General workflow:

- 1. convert PDF to gray scale images
    - [x] 1.1 for local version just make sure the pdf is in an empty folder
    - [ ] 1.2 needs some design for online service version;
        - 1.2.1 make sure that only one user at a time by throw back 'server in use' (can check on multiuser app later)
        - 1.2.2 save the user-uploaded pdf; convert and save images to a destinated folder; clear folder after task
        - 1.2.3 need a timer to monitor and shut down inactive user; clear folder if confirmed inactive
- 1. establish template
    - [x] 2.1 build template for alignment and response reading
    - [x] 2.2 by using pre-made form, just pick the first image as template, pick 4 corner circles and establish ALIGNMENT_PARAM **probably can square it by ave(x), ave(y)**
    - [x] 2.3 align self, assert alignment fail other than move on if issue occurs
    - [x] 2.4 set ROIs for id, answers. currently hard coded, need gui for manual pick and/or ML for autodetect
    - [ ] 2.5 find the ID box (centroid, w, h) as orientation check point. ML for autodetect
- 1. process images and combine results for each image,
    - [ ] 3.1 *align image*; use the ID box to check orientation, rotate 180 if necessary; *try to pick out the 4 corner circles by fine-tuning params;*
        - [x] i. if fail, image with last attempt saved as 'e'; move on to next image
        - [x] ii. if good, 'alignment succeeded'
    - 3.2 extract responses; check ROIs (ans and ids combined) for filled answers;
        - [x] i. if non-fill or multi-fill in any entry, entry with quetionalbe answers are highlighted, image save as 'b'; move on to next image
        - [x] ii. if good, extracted answers highlighted, image save as 'a'
- 1. generate output files
    - [ ] 3.1 if there are errors, report all errors (error ID, image name, error name), label and save error images in destinated folder; early termination if more than 10 errors detected.
    - [ ] 3.2 save output as a .csv file for BubbleProcess

some thoughts for future improvement:

1. directory check and reuse
2. rotate up-side-down pictures before alignment
3. save grading results on image and combine to pdf
4. autoadjust region of interest(ROI)
5. exam analysis

developer journal

- 2019.9.21 framework established. quite a few misses and false results. need to save result image along the way.

3 images processed in 37066.910 ms. with 3 success and 0 errors.
['Image001;B;E;C;A;D;D;D;D;E;C;C;B;C;C;D;C;D;C;E;B;A;B;C;C;D;B;A;D;A;C;D;A;C;B;A;A;;B;;;A',
'Image002;B;E;E;D;D;E;D;E;E;B;C;B;A;C;D;A;E;D;C;C;B;B;A;C;A;B;;B;C;;;B;;;;A;;;D;B;',
'Image003;D;E;E;A;D;D;C;D;E;B;C;B;B;C;D;B;E;A;B;B;D;A;B;C;D;A;A;C;A;A;D;B;A;D;D;A;;D;;E;'] []

- 2019.9.22 adjust processing logic. add read ids. now 3 misses in 13 * (35 + 3) readings. saving outputs along the way is not slowing the program down.
  - before adjusting threshhold to fix the misses. implement catcher for exams that have either 0 or more than 1 answers for any question. if 0 answer, circle all options the program checked; if more than 1, circle all answers

13 images processed in 159644.334 ms. with 13 success and 0 errors.
['Image001;B;E;C;A;D;D;D;D;E;C;C;B;C;C;D;C;D;C;E;B;A;B;C;C;D;B;A;D;A;C;D;A;C;B;A;0;1;5',
'Image002;B;E;E;D;D;E;D;E;E;B;C;B;A;C;D;A;E;D;C;C;B;B;A;C;A;B;;B;C;;;B;;;;0;8;1',
'Image003;D;E;E;A;D;D;C;D;E;B;C;B;B;C;D;B;E;A;B;B;D;A;B;C;D;A;A;C;A;A;D;B;A;D;D;0;3;4', ...

catcher implemented: 13 images processed in 161.699 s (12.44 s/image). with 13 success and 0 errors.

- 2019.9.23 almost done with process_imgs. make find_corners harsher to force misalignment and errors. the catcher seems working. the Hough circle method is not very robust, probably need countours + area filter -> weight centroid instead.
  - new find_corners implemented. alignment looks neat and sharp now.

13 images processed in 160.509 s (12.35 s/image). with 13 success and 0 errors.

In [258]:
```python
# Standard imports
import tempfile
import os
import time
from pdf2image import convert_from_path, convert_from_bytes
from pdf2image.exceptions import (
    PDFInfoNotInstalledError,
    PDFPageCountError,
    PDFSyntaxError
)
from fpdf import FPDF
%matplotlib inline
from scipy.signal import convolve2d
import argparse
import cv2
from matplotlib import pyplot as plt
from matplotlib import gridspec
import pandas as pd
import numpy as np;
```

In [144]:
```
# parameters
PDFINPUT = 'test.pdf'

# PATH = os.path.dirname(os.path.realpath('__file__'))
SRC_IMG = 'srcimg'
OUT_IMG = 'outimg'
# TEMPLATE = "{}\Image001.jpg".format(os.path.join(PATH, SRC_IMG))

IDS = [[3,'0123456789',1]]
ANSWERS = [[35,'ABCDE',0]]
```

