Computer Vision Seamless Panoramic Image

Ian Sinclair
ENCE 3620-1

March 30, 2022
Dr. Mohammad H. Mahoor

Abstract

Creating a panoramic image from related smaller images can help resolve field of view limitations from image capture devices. Preserving critical information while expanding the image plan by a combination of sub-images. Within this report, two algorithms for panoramic image stitching or mosaicing are considered. The first, uses openCV SIFT key point detector and description generator to match features on sub-images, then uses RANSAC and ratio testing to prune trivial features. Finally, uses a projective transformation to warp the sub-images to a final larger panoramic image. The second method uses Harris corner detector to general key points, then SIFT to find descriptions, and then proceeds the same as method 1. As a result, the scope of the report handles an implementation of simple image mosaicing algorithms and briefly compares there result from a visual standpoint.

1 Background

Image stitching is a technique in computer vision to warp many smaller images taken from different optical perspective together to make a single (ideally seamless) panoramic image. This involves detecting 'rare' attributes on each sub-image as features. Then creating a map between similar attributes and warping the images to make a seamless overlay.

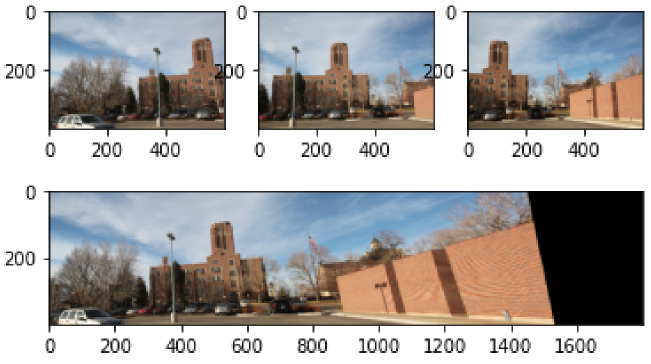

Figure 1: Image Stitching of 3 sub-images into a panoramic larger image.

1.1 Key Point Detection

Attributes of interest on each image are elements that are uncommon in the range of images. Or features that may be able to uniquely identify each image. Examples include corners or sharp edges (things that are typically not naturally occurring). The points where these attributes are located are called 'key points', and two detection methods are considered.

1.1.1 Harris corner detection

A corner in an image is a region with high changes of intensity in all directions, making them unusually in terms of elements consistent across all images. Or are a good candidate for detecting unique features on each sub-image. Harris corner detection finds the change of intensity in all directions at a given pixel location. Then, compute the eigen values of this gradient change to determine if the point is near a corner, edge, or smooth surface. In general, this involves the kernel,

$$E(u,v) = \sum_{x,y\in I} \underbrace{\omega(x,y)}_{\text{Window Function}} \; [\underbrace{I(x+u,y+v)}_{\text{Shifted Intensity}} - \underbrace{I(x,y)}_{\text{Intensity}}]^2$$

For x, y in the image plane, and $I(x, y)$ being the intensity value of the image at coordinate (x, y). Then the windowing function is a Gaussian kernel to weight the underlaying pixels.
Note, maximizing this function corresponds to large changes in all directions which (is sufficiently large) may indicate a corner element on the image. And so consider the equivalent expression

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

Where,

$$M = \sum_{x,y\in I} \omega(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

Where, I_q is the derivative in the direction q.
Typically, scoring is done by,

$$r = \det(M) - k \cdot (trace(M))^2$$

For some controllable parameter k.
Then, if r is sufficiently large, there is a corner at pixel location (x, y).

