函数来实现对确定区域的裁剪。

1.5 图像增强

突出文字、淡化边缘的方案有很多，总的来说就是要找到文字的特征，增强这些部分，同时相对来说就减弱了其他区域。这里选择其中一种综合的、经实验测试较为可靠的方案。首先对图像进行均值滤波(a)，这可以一定程度上去除灰色阴影区域。接下来用一次高斯滤波(b)。然后按阈值筛选出低对比度区域，选出文字区域(c)，为最后的处理提供模版 (mask)。再将之前均值滤波的图像按一定参数（暂且称之为文字增强参数）减去高斯滤波的结果，就能突出文字区域，而相对来说就隐去了其他区域(d)。最后再仿照 OpenCV 中 copyTo() 函数的实现，得到最终的处理图(e)。

					
原图	a	b	c	d	e

我们可以发现经过这一系列处理得到的结果还是比较不错的。

二、实验内容与步骤

2.1 界面总体设计与图像导入

本实验的 GUI 界面延续之前一贯的简洁明快的设计风格。同时为了集成计算机视觉 (1) 的所有实验项目, 在 GUI 的顶部特别设计了导航栏, 可以在同一窗口定位到不同的实验项目。点击“7-扫描全能王”后即可得到本实验的整体操作界面。左侧区域显示的是原始图像, 中间区域显示的是预处理后的图像, 中间下部可选择具体的预处理操作, 右侧区域显示的是对预处理后图像进行图像增强的最终结果。



和前几个实验一样, 左上角有“打开文件”和“保存图像”两个图标。点击“打开文件”图标选择相应图片, 即可在坐标图区 axes1 (第一行第一列图像显示区) 导入这张图片。“打开文件”按钮的回调函数如下图所示。前四行获取了所选定图片的文件名和路径名, 再根据得到的路径名和文件名读取文件。后两行则把读取到的文件显示到坐标图区, 并设定顶部的文本框显示图片路径名/文件名。

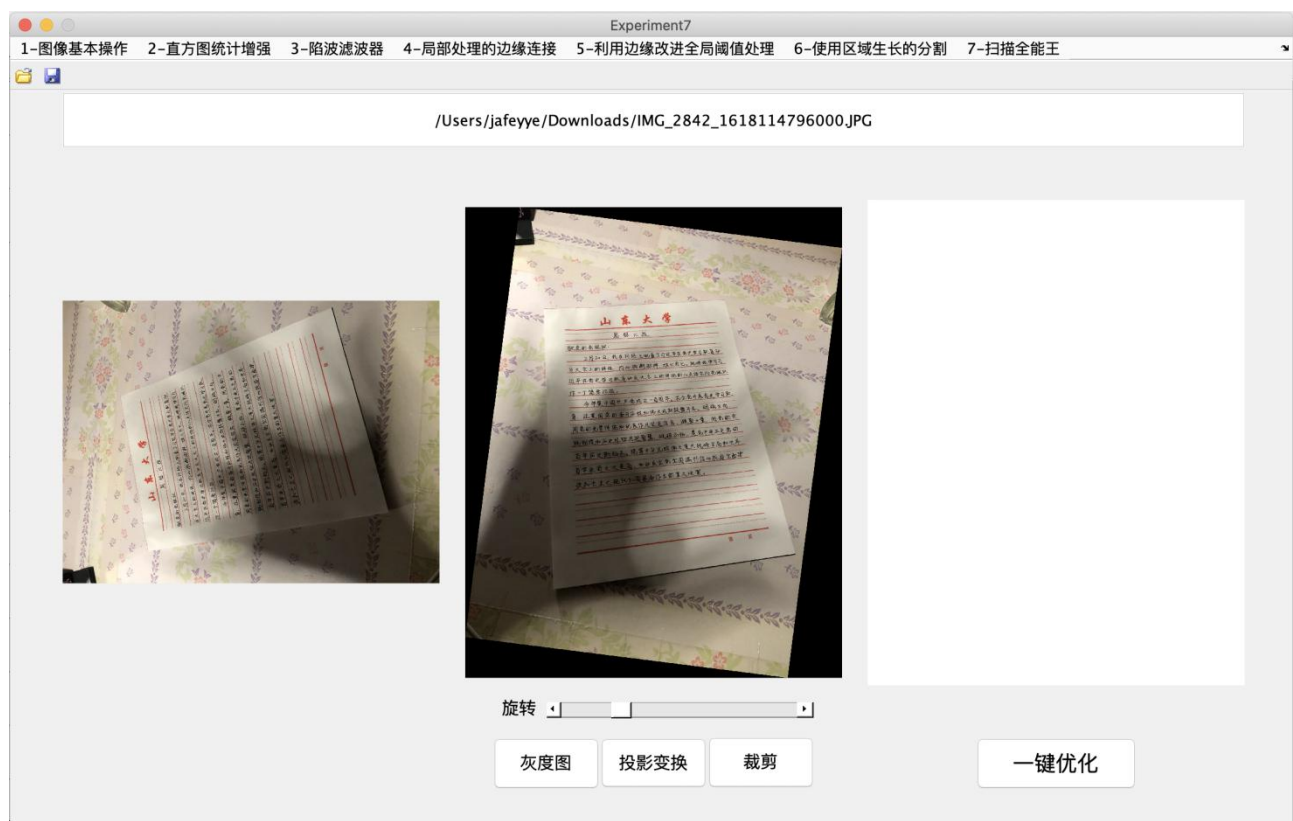
```
% -----  
function uipushtool1_ClickedCallback(hObject, eventdata, handles)  
% hObject    handle to uipushtool1 (see GCBO)  
% eventdata  reserved - to be defined in a future version of MATLAB  
% handles    structure with handles and user data (see GUIDATA)  
[filename,pathname]=uigetfile({'*.bmp;*.jpg;*.png;*.jpeg;*.tif'}, '请选择图片');  
str=[pathname filename];  
OriginalPic = imread(str);  
axes(handles.axes1);  
  
imshow(OriginalPic);  
set(handles.edit1, 'String', str);
```