In [274]:
```
def convert_pdf(PDFINPUT, srcimg):
    os.makedirs(srcimg, exist_ok=True)
    for root, dirs, files in os.walk(srcimg, topdown=False):
        for name in files:
            os.remove(os.path.join(root, name))
    with tempfile.TemporaryDirectory() as path:
        images_from_path = convert_from_path(PDFINPUT, output_folder=path) #,
 grayscale=True # cause weird racing problem
        cnt = 0
        for image in images_from_path:
            cnt += 1
            image.save("{}\Image{:03d}.jpg".format(srcimg, cnt), "JPEG")
    return cnt
# convert_pdf('test.pdf', srcimg)
```

Out[274]:
```
13
```

In [256]:
```
# build alignment
def get_corners2(im):
    print('get corners2')
    gc = im.copy()
    gc[220:1900] = 255
    gc[:,320:1400] = 255
    _, gc = cv2.threshold(gc, 127, 255, cv2.THRESH_BINARY_INV)
    kernel = np.ones((12,12),np.uint8)
    gc = cv2.erode(gc,kernel,iterations = 1)
    gc = cv2.GaussianBlur(gc, (7, 7), 0)
#     _, gc = cv2.threshold(gc, 127, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)
#     gc = cv2.Canny(gc,100,200)
    circles = cv2.HoughCircles(gc, cv2.HOUGH_GRADIENT,
                               dp=0.9, minDist=500, param1 = 42,
                               param2 = 28, minRadius = 30,
                               maxRadius = 50)
#     print(circles)
    if (circles is None or len(circles[0]) < 4):
        circles = cv2.HoughCircles(gc, cv2.HOUGH_GRADIENT,
                               dp=0.9, minDist=500, param1 = 40,
                               param2 = 28, minRadius = 60,
                               maxRadius = 85)
    elif (len(circles[0]) > 4):
        circles = cv2.HoughCircles(gc, cv2.HOUGH_GRADIENT,
                               dp=1, minDist=500, param1 = 45,
                               param2 = 30, minRadius = 60,
                               maxRadius = 85)


#     print(circles)

# #     Visual debug block
#     img_circled = cv2.cvtColor(im.copy(), cv2.COLOR_GRAY2BGR)
#     for i in range(circles.shape[1]):
#             c = circles[0, i]

#             cv2.circle( img_circled, (c[0], c[1]), c[2], (0, 255, 0), 4)
#     if circles is not None:
#         print("Marked")
#     else:
#         print("circle is None")
#     fig = plt.figure(figsize = (18,15))

#     fig.add_subplot(121)
#     plt.imshow(gc, cmap = 'gray')
#     fig.add_subplot(122)
#     plt.imshow(img_circled)

    df = circles[0][:,:2]
    df = df[df[:,0].argsort()]
#     print(df)
    a, b = df[:2], df[2:]
    a = a[a[:,1].argsort()]
    b = b[b[:,1].argsort()]
    df = np.vstack([a,b])
    return df
```

```
def get_corners(im):
    print('get corners')
#     g = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
    gc = im.copy()
    gc[220:1900] = 255
    gc[:,320:1400] = 255

    _, gc = cv2.threshold(gc, 127, 255, cv2.THRESH_BINARY_INV)
    kernel = np.ones((5,5),np.uint8)
    gc = cv2.erode(gc,kernel,iterations = 3)
    gc = cv2.GaussianBlur(gc, (7, 7), 0)

    img_contoured = cv2.cvtColor(im.copy(), cv2.COLOR_GRAY2BGR)
    _, contours,hierarchy = cv2.findContours(gc.copy(), cv2.RETR_EXTERNAL, cv2
.CHAIN_APPROX_SIMPLE)
#     print(len(contours))

#     cv2.drawContours(img_contoured, contours, -1, (255,0,0), 2)

#     img_fitted = cv2.cvtColor(im.copy(), cv2.COLOR_GRAY2BGR)
#     cv2.drawContours(img_fitted, contours, -1, (255,0,0), -1)
    circles = []
    for c in contours:
        (x,y), r = cv2.minEnclosingCircle(c)
        x,y,r = int(x), int(y), int(r)
#         cv2.circle(img_fitted, (x,y),r,(0,255,0),2)
        circles.append([x,y,r])

    print(circles)

#     fig = plt.figure(figsize = (18,15))

#     fig.add_subplot(121)
#     plt.imshow(img_contoured, cmap = 'gray')
#     fig.add_subplot(122)
#     plt.imshow(img_fitted, cmap='gray')

    circles = np.array(circles)
    df = circles[:,:2]
    df = df[df[:,0].argsort()]
#     print(df)
    a, b = df[:2], df[2:]
    a = a[a[:,1].argsort()]
    b = b[b[:,1].argsort()]
    df = np.vstack([a,b])
    return df

def debug_get_corners():
    srcimg = os.path.join(PATH, SRC_IMG)
    firstImg = cv2.imread('{}\Image001.jpg'.format(srcimg), 0)
    for i, filename in enumerate(os.listdir(srcimg)):
        print(i+1, filename)
        if i > 2: break
        img = cv2.imread("{}\{}".format(srcimg,filename),0)
        get_corners(img)
```