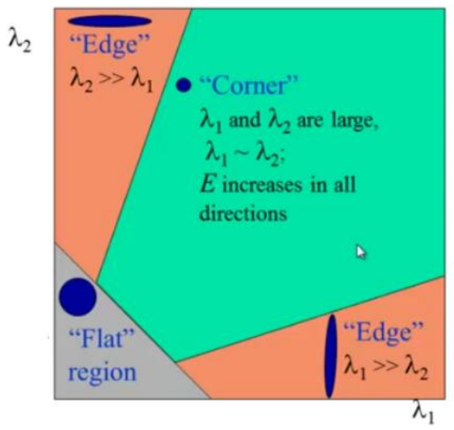

Figure 2: Concept are describing the relationship of r scoring and image elements

And so, corner features are can be extracted and used as key points when designed the mosaic.

1.1.2 SIFT Compute

An unfortunate limitation of Harris corner detection is it is not scale inveriant, meaning if the size of share attributes in different sub-images changes then the map may misidentify each key point or their relationship. And so, SIFT (Scale-Invariant Feature Transform) is an algorithm that overcomes this limitation by adding sensitivity to scale. Correspondingly, SIFT uses DoG operator's to detect corners of different sizes within an image. And is iterated over many octaves of a Gaussian pyramid.

1.2 Feature Matching

Now that key points have been generated, it is possible to create a map between them that can eventually be used to warp the images to create a seamless overlay.

1.3 Best Matching/ Thresholding

1.4 RANSAC Homography

1.5 Projective Transformation and Image Warping

2 Procedure

2.1 Raw Images

Consider the raw image files,

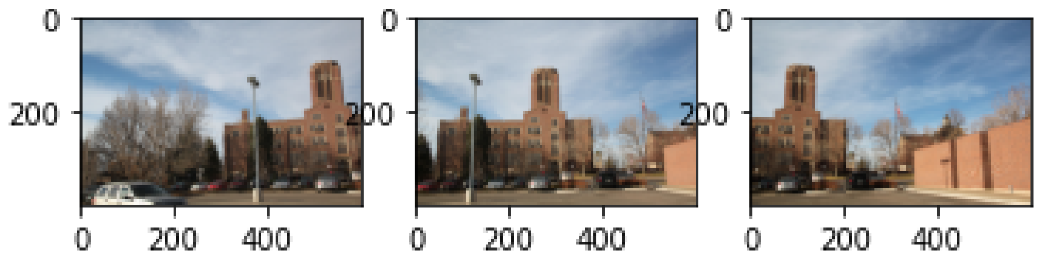

Figure 3: Raw images of test set 1.

2.2 Selecting Key Points

2.2.1 SIFT

Consider the SIFT algorithm

```
def SIFT_detection( imagee ) :
  im = imagee.copy()
  sift = cv.SIFT_create()
  gray = cv.cvtColor(im,cv.COLOR_BGR2GRAY)

  kp,desc = sift.detectAndCompute(gray,None)

  SIFT_Image = cv.drawKeypoints(gray,kp,im)

  return kp, desc, SIFT_Image
```

Then consider the Harris corner detection for method 2.

```
def custom_Harris_Keypoints( imageeee, kernal_size, k , threshold) :
  im = imageeee.copy()
  sifty = cv.SIFT_create()
  gray = cv.cvtColor(im,cv.COLOR_BGR2GRAY)
  gray = np.float32(gray)

  sobel_x = np.array([[-1, 0, 1],[-2, 0, 2],[-1, 0, 1]])
  sobel_y = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])
  Blurx = cv.filter2D( gray,-1,sobel_x )
  Blury = cv.filter2D( gray,-1,sobel_y )
  Blurx = cv.filter2D( Blurx,-1,sobel_x )
  Blury = cv.filter2D( Blury,-1,sobel_y )
```