2.2 图像的旋转

图像的旋转通过一根滑动条来实现（由于用户一般不知道把图像摆正所需要旋转的确切角度，只能靠不断调整来确定，所以没有设置可编辑文本框供用户输入旋转角度）。用户可在-180 度到 180 度之间旋转图像，直到将图像基本摆正，以方便下一步的投影变换。（实验表明，不经过旋转而直接进行投影变换也无伤大雅，不过可能会对图像形成挤压、造成形变，影响最终的成像效果。）

滑动条的回调函数如下图所示。首先读取左侧图像作为待处理图像，然后从滑动条中读取数据作为旋转的参数，调用 Matlab 的 `imrotate()` 函数执行旋转操作。注意这里不添加其它参数，这样就不会对图像进行裁剪，能保留图像的完整信息。最后将旋转后的图像显示在中间区域。

```
% --- Executes on slider movement.  
function slider1_Callback(hObject, eventdata, handles)  
% hObject    handle to slider1 (see GCBO)  
% eventdata  reserved - to be defined in a future version of MATLAB  
% handles    structure with handles and user data (see GUIDATA)  
% Hints: get(hObject,'Value') returns position of slider  
%         get(hObject,'Min') and get(hObject,'Max') to determine range of slider  
A = getimage(handles.axes1);  
data = get(handles.slider1,'value');  
B = imrotate(A,data);  
axes(handles.axes2);  
imshow(B);
```



2.3 图像的灰度转化

要将彩色图像转化为灰度图像，可以使用 `rgb2gray()` 函数。完整的回调函数如下。首先对图像进行判定，如果其通道数为 3，则将其转化为灰度图，否则不进行处理，以免报错。

```
% --- Executes on button press in pushbutton4.  
function pushbutton4_Callback(hObject, eventdata, handles)  
% hObject    handle to pushbutton4 (see GCBO)  
% eventdata  reserved - to be defined in a future version of MATLAB  
% handles    structure with handles and user data (see GUIDATA)  
A = getimage(handles.axes2);  
s = size(A);  
if s(3) == 3  
    A = rgb2gray(A);  
end  
axes(handles.axes2);  
imshow(A);
```