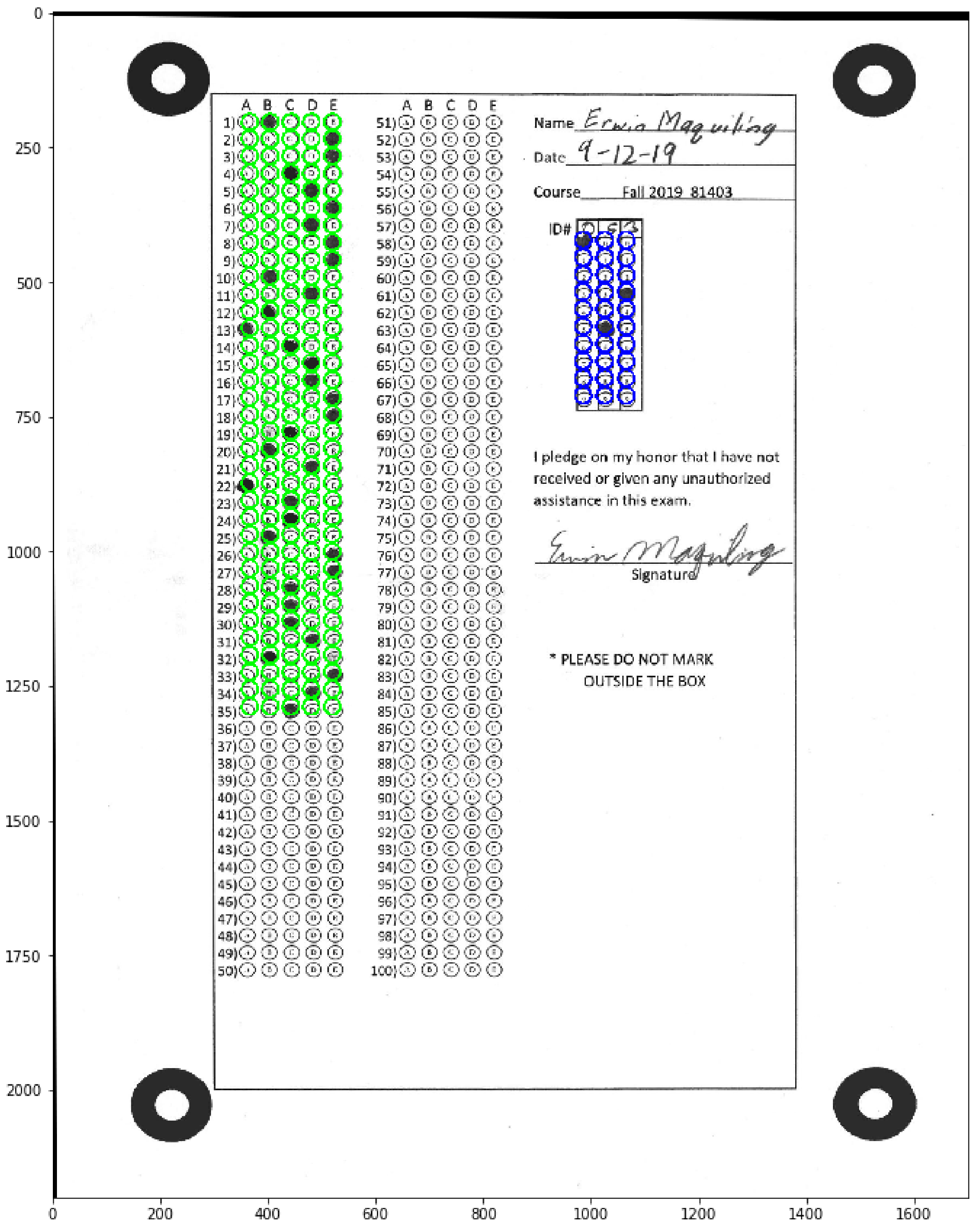
```python
# debug_get_corners()

def build_alignment(img):
    '''
    find 4 corner points, return the averaged 4 points
    '''
    df = get_corners(img)
    print(df)
    assert(len(df) == 4), 'fail to find 4 corners to build alignment'
    x1, x2 = (df[0][0] + df[1][0]) / 2, (df[2][0] + df[3][0]) / 2
    y1, y2 = (df[0][1] + df[2][1]) / 2, (df[1][1] + df[3][1]) / 2

    df = np.array([(x1,y1), (x1,y2), (x2,y1),(x2,y2)])

    print(df, '\n*****  alignment established')

    return df

# srcimg = os.path.join(PATH, SRC_IMG)
# ti = cv2.imread('{}\Image001.jpg'.format(srcimg), 0)
# arr = build_alignment(ti)
# cti = cv2.cvtColor(ti, cv2.COLOR_GRAY2BGR)
# for a in arr:
#     cv2.circle(cti, (int(a[0]), int(a[1])), 60, (255,0,0), 5)

# plt.figure(figsize = (15,15))
# plt.imshow(cti, cmap='gray');
```

