Assignment 03: Feature Matching

Brock Davis

Part 0: Input Images

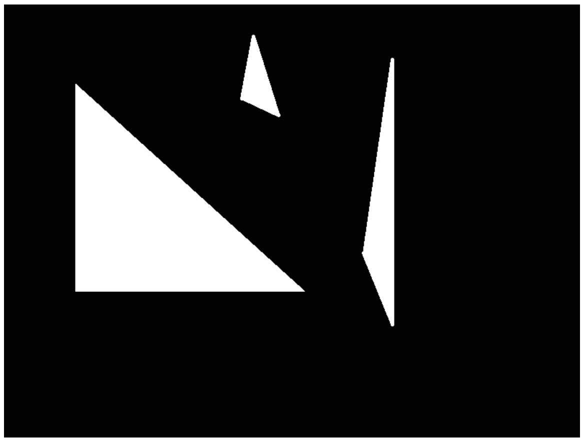


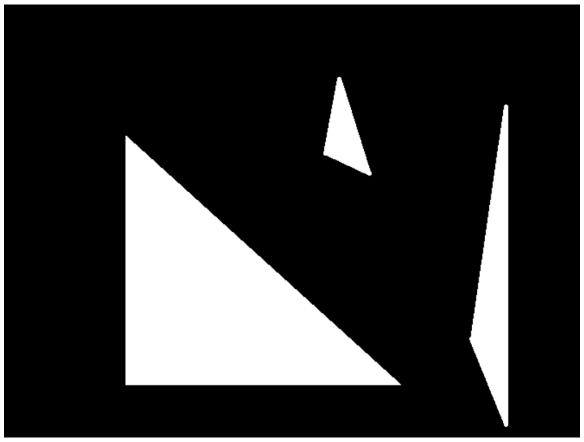
image1.png

pic1.png

pic2.png

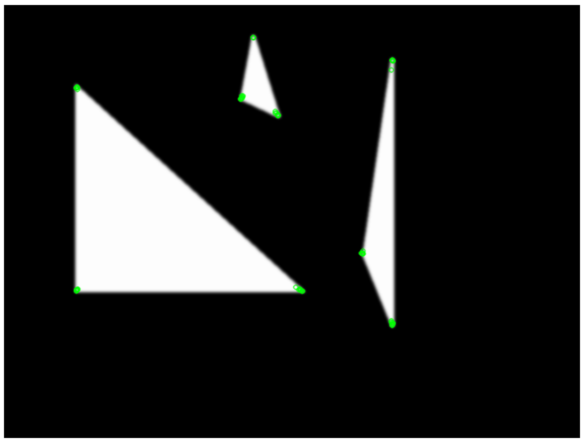
Part 1: Synthetic Images

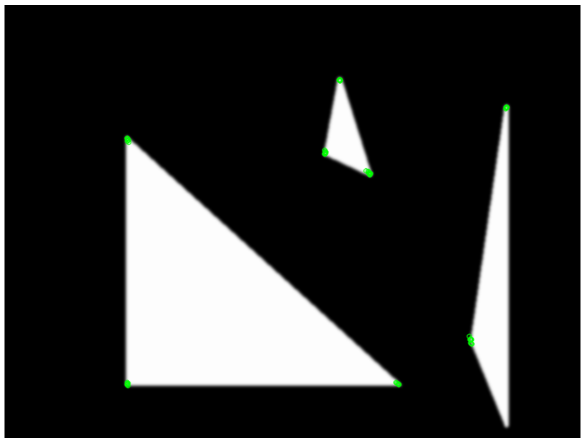
```
M_actual=[[1.2, 0, 50],
          [0, 1.2, 50],
          [0, 0, 1]]
```