2.4 图像的投影变换

在图像投影变换的部分，关键在于找到原图与目标图之间的一系列对应点。这一步需要与用户交互完成。当用户点击“投影变换”，会出现十字光标，需要依次序点击目标图像的“左上-右上-左下-右下”四个点的位置，然后按下回车键，程序就会执行运算。具体的回调函数如下图所示。首先读取中间区域的图像，并获得图像的大小。然后通过交互得到目标区域的左上、右上、左下、右下四个点的坐标，分别将它们对应到新图的四个角。通过 `calc_homography()` 函数计算出这种对应下的单应映射矩阵，再调用自带的 `maketform()` 函数和 `imtransform()` 函数计算出投影矩阵并由此完成投影。最后将投影后的结果显示在中间的图像区域。


```

% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
img=getimage(handles.axes2);
[H W]=size(img);
p1=ginput();          %依次点击左上、右上、左下、右下
p2=[1,1;W,1;1,H;W,H];

T=calc_homography(p1,p2); %计算单应性矩阵
T=makeform('projective',T); %投影矩阵
[imgn X Y]=imtransform(img,T); %投影

axes(handles.axes2);
imshow(imgn,[]);

```

函数 `calc_homography()` 的代码如下。只需要根据点（分别都是矩阵）的映射套用 1.3 中的公式，则投影变换矩阵的每一项就呼之欲出了。

```

function T = calc_homography(points1, points2)

xaxb = points2(:,1) .* points1(:,1);
xayb = points2(:,1) .* points1(:,2);
yaxb = points2(:,2) .* points1(:,1);
yayb = points2(:,2) .* points1(:,2);

A = zeros(size(points1, 1)*2, 9);
A(1:2:end,3) = 1;
A(2:2:end,6) = 1;
A(1:2:end,1:2) = points1;
A(2:2:end,4:5) = points1;
A(1:2:end,7) = -xaxb;
A(1:2:end,8) = -xayb;
A(2:2:end,7) = -yaxb;
A(2:2:end,8) = -yayb;
A(1:2:end,9) = -points2(:,1);
A(2:2:end,9) = -points2(:,2);

[junk1,junk2,V] = svd(A);
h = V(:,9) ./ V(9,9);
T= reshape(h,3,3);

end

```



2.5 图像的裁剪

经过投影变换后，我们发现虽然图像变“正”了，但也变“小”了，因此我们需要裁剪出目标区域，以使其处于主要地位，而忽略其它次要区域。

我们可以通过与用户交互来实现，用户需要依次点击文档区域的左上角和右下角（默认文档区域为矩形，所以只需确定对角线即可），然后按下回车键，程序就会执行运算。具体的回调函数如下图所示。首先读取中间区域的投影变换后的图像，再用 edit 矩阵来存储用户点击的前两个点，并以此确定裁剪区域与原图像四周的距离，然后调用 `imcrop()` 函数来对图像进行裁剪，并将选定的裁剪区域用线宽为 2 的红色矩形显示出来。最后将新图显示在中间图像区域。

```
% --- Executes on button press in pushbutton5.  
function pushbutton5_Callback(hObject, eventdata, handles)  
% hObject    handle to pushbutton5 (see GCBO)  
% eventdata  reserved - to be defined in a future version of MATLAB  
% handles    structure with handles and user data (see GUIDATA)  
A = getimage(handles.axes2);  
edit = ginput();  
rect = [edit(1,1) edit(1,2) edit(2,1) edit(2,2)];  
B=imcrop(A,rect);  
rectangle('Position',rect,'LineWidth',2,'EdgeColor','r');  
axes(handles.axes2);  
imshow(B);
```