In [126]:
```
# building template
def build_align_img(img, alignment):
    print('build_align_img')
    points1 = get_corners(img)
    points2 = alignment
    print(points1)
    print(points2)

    if (points1 is None or len(points1) != 4):

        if points1 is not None:
            img_circled = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
            for i in range(len(points1)):
                c = points1[i]

                cv2.circle( img_circled, (c[0], c[1]), 60, (0, 255, 0), 4)
        #          print("i = %d, r = %f" % (i, c[2]))

            print("Circles Marked")
            plt.figure(figsize = (15,15))

            plt.imshow(img_circled, cmap='gray');

        else:
            print("circle is None")
        assert(False), "alignment failed!!"

    # Find homography
    h, mask = cv2.findHomography(points1, points2, cv2.RANSAC)

    # Use homography
    if len(img.shape) == 3:
        height, width, _ = img.shape
    else:
        height, width = img.shape
    im1Reg = cv2.warpPerspective(img, h, (width, height))
#     plt.imshow(im1Reg);

    return im1Reg, h

def debug_align_img():
    srcimg = os.path.join(PATH, SRC_IMG)
    ti = cv2.imread('{}\Image001.jpg'.format(srcimg), 0)
    aimg, h = align_img(ti, build_alignment(ti))
    cti = cv2.cvtColor(ti, cv2.COLOR_GRAY2BGR)
    ret, mask = cv2.threshold(ti, 127, 255,cv2.THRESH_BINARY_INV)
    aimg = cv2.cvtColor(aimg, cv2.COLOR_GRAY2BGR)
    print(cti[mask == 255].shape, cti.shape, aimg.shape)
    cti[mask == 255] = [0,0,255]
    # cti[mask == 255] = (0,0,255)
    # for a in arr:
    #     cv2.circle(cti, (int(a[0]), int(a[1])), 60, (255,0,0), 5)
    weighted = cv2.addWeighted(cti, 0.6, aimg, 0.4, 0)
    plt.figure(figsize = (15,15))
    plt.imshow(weighted, cmap='gray');
# debug_align_img()
```

```
def build_roi(img, item):
    radius = 15
    if (item[2] == 0):
        top_left, bottom_right = (365, 204), (518, 1288) # manual input vs. au
todetect
        entries, options, orientation = item
        xs = np.linspace(top_left[0],bottom_right[0],len(options)).astype(int)
        ys = np.linspace(top_left[1],bottom_right[1],entries).astype(int)
        circles = np.array([[x,y,radius] for y in ys for x in xs])
    else:
        top_left, bottom_right = (983, 423), (1063, 711) # manual input vs. au
todetect
        entries, options, orientation = item
        xs = np.linspace(top_left[0],bottom_right[0],entries).astype(int)
        ys = np.linspace(top_left[1],bottom_right[1],len(options)).astype(int)
        circles = np.array([[x,y,radius] for x in xs for y in ys])
    print('building roi',circles.shape,circles[0],circles[-1])

    return circles

def debug_build_roi():
    srcimg = os.path.join(PATH, SRC_IMG)
    ti = cv2.imread('{}\Image001.jpg'.format(srcimg), 0)
    aimg, h = align_img(ti, build_alignment(ti))
    cv2.imwrite('alignedTemp.jpg',aimg)
    ai = cv2.cvtColor(aimg, cv2.COLOR_GRAY2BGR)
#     cs = build_roi(ti, ANSWERS[0])
    cs = build_roi(ti, IDS[0])
    for c in cs:
        cv2.circle(ai,(c[0],c[1]),c[2],(0,255,0),-1)
    plt.figure(figsize=(10,10))
    plt.imshow(ai, cmap='gray')
    ######################
    # Manually search tl, br corners:
    # cv2.circle(aimg, (365,204), 13, (0,255,0), -1)
    # cv2.circle(aimg, (518,1288), 13, (0,255,0), -1)
    # plt.figure(figsize=(8,8))
    # # plt.imshow(aimg, cmap='gray')
    # # plt.imshow(aimg[190:220,350:380], cmap='gray')
    # plt.imshow(aimg[1250:1310,300:580], cmap='gray')
    return
# debug_build_roi()

def build_template(img, alignment, IDS, ANSWERS):
    '''
    align img and build ROIs for ID, Ans (list of circles to scan)
    '''
#     img = cv2.imread(file, 0)

    img, h = build_align_img(img, alignment)
    ans = [build_roi(img, ans) for ans in ANSWERS]
    ids = [build_roi(img, _) for _ in IDS]

    print('build_template succeed!')
    return ids, ans
```

```
def debug_build_template():
    firstImg = cv2.imread('{}\Image001.jpg'.format(srcimg), 0)
    alignment = build_alignment(firstImg)
    testImg = cv2.imread("{}\Image004.jpg".format(os.path.join(PATH, SRC_IMG
)), 0)
    ids, ans = build_template(testImg, alignment, IDS, ANSWERS)
    print(np.array(ids).shape, np.array(ans).shape)
    img = testImg.copy()
    img, h = align_img(img, alignment)
    img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
    for i in ids[0]:
        cv2.circle(img, (i[0], i[1]), i[2], (0,0,255), 3)
    for i in ans[0]:
        cv2.circle(img, (i[0], i[1]), i[2], (0,255,0), 3)
    plt.figure(figsize=(15,15))
    plt.imshow(img, cmap='gray')
debug_build_template()
```

```
get corners
[[ 212.5  123.5]
 [ 231.5 2023.5]
 [1518.5  124.5]
 [1524.5 2023.5]]
[[ 222.   124. ]
 [ 222.  2023.5]
 [1521.5  124. ]
 [1521.5 2023.5]]
*****  alignment established
build_align_img
get corners
[[ 217.5  113.5]
 [ 213.5 2006.5]
 [1504.5  107.5]
 [1508.5 2007.5]]
[[ 222.   124. ]
 [ 222.  2023.5]
 [1521.5  124. ]
 [1521.5 2023.5]]
building roi (175, 3) [365 204  15] [ 518 1288   15]
building roi (30, 3) [983 423  15] [1063  711   15]
build_template succeed!
(1, 30, 3) (1, 175, 3)
align_img
get corners
[[ -4.5 -10.5]
 [ -8.5 -17. ]
 [-17.  -16.5]
 [-13.  -16. ]]
```