pic_1_a.png

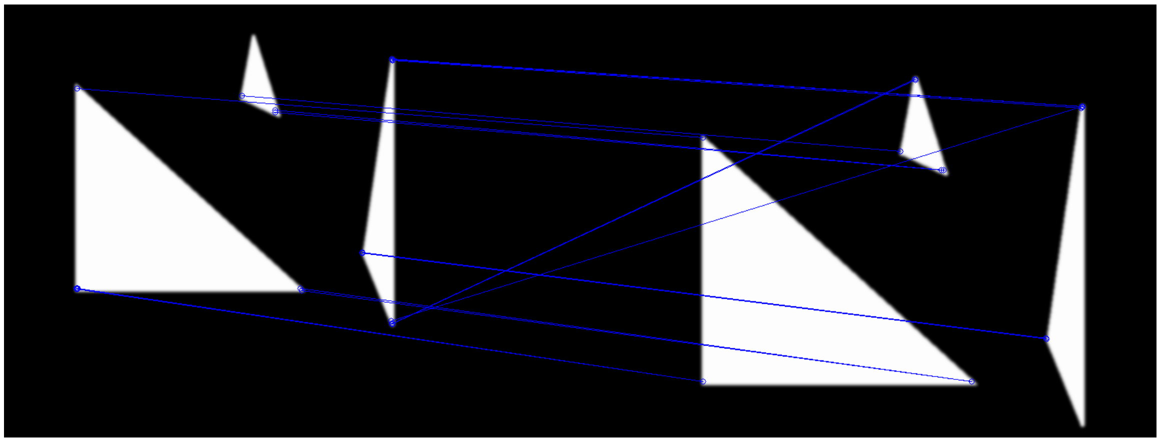
This image was created by applying the above transformation matrix to image1.png using the cv2.warpPerspective function.


pic_1_b_1.png


pic_1_b_2.png

These pictures were created by finding the key points of image1 and image2.

```
orb=cv2.ORB()
kp,des=orb.detectAndCompute(img,None)
img=cv2.drawKeypoints(img,kp,None,color=(0,255,0),flags=0)
```


pic_1_e.png

The above image was created by finding the key points of each image and matching them using the knnMatcher to pair the matches. This is the process of the given code. Then using a drawMatches functions similar to the built in cv2 function, lines between the top matches of the keypoints were drawn.

The draw part of the drawMatches function, retrieved from StackOverflow is shown below.

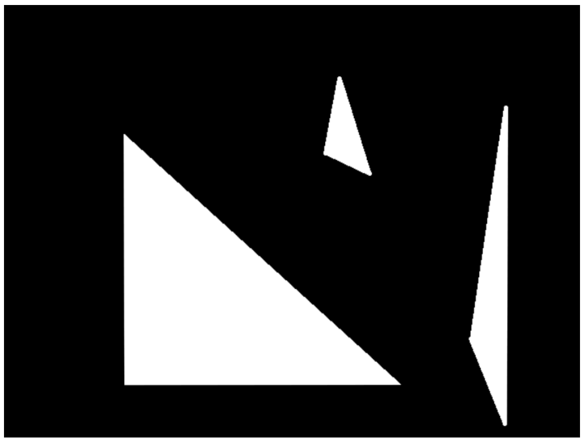
```
for mat in matches:
        # Get the matching keypoints for each of the images
        img1_idx = mat.queryIdx
        img2_idx = mat.trainIdx

        (x1,y1) = kp1[img1_idx].pt
        (x2,y2) = kp2[img2_idx].pt
        cv2.circle(out, (int(x1),int(y1)), 4, (255, 0, 0), 1)
        cv2.circle(out, (int(x2)+cols1,int(y2)), 4, (255, 0, 0), 1)
        cv2.line(out, (int(x1),int(y1)), (int(x2)+cols1,int(y2)), 
(255, 0, 0), 1)
```

In that same function the transformation matrix was calculated form the keypoints of the two images. The order of magnitude of the matricies are the same, and the only more than trivial differences are the 1.22 vs the 1.2 of the scale factor and the 44's instead of 50's in the translate.

```
M_calculated=[[1.22985, 0.00636, 44.4373],
              [0.01359, 1.22252, 44.1310],
              [0.00003, 0.00002, 1.00000]]
```

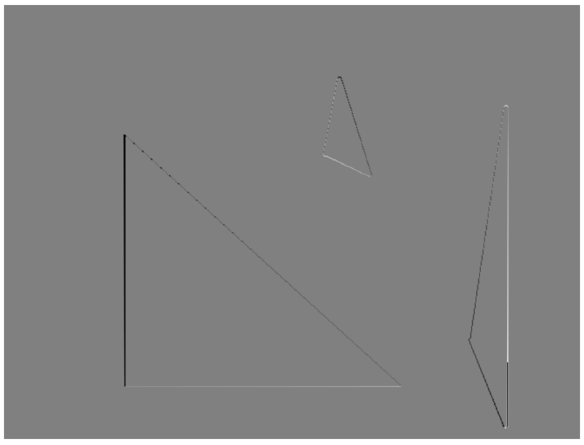
This pictures is image1, or the original synthetic image, transformed using the calculated transformation matrix. Ideally, this would be identical to image2 (pic_1_a). This was done using the warpPerspective function with the calculated transform matrix.

```
h2,w2=pic2.shape[:2]
pic_2_d=cv2.warpPerspective(pic1,M2,
(w2,h2))
```

