imgs_2_panorama

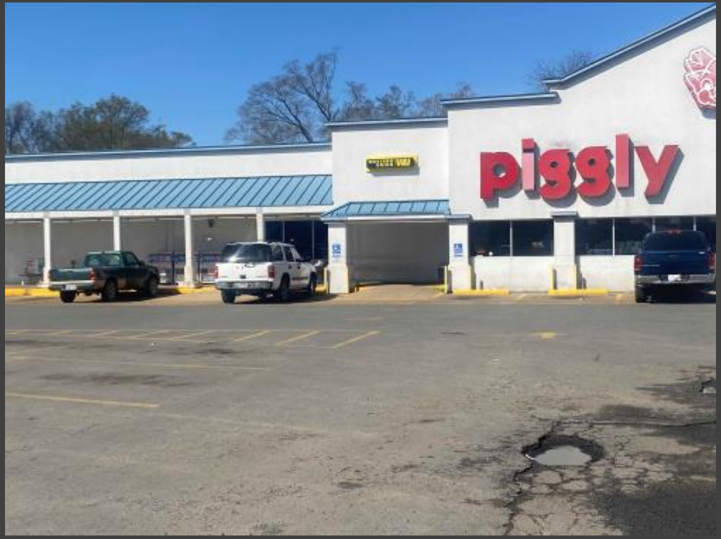
Quade Martin

Introduction

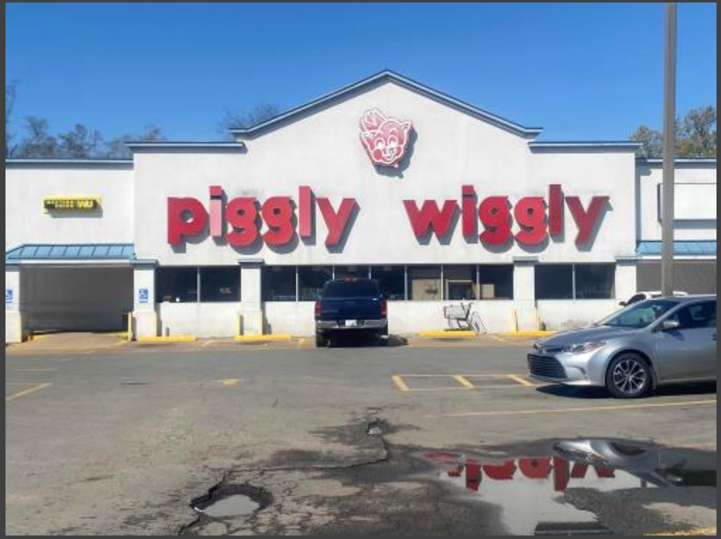
Using NumPy slicing, masks of an image can be created and used to combine overlapping images into a larger panorama image.

Panorama Images

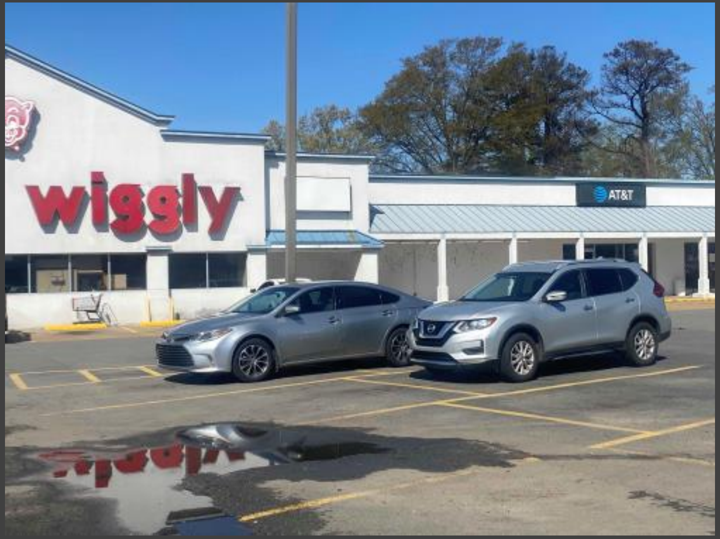
These are the three images used to create the full panorama:



piggly1.png



piggly2.png



piggly3.png

*Images are sized for convenience and are not necessarily to scale

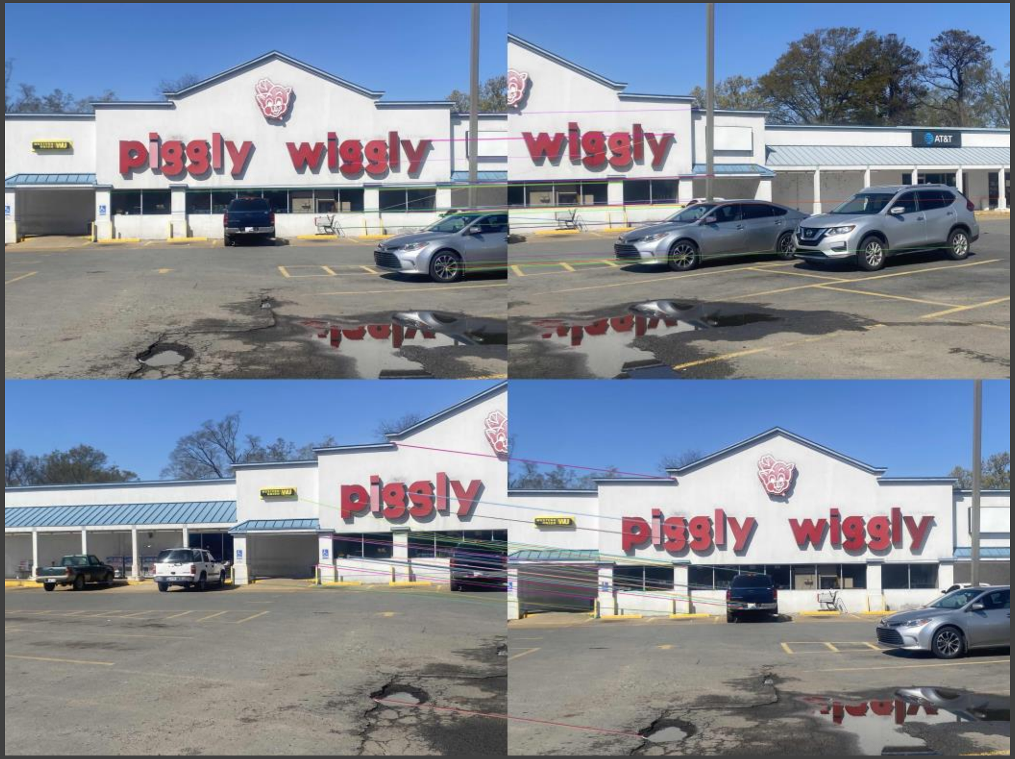
Keypoint Matches

The following code can be used to find and display the keypoint matches, or the matches the program thinks are similar or the same.

```
bf = cv2.BFMatcher(cv2.NORM_HAMMING)
rc_matches = bf.knnMatch(des1,des2,k=2)
lc_matches = bf.knnMatch(des2,des3,k=2)
# ~ print(matches)
rc_good_matches=[]
for best_match,second_best_match in rc_matches:
    if best_match.distance<.75*second_best_match.distance:
        rc_good_matches.append(best_match)
lc_good_matches=[]
for best_match,second_best_match in lc_matches:
    if best_match.distance<.75*second_best_match.distance:
        lc_good_matches.append(best_match)
rc_match=cv2.drawMatches(image1,kp1,image2,kp2,rc_good_matches,None,flags=cv
2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
lc_match=cv2.drawMatches(image2,kp2,image3,kp3,lc_good_matches,None,flags=cv
2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
matched = np.vstack((lc_match, rc_match))
cv2.imwrite("output\matches.png", matched)
```

Keypoint Matches

The following image is what is created by the previous code snippet:



matches.png

*Images are sized for convenience and are not necessarily to scale

Separate Images

The following code creates the translation matrix H1 which translates piggly1.png onto the new canvas.

```
bf = cv2.BFMatcher(cv2.NORM_HAMMING)
rc_matches = bf.knnMatch(des1,des2,k=2)
lc_matches = bf.knnMatch(des2,des3,k=2)
# ~ print(matches)
rc_good_matches=[]
for best_match,second_best_match in rc_matches:
    if best_match.distance<.75*second_best_match.distance:
        rc_good_matches.append(best_match)
lc_good_matches=[]
for best_match,second_best_match in lc_matches:
    if best_match.distance<.75*second_best_match.distance:
        lc_good_matches.append(best_match)
rc_match=cv2.drawMatches(image1,kp1,image2,kp2,rc_good_matches,None,flags=cv
2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
lc_match=cv2.drawMatches(image2,kp2,image3,kp3,lc_good_matches,None,flags=cv
2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
matched = np.vstack((lc_match, rc_match))
cv2.imwrite("output\matches.png", matched)
```