0
250
500
750
1000
1250
1500
1750
2000
0
200
400
600
800
1000
1200
1400
1600
A B C D E
A B C D E

Name Erwin Maquiling

Date 9-12-19

Course Fall 2019 81403

ID#

I pledge on my honor that I have not received or given any unauthorized assistance in this exam.

[illegible]

Signature

* PLEASE DO NOT MARK OUTSIDE THE BOX

In [267]:
```
# process img
def align_img(img, alignment):
    print('align_img')
    points1 = get_corners(img)
    points2 = alignment

    if (points1 is None or len(points1) != 4):
        img_circled = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
        if points1 is not None:
            for i in range(len(points1)):
                c = points1[i]
                cv2.circle(img_circled, (c[0], c[1]), 60, (0, 255, 0), 4)
        else:
            cv2.putText(img_circled, "Cannot find any corner!!!",
                        (200,200), cv2.FONT_HERSHEY_SIMPLEX, 5, (0,0,255),
                        10, cv2.LINE_AA)
        return img_circled, []

    print(points1-points2)

    # Find homography
    h, mask = cv2.findHomography(points1, points2, cv2.RANSAC)

    # Use homography
    if len(img.shape) == 3:
        height, width, _ = img.shape
    else:
        height, width = img.shape
    im1Reg = cv2.warpPerspective(img, h, (width, height))

    return im1Reg, h

def process_img(img, rois, items, outimg):
    print('processing image...')
    res = []
    t = 'a'
    circled_img = cv2.cvtColor(img.copy(), cv2.COLOR_GRAY2BGR)
    for roi, item in zip(rois, items):
#         circles_rounded = np.uint16(np.around(circles))[0]
        circles_sorted = roi # circles_rounded[circles_rounded[:,1].argsort()]

        circle_radius = circles_sorted[...,-1].min()
        kernel = np.ones((2*circle_radius,2*circle_radius),dtype=int)
        out0 = convolve2d(255-img, kernel,'same')
        detected_vals = out0[circles_sorted[...,1], circles_sorted[...,0]]
        detected_vals -= detected_vals.min()
        mask = detected_vals>detected_vals.max()/2

        ans = []
        choices = item[1]

        for i in range(0, len(circles_sorted), len(choices)):
            a, v = '', 0
            cnt = mask[i:i+len(choices)].sum()
#               print(int(i/len(choices)), cnt)
            if cnt != 1:
                t = 'b'
                c1, c2 = circles_sorted[i], circles_sorted[i+len(choices)-1]
                if c1[0] == c2[0]: # vertical
                    p1, p2 = (c1[0]+c1[2]+3, c1[1]-c1[2]), (c2[0]+c2[2]+3, c2[
1]+c2[2]*3)
                else: # horizontal
                    p1, p2 = (c1[0]-c1[2], c1[1]+c1[2]+3), (c2[0]+c2[2]*3, c2[
1]+c2[2]+3)
                cv2.line(circled_img,p1,p2,(255,0,0),1)
                cv2.putText(circled_img, '*', p2, cv2.FONT_HERSHEY_SIMPLEX,
                            1, (255,0,0), 3, cv2.LINE_AA)
            if cnt == 0:
                for j in range(len(choices)):
                    c = circles_sorted[i+j]
                    cv2.circle(circled_img, (c[0],c[1]), c[2], (0,205,0), 1)

            elif cnt > 1:
                for j in range(len(choices)):
                    if mask[i+j]==1:
                        c = circles_sorted[i+j]
                        a = choices[j] if detected_vals[i+j] > v else a
                        v = detected_vals[i+j] if detected_vals[i+j] > v else
v
                        cv2.circle(circled_img, (c[0],c[1]), c[2], (0,255,0),
3)
                c = circles_sorted[i+choices.find(a)]
                cv2.putText(circled_img, a, (c[0], c[1]), cv2.FONT_HERSHEY_SIM
PLEX,
                            1, (0,0,255), 3, cv2.LINE_AA)

            else:
                for j in range(len(choices)):
                    if mask[i+j]==1:
                        c = circles_sorted[i+j]
                        a = choices[j]
                        cv2.circle(circled_img, (c[0],c[1]), c[2], (0,255,0),
3)
                        cv2.putText(circled_img, a, (c[0], c[1]), cv2.FONT_HER
SHEY_SIMPLEX,
                                    1, (0,0,255), 3, cv2.LINE_AA)

            ans.append(a)

        res.extend(ans)
    if t == 'b':
        cv2.putText(circled_img, '***', (320,130), cv2.FONT_HERSHEY_SIMPLEX,
                    4, (255,0,0), 8, cv2.LINE_AA)
    return res,t, circled_img

def process_imgs(srcimg, alignment, ids, IDS, ans, ANSWERS, outimg, image_scan
ned):
    # fetch each image
    start = time.time()
    print('processing images ... ...')
    res, err = [], []
    rois, items = [], []
    rois.extend(ans)
    rois.extend(ids)
    items.extend(ANSWERS)
    items.extend(IDS)
    for i, filename in enumerate(os.listdir(srcimg)):
        print(i+1, filename)
#         if i > 4: break
        img = cv2.imread("{}\{}".format(srcimg,filename),0)
        img, _ = align_img(img, alignment)
        if len(_) == 0:
            err.append(filename)
            assert (len(err) < 5), '\nAlignment on five (5) or more images fai
led.\
            \nPlease check input file and resubmit.'
#             cv2.putText
            cv2.imwrite("{}\{}{}".format(outimg,'e',filename), img)