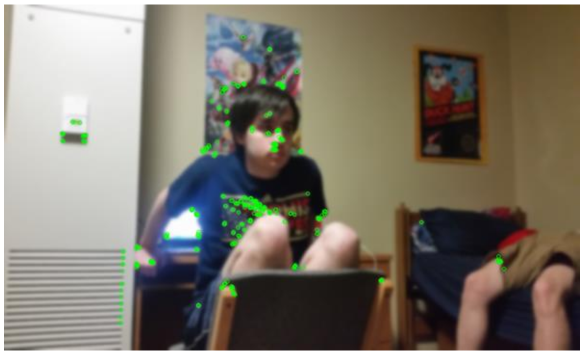

pic_1_e.png

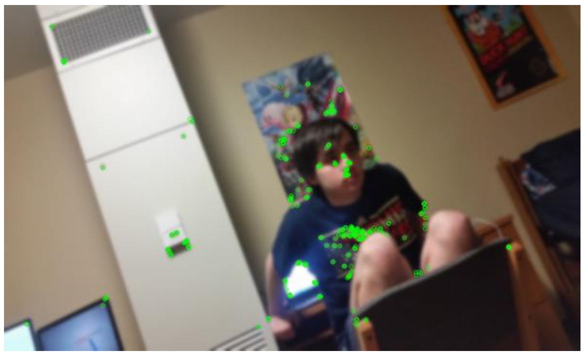
This picture is the difference between pic_1_e and image2 after normalization. If the transformation matrix was perfect, this picture would be completely gray, but the slight inconsistencies between the two pictures created the outline.

```
pic_2_e=np.float32
(funcs.grayscale(pic2))-
funcs.grayscale(pic_2_d)
normalize(pic_2_e)
```

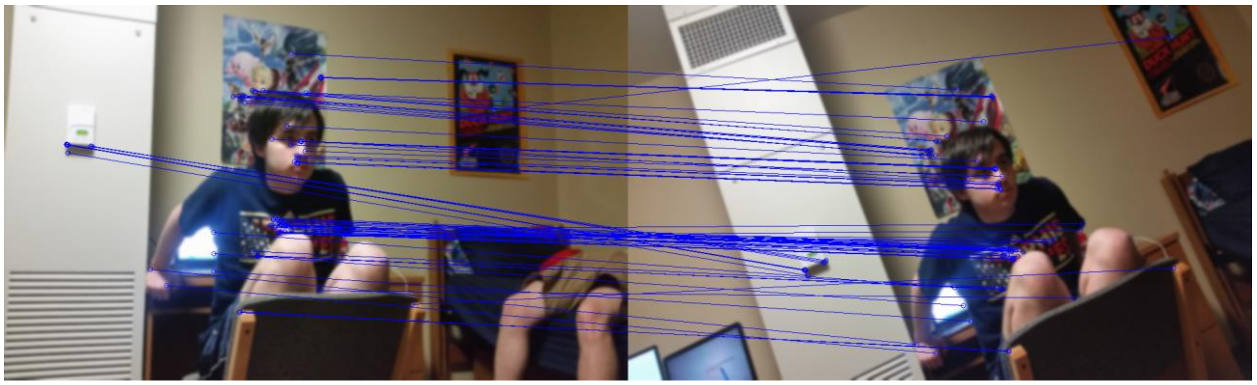

pic_1_f.png

Part 2: Real Images


pic_2_a_1.png


pic_2_a_2.png

This is done using the same process as pic_1_b_1 and pic_1_b_2 after a 15-size kernel Gaussian blur.


pic_2_b.png

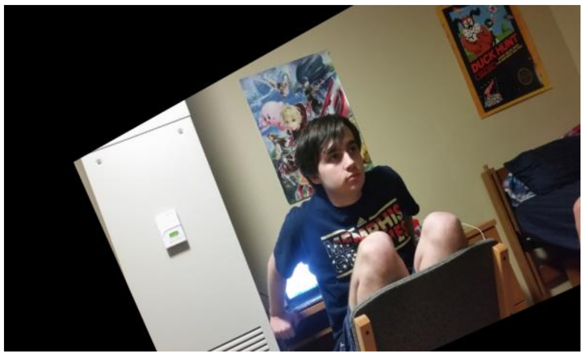
This image is created using the same process and functions as pic_1_c.