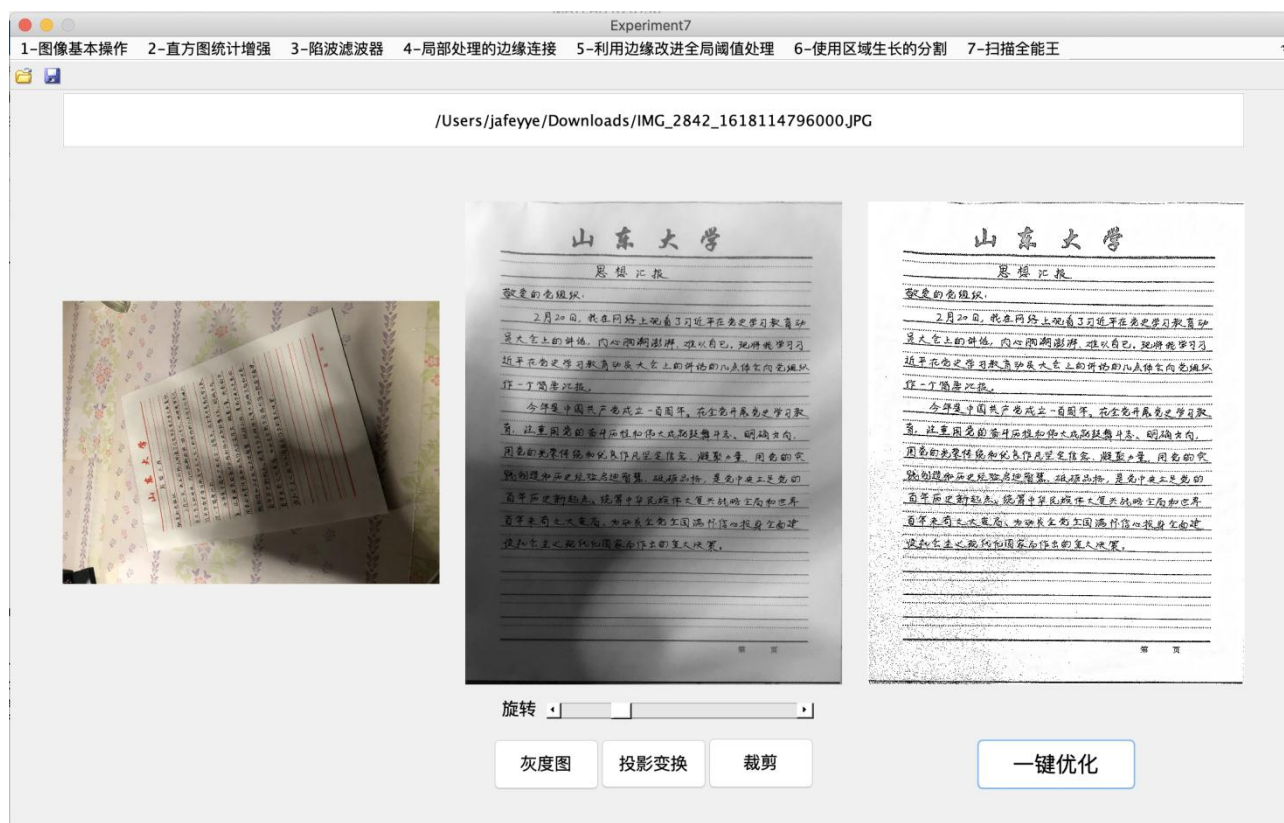


2.6 图像增强

图像增强方案的具体步骤已经在 1.5 中写明，下图回调函数中的注释也表明了每一行代码的具体任务。前四行是参数设置。参数的具体大小是经过一定的控制变量实验最终确定的，下图给出的参考值不一定最优但也一定较优。然后根据设置的参数设定滤波器进行滤波、根据设定的阈值进行筛选、根据文字增强参数进行文字区域的增强。最后将处理结果显示在右侧图像区，此即为最终的处理结果。

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
nAmount=101;      % 滤波半径
sigma=3;          % 高斯滤波方差
threshold=0.04;   % 低对比度mask阈值, 可根据下面src的直方图调整
amount=20;        % 文字增强的参数