            continue
        r, t, proc_img = process_img(img, rois, items, outimg)
        entry = [t+filename[:-4]]
        entry.extend(r)
        res.append(";".join(entry))
        cv2.imwrite("{}\{}{}".format(outimg,t,filename), proc_img)
    print('''{} images processed in {:.03f} s ({:.02f} s/image).
    with {} success and {} errors.'''.format(
        len(res) + len(err), (time.time()-start),
        (time.time()-start)/(len(res) + len(err)),
        len(res), len(err)))
    print(res, err)
    return res, err

def debug_process_imgs():
    srcimg = os.path.join(PATH, SRC_IMG)
    outimg = os.path.join(PATH, OUT_IMG)
    firstImg = cv2.imread('{}\Image001.jpg'.format(srcimg), 0)
    ali = np.array([[ 222.,   124. ], [ 222.,  2023.5], [1521.5,  124. ], [152
1.5, 2023.5]])
    ids, ans = build_template(firstImg, ali, IDS, ANSWERS)
    res, err = process_imgs(srcimg, ali, ids, IDS, ans, ANSWERS, outimg, 13)
# debug_process_imgs()
```

```
build_align_img
get corners
[[223, 2024, 70], [1516, 2023, 73], [216, 125, 71], [1510, 124, 73]]
[[ 216  125]
 [ 223 2024]
 [1510  124]
 [1516 2023]]
[[ 222.   124. ]
 [ 222.  2023.5]
 [1521.5  124. ]
 [1521.5 2023.5]]
building roi (175, 3) [365 204  15] [ 518 1288   15]
building roi (30, 3) [983 423  15] [1063  711   15]
build_template succeed!
processing images ... ...
1 Image001.jpg
align_img
get corners
[[223, 2024, 70], [1516, 2023, 73], [216, 125, 71], [1510, 124, 73]]
[[ -6.    1. ]
 [  1.    0.5]
 [-11.5   0. ]
 [ -5.5  -0.5]]
processing image...
2 Image002.jpg
align_img
get corners
[[210, 2005, 71], [1505, 2003, 72], [208, 108, 71], [1499, 104, 71]]
[[-14.  -16. ]
 [-12.  -18.5]
 [-22.5 -20. ]
 [-16.5 -20.5]]
processing image...
3 Image003.jpg
align_img
get corners
[[224, 2024, 72], [1520, 2021, 72], [218, 125, 72], [1513, 124, 71]]
[[-4.   1. ]
 [ 2.   0.5]
 [-8.5  0. ]
 [-1.5 -2.5]]
processing image...
4 Image004.jpg
align_img
get corners
[[1510, 2009, 72], [211, 2012, 71], [209, 114, 71], [1506, 111, 72]]
[[-13.  -10. ]
 [-11.  -11.5]
 [-15.5 -13. ]
 [-11.5 -14.5]]
processing image...
5 Image005.jpg
align_img
get corners
[[223, 2026, 71], [1518, 2025, 73], [217, 128, 71], [1512, 127, 72]]
[[-5.   4. ]
 [ 1.   2.5]
 [-9.5  3. ]
 [-3.5  1.5]]
processing image...
6 Image006.jpg
align_img
get corners
[[223, 2023, 71], [1517, 2019, 73], [215, 123, 72], [1509, 122, 72]]
[[ -7.   -1. ]
 [  1.   -0.5]
 [-12.5  -2. ]
 [ -4.5  -4.5]]
processing image...
7 Image007.jpg
align_img
get corners
[[1516, 2020, 72], [223, 2024, 71], [217, 125, 72], [1510, 124, 71]]
[[ -5.    1. ]
 [  1.    0.5]
 [-11.5   0. ]
 [ -5.5  -3.5]]
processing image...
8 Image008.jpg
align_img
get corners
[[223, 2024, 71], [1519, 2021, 73], [216, 126, 72], [1512, 122, 72]]
[[-6.   2. ]
 [ 1.   0.5]
 [-9.5 -2. ]
 [-2.5 -2.5]]
processing image...
9 Image009.jpg
align_img
get corners
[[224, 2021, 71], [1521, 2019, 73], [217, 122, 71], [1514, 120, 73]]
[[-5.  -2. ]
 [ 2.  -2.5]
 [-7.5 -4. ]
 [-0.5 -4.5]]
processing image...
10 Image010.jpg
align_img
get corners
[[210, 2009, 71], [1507, 2007, 73], [208, 111, 71], [1502, 107, 73]]
[[-14.  -13. ]
 [-12.  -14.5]
 [-19.5 -17. ]
 [-14.5 -16.5]]
processing image...
11 Image011.jpg
align_img
get corners
[[223, 2019, 71], [1516, 2019, 73], [1512, 121, 72], [219, 122, 72]]
[[-3.  -2. ]
 [ 1.  -4.5]
 [-9.5 -3. ]
 [-5.5 -4.5]]
processing image...
```

```
12 Image012.jpg
align_img
get corners
[[1517, 2020, 73], [223, 2020, 72], [1513, 122, 72], [220, 122, 71]]
[[-2.  -2. ]
 [ 1.  -3.5]
 [-8.5 -2. ]
 [-4.5 -3.5]]
processing image...
13 Image013.jpg
align_img
get corners
[[1509, 2007, 74], [211, 2008, 71], [209, 111, 71], [1505, 107, 73]]
[[-13.  -13. ]
 [-11.  -15.5]
 [-16.5 -17. ]
 [-12.5 -16.5]]
processing image...
13 images processed in 167.876 s (12.91 s/image).
    with 13 success and 0 errors.
['aImage001;B;E;C;A;D;D;D;D;E;C;C;B;C;C;D;C;D;C;E;B;A;B;C;C;D;B;A;D;A;C;D;A;
C;B;A;0;1;5', 'aImage002;B;E;E;D;D;E;D;E;E;B;C;B;A;C;D;A;E;D;C;C;B;B;A;C;A;B;
E;B;C;C;D;B;C;D;C;0;8;1', 'bImage003;D;E;E;A;D;D;C;D;E;B;C;B;B;C;D;B;E;A;B;B;
D;A;B;C;D;A;A;C;A;A;D;B;A;D;D;0;3;4', 'aImage004;B;E;E;C;D;E;D;E;E;B;D;B;A;C;
D;D;E;E;C;B;D;A;C;C;B;E;E;C;C;C;D;B;E;D;C;0;5;3', 'bImage005;A;B;C;B;B;D;D;D;
E;A;B;C;B;B;C;B;C;A;B;B;C;E;C;C;D;B;E;D;C;C;D;A;A;C;D;0;8;', 'aImage006;A;E;
C;C;D;E;D;D;E;B;C;B;C;E;A;A;D;C;A;B;D;A;C;B;A;B;E;C;D;D;D;A;A;D;C;0;0;4', 'aI
mage007;B;E;E;D;D;D;D;E;E;B;C;B;A;C;D;A;D;E;E;C;B;B;A;C;E;B;E;C;C;C;D;B;E;D;
C;0;4;8', 'aImage008;A;E;E;D;D;E;A;C;E;D;C;C;A;A;D;D;E;C;C;C;D;B;A;C;A;B;E;C;
C;C;D;B;A;B;C;0;8;3', 'aImage009;C;E;E;D;D;B;D;E;E;B;C;B;A;C;D;B;D;E;E;C;B;B;
A;D;C;B;E;C;A;C;E;B;C;D;C;0;5;5', 'aImage010;B;E;E;D;D;E;D;E;E;B;C;B;A;C;D;A;
E;D;C;C;B;B;A;B;A;B;E;C;C;C;E;B;E;D;C;0;0;8', 'aImage011;B;E;C;D;D;E;D;E;E;B;
C;B;A;C;D;B;E;D;C;C;D;B;A;D;B;B;E;B;A;C;E;B;E;D;C;0;5;2', 'aImage012;C;E;E;D;
D;C;D;C;B;B;C;B;D;C;D;C;E;A;C;C;D;B;A;C;D;A;E;A;C;D;E;A;E;B;B;0;7;9', 'aImage
013;D;E;E;D;D;E;D;A;E;B;C;B;A;C;D;A;E;C;C;C;D;B;A;C;A;B;E;C;C;C;D;B;E;D;C;0;
1;7'] []
```