The matrix found from these matches is:

```
M=[[0.40479, 0.39018, 120.509],
   [-0.4632, 0.88074, 271.829],
   [-0.0001, 0.00006, 1.00000]]
```

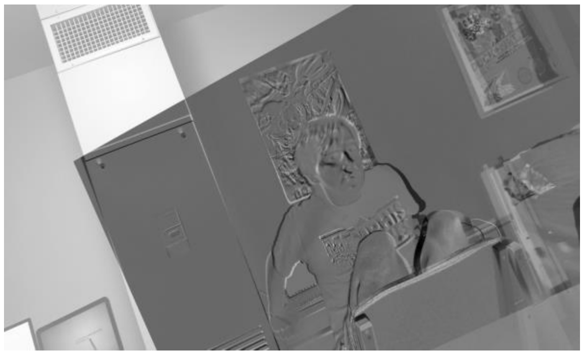
This image is pic1 after it is translated using the above transformation matrix using the warpPerspective function. This should be as close to pic2 as possible.

```
h2,w2=pic2.shape[:2]
pic_2_d=cv2.warpPerspective(pic1,M2,
(w2,h2))
```


pic_2_d.png

This image is the difference between pic_2_d and pic2 after normalization. This should be a gray image in the area where there are two pictures.

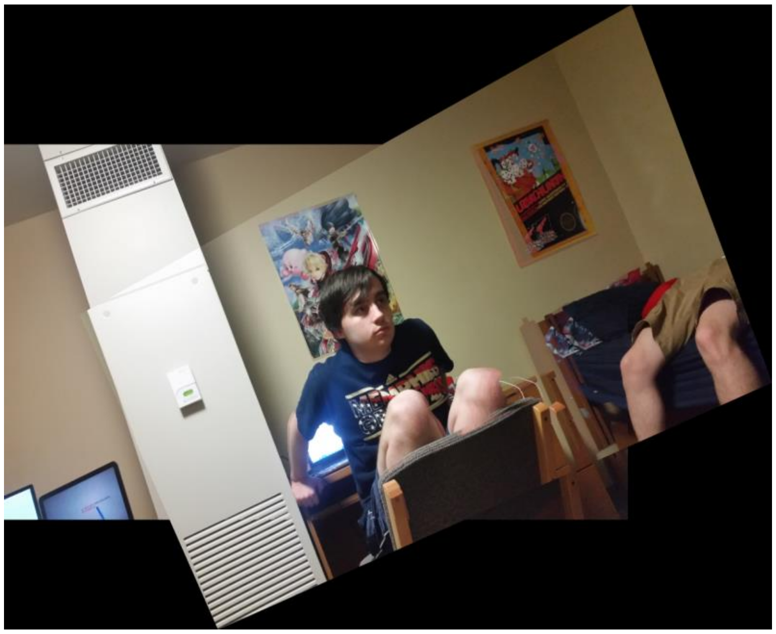
```
pic_2_e=np.float32(funcs.grayscale
(pic2))-funcs.grayscale(pic_2_d)
normalize(pic_2_e)
```



pic_2_e.png

This image is pic2 and the non-clipped version of pic_2_d overlaid on top of each other.

```
def getBoundingRectangle(w,h,M):
      For corner:
            p1 =
      M.dot(np.float32([[x], [y],
      [1]]))
      return p1[0, 0] / p1[2, 0],
             p2[1, 0] / p2[2, 0],
             p3[0, 0] / p3[2, 0],
             p4[1, 0] / p4[2, 0]
```



```
minX,minY,maxX,maxY=getBoundingRectangle(w2, h2, M2)
M3=np.float32([[1, 0, -minX if minX < 0 else 0],
               [0, 1, -minY if minY < 0 else 0],
               [0, 0, 1]])
warp_dim_x = int((abs(minX) if minX < 0 else 0) + (abs(maxX) if maxX >
w2 else w2))
warp_dim_y = int((abs(minY) if minY < 0 else 0) + (abs(maxY) if maxY >
h2 else h2))
pic1_warp = cv2.warpPerspective(pic1, M3.dot(M2), (warp_dim_x,
warp_dim_y))
pic2_warp = cv2.warpPerspective(pic2, M3, (warp_dim_x, warp_dim_y))
pic_2_f = np.maximum(pic1_warp, pic2_warp)
cv2.imwrite("output/pic_2_f.png",pic_2_f)
```