Separate Images

The following code creates the translation matrix H1 which translates piggly1.png onto the new canvas. Piggly2.png is translated onto the new canvas without a translation matrix, as it does not have a perspective warp applied to it.

```
src_points=[]
dest_points=[]
for m in rc_good_matches:
    src_points.append(kp1[m.queryIdx].pt)
    dest_points.append(kp2[m.trainIdx].pt)
src_points=np.float32(src_points)
dest_points=np.float32(dest_points)
H1,_=cv2.findHomography(src_points,dest_points,method=cv2.RANSAC,ransacReprojThreshold=2,maxIters=10000000)
out1=cv2.warpPerspective(image1,T@H1,(new_width,new_height))
out2=cv2.warpPerspective(image2,T,(new_width,new_height))
```

*Images are sized for convenience and are not necessarily to scale

Separate Images

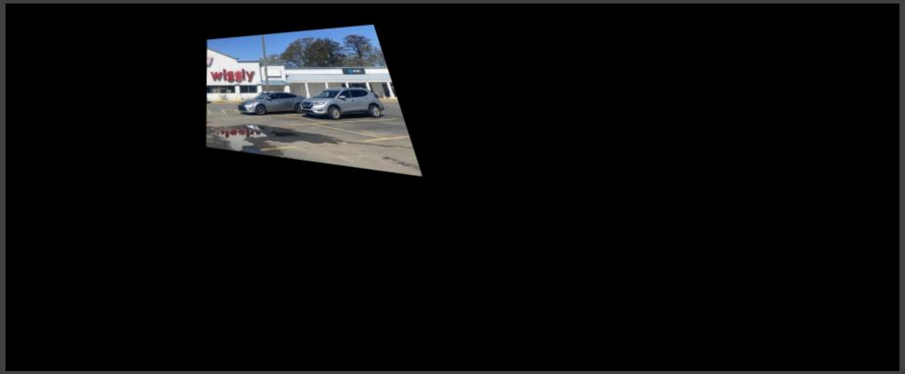
The following code creates the translation matrix H2 which translates piggly3.png onto the new canvas. This is accomplished by taking the keypoints from image4, with image4 being the merged image of image1 and image2.

```
kp4, des4 = orb.detectAndCompute(image4,None)
stitch_matches = bf.knnMatch(des3,des4,k=2)
stitch_good_matches=[]
for best_match,second_best_match in stitch_matches:
    if best_match.distance<.75*second_best_match.distance:
        stitch_good_matches.append(best_match)
src_points2=[]
dest_points2=[]
for m in stitch_good_matches:
    src_points2.append(kp3[m.queryIdx].pt)
    dest_points2.append(kp4[m.trainIdx].pt)
src_points2=np.float32(src_points2)
dest_points2=np.float32(dest_points2)
H2,_=cv2.findHomography(src_points2,dest_points2,method=cv2.RANSAC,ransacReprojThreshold=2,maxIters=10000000)
out3=cv2.warpPerspective(image3, T@H2, (new_width,new_height))
```

Separate Images

These are the three images produced by the previous code snippets:

out1.png


out3.png

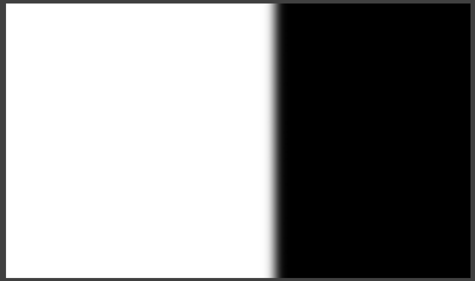
out2.png

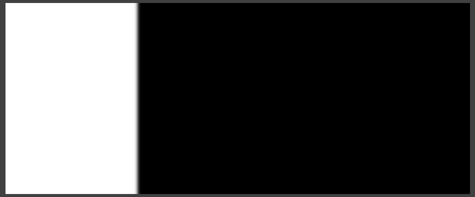
*Images are sized for convenience and are not necessarily to scale