In [271]:
```
# generate report
def generate_report(res, err, outimg):
    a, b = [i for i in res if i[0] == 'a'], [i for i in res if i[0] == 'b']
    a.sort(key = lambda x: ''.join(x[-3:]))
    b.sort(key = lambda x: ''.join(x[-3:]))
    results = []
    results.extend(a)
    results.extend(b)
    results.extend(err)
    cover = cv2.imread("{}\{}.jpg".format(outimg,results[0][:9]))
    plt.figure(figsize=(8,8))
    plt.imshow(cover)
    print(cover.shape)
    w,h = cover.shape[1],cover.shape[0]
    pdf = FPDF(unit = 'pt', format = [w,h])
    for filename in results:
        pdf.add_page()
        pdf.image("{}\{}.jpg".format(outimg,filename[:9]),0,0,w,h)
    pdf.output("{}\\re.pdf".format(outimg), "F")
    df = pd.DataFrame(results)
    df.to_csv("{}\\re.csv".format(outimg),index = False,header=False)
    pass

def debug_generate_report():
    res, err = ['aImage001;B;E;C;A;D;D;D;D;E;C;C;B;C;C;D;C;D;C;E;B;A;B;C;C;D;
B;A;D;A;C;D;A;C;B;A;0;1;5', 'aImage002;B;E;E;D;D;E;D;E;E;B;C;B;A;C;D;A;E;D;C;
C;B;B;A;C;A;B;E;B;C;C;D;B;C;D;C;0;8;1', 'bImage003;D;E;E;A;D;D;C;D;E;B;C;B;B;
C;D;B;E;A;B;B;D;A;B;C;D;A;A;C;A;A;D;B;A;D;D;0;3;4', 'aImage004;B;E;E;C;D;E;D;
E;E;B;D;B;A;C;D;D;E;E;C;B;D;A;C;C;B;E;E;C;C;C;D;B;E;D;C;0;5;3', 'bImage005;A;
B;C;B;B;D;D;D;E;A;B;C;B;B;C;B;C;A;B;B;C;E;C;C;D;B;E;D;C;C;D;A;A;C;D;0;8;', 'aI
mage006;A;E;C;C;D;E;D;D;E;B;C;B;C;E;A;A;D;C;A;B;D;A;C;B;A;B;E;C;D;D;D;A;A;D;C;
0;0;4', 'aImage007;B;E;E;D;D;D;D;E;E;B;C;B;A;C;D;A;D;E;E;C;B;B;A;C;E;B;E;C;C;
C;D;B;E;D;C;0;4;8', 'aImage008;A;E;E;D;D;E;A;C;E;D;C;C;A;A;D;D;E;C;C;C;D;B;A;
C;A;B;E;C;C;C;D;B;A;B;C;0;8;3', 'aImage009;C;E;E;D;D;B;D;E;E;B;C;B;A;C;D;B;D;
E;E;C;B;B;A;D;C;B;E;C;A;C;E;B;C;D;C;0;5;5', 'aImage010;B;E;E;D;D;E;D;E;E;B;C;
B;A;C;D;A;E;D;C;C;B;B;A;B;A;B;E;C;C;C;E;B;E;D;C;0;0;8', 'aImage011;B;E;C;D;D;
E;D;E;E;B;C;B;A;C;D;B;E;D;C;C;D;B;A;D;B;B;E;B;A;C;E;B;E;D;C;0;5;2', 'aImage01
2;C;E;E;D;D;C;D;C;B;B;C;B;D;C;D;C;E;A;C;C;D;B;A;C;D;A;E;A;C;D;E;A;E;B;B;0;7;9'
, 'aImage013;D;E;E;D;D;E;D;A;E;B;C;B;A;C;D;A;E;C;C;C;D;B;A;C;A;B;E;C;C;C;D;B;
E;D;C;0;1;7'], []
    outimg = os.path.join(PATH, OUT_IMG)
    generate_report(res, err, outimg)
# debug_generate_report()
```