I=double(getimage(handles.axes2));
w1=fspecial('average',[nAmount,nAmount]);      % 均值滤波
avg=imfilter(I,w1);                            % 利用均值滤波消除灰色区域
src=I./avg;                                     % 高斯滤波
w2=fspecial('gaussian',[nAmount,nAmount],sigma);
imgBlurred=imfilter(src,w2);
lowContrastMask = abs(src-imgBlurred)<threshold; % 按阈值筛选低对比度区域
dst = src*(1+amount)+imgBlurred*(-amount);      % 减去高斯滤波结果
dst = double(src.*lowContrastMask+dst.*(~lowContrastMask)); % 原代码中copyTo函数的实现
res = im2uint8(dst);
axes(handles.axes3);
imshow(dst);
```



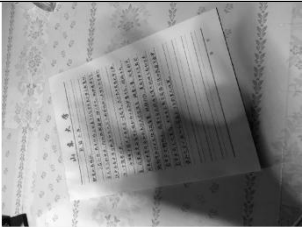
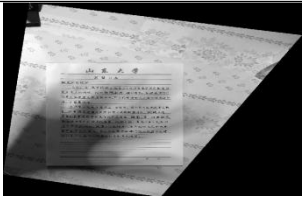

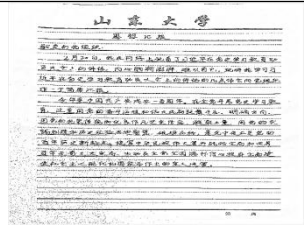
三、实验结果与总结

3.1 微信小程序的实现

由于微信小程序难以读取图像矩阵，无法实现离线处理，所以这里采取与后端服务器通信的方式进行在线处理，以本地计算机作为测试服务器。具体成果以 4 月 15 日的课堂展示为准。

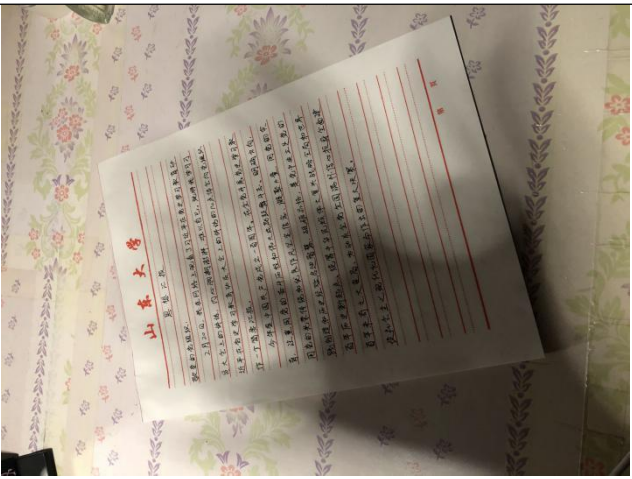
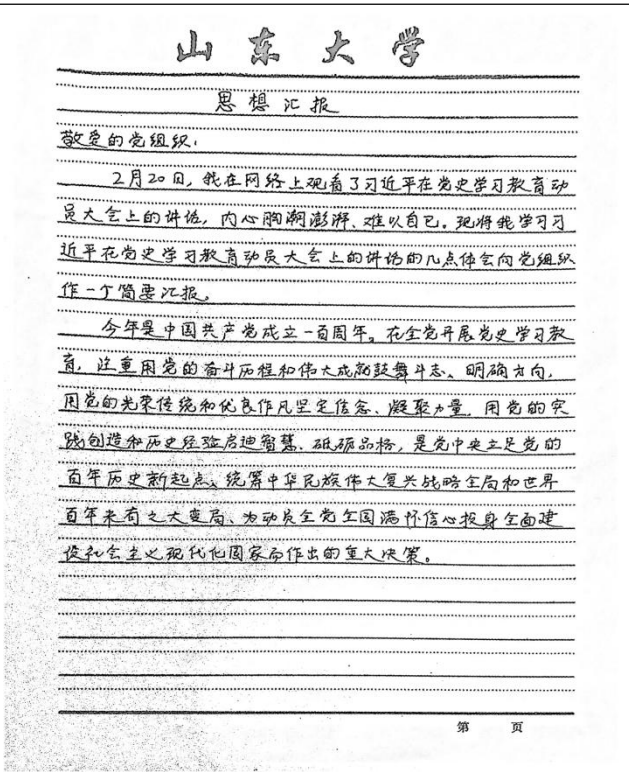
3.2 一些问题的探讨

前文提到，“不经过旋转而直接进行投影变换也无伤大雅，不过可能会对图像形成挤压、造成形变，影响最终的成像效果。”下面就来看一下舍去旋转这一步会有什么结果。

灰度图像	投影变换	裁剪	最终结果
			

可以发现，图像确实是被挤压了，这在一定程度上影响了观感。

3.3 最终处理结果的展示

	
---	--