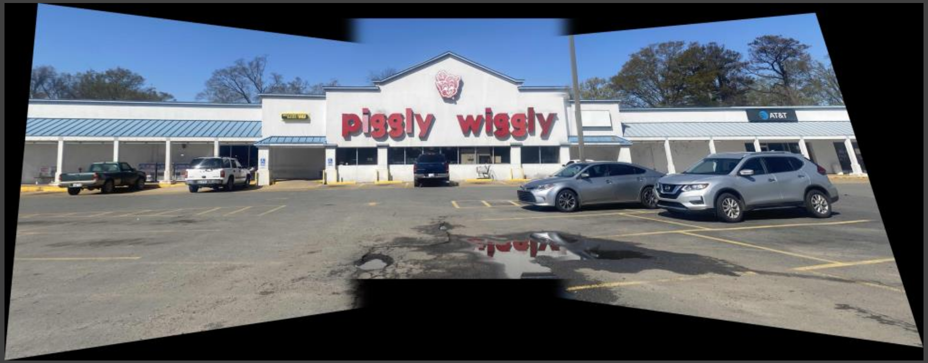
Final Panorama without Cropping

The following code snippet creates 2 masks which are used to merge image1 and image2 into a new image called image4, and then merge image4 and image3 to create the final panorama.

```
diff2=np.uint8(abs(out4*1.0-out3))
mask2=diff2*0
mask2[:,:cut_x-int(minx)]=255
mask2=cv2.GaussianBlur(mask2,(255,1),sigmaX=-1)
cv2.imwrite("output\mask2.png", mask2[::2,::2])
final=diff2*0
percent=mask2/255.0
final=np.uint8(out4*percent+out3*(1-percent))
cv2.imwrite("output\panoFinal.png", final[::2,::2])
```


mask1.png


mask2.png



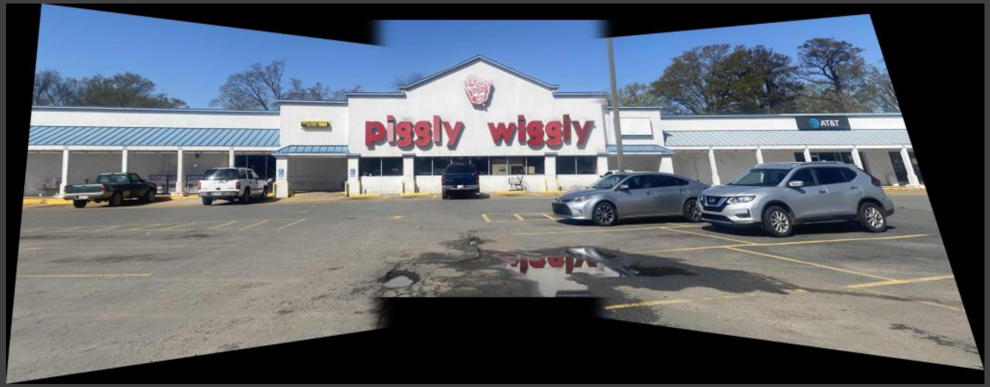
panoFinal.png

*Images are sized for convenience and are not necessarily to scale

Final Panorama with Cropping

The following code crops panoFinal.png around the parts that have color, so that the image is more visible.

```
panorama_image=final
gray = cv2.cvtColor(panorama_image, cv2.COLOR_BGR2GRAY)
_, thresh = cv2.threshold(gray, 1, 255, cv2.THRESH_BINARY)
contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
largest_contour = max(contours, key=cv2.contourArea)
x, y, w, h = cv2.boundingRect(largest_contour)
cropped_image = panorama_image[y:y+h, x:x+w]
cv2.imwrite("output\cropped_panoFinal.png", cropped_image)
```



cropped_panoFinal.png

*Images are sized for convenience and are not necessarily to scale

Panorama

Using NumPy slicing and masks, images can be merged together into panoramas. The cut_x variable used in this program has a large effect on the final panorama, as it dictates how the image will be cut and merged. If the cut_x variable is too large, the merging images can be cut incorrectly and pasted together to not complete a full panorama.