In [247]:
```
def main(PDFINPUT, SRC_IMG, OUT_IMG, IDS, ANSWERS):
    PATH = os.path.dirname(os.path.realpath('__file__'))
    srcimg = os.path.join(PATH, SRC_IMG)
    outimg = os.path.join(PATH, OUT_IMG)
    print("source images @ {}".format(srcimg))
    print("Output images @ {}".format(srcimg))
    t1 = time.time()
    image_scanned = convert_pdf(PDFINPUT, srcimg)
    t2 = time.time()
    print('{} pages converted into JPEG. {:.02f} s used. {:.02f} s/page'.forma
t(
        image_scanned, t2-t1, (t2-t1)/image_scanned))
    firstImg = cv2.imread('{}\Image001.jpg'.format(srcimg), 0)
    alignment = build_alignment(firstImg)

    ids, ans = build_template(firstImg, alignment, IDS, ANSWERS)

    res, err = process_imgs(srcimg, alignment, ids, IDS, ans, ANSWERS, outimg,
image_scanned)

    generate_report(res, err, outimg)
    t3 = time.time()
    print('Grading complete. {:.02f} s total. {:.02f} s/page'.format(
        t3-t1, (t3-t1)/image_scanned))
```

In [ ]:
```
main(PDFINPUT, SRC_IMG,OUT_IMG, IDS, ANSWERS)
```