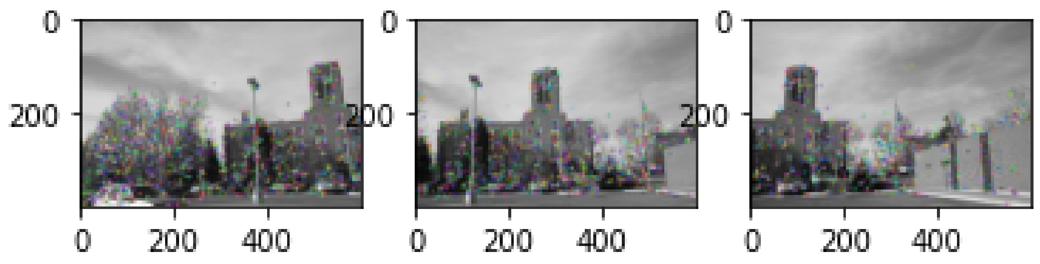
Figure 4: SIFT Algorithm key points for test set 1.

```

    Blurxx = Blurx**2
    Blurxy = Blurx*Blury
    Bluryy = Blury**2

    keyPoints = []
    kernal_w = kernal_size
    half_w = int( kernal_w/2 )
    for i in range( half_w, gray.shape[0]-half_w ) :
      for j in range( half_w, gray.shape[1]-half_w ) :
        Sum_xx = np.sum( Blurxx[ i-half_w:i+1+half_w, j-half_w:j+1+half_w ] )
        Sum_yy = np.sum( Bluryy[i-half_w:i+1+half_w, j-half_w:j+1+half_w ] )
        Sum_xy = np.sum( Blurxy[i-half_w:i+1+half_w, j-half_w:j+1+half_w ] )

        det = (Sum_xx * Sum_yy) - (Sum_xy ** 2)
        trace = Sum_xx + Sum_yy
        r = det - k * (trace ** 2)

        keyPoints.append([j, i, r])


    out_image = imageeee.copy()
    valid_keyPoints = []
    #thr = threshold
    for kp in keyPoints :
      if kp[2] > threshold :
        #valid_keyPoints += [[kp[0], kp[1]]]
        valid_keyPoints += [ cv.KeyPoint(kp[1],kp[0], 1) ]
        out_image[kp[1], kp[0]] = [255,0,0]
    return valid_keyPoints, out_image


def SIFT_description_detection( image_A, key_points ) :
    img = image_A.copy()
    sift = cv.SIFT_create()
    gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
    k, desc = sift.compute(gray, key_points)

    return k, desc
```

Which first applies a Gaussian filter, Ultimately resulting in,

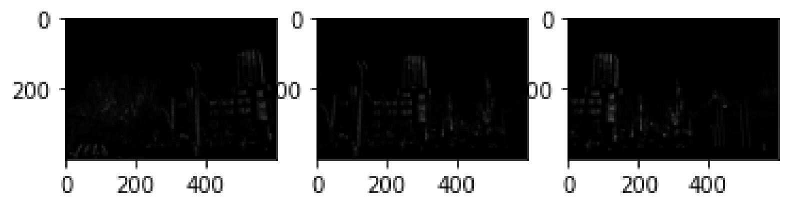
Figure 5: Vertical Sobel Filter on test set 1.

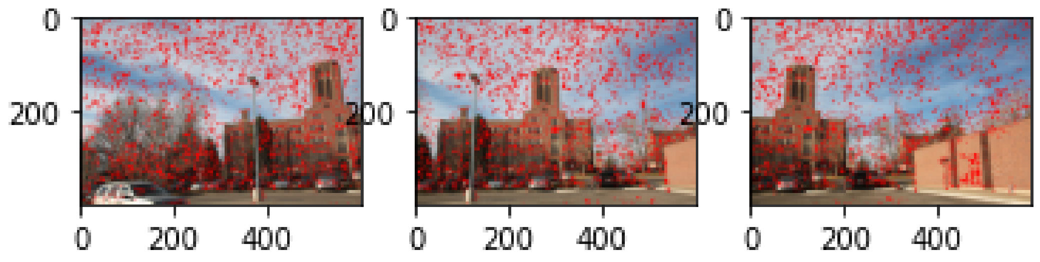
Figure 6: Harris Corner Detection key points.

2.3 Feature Matching

Now the best features are obtained by,

```
def keyPoint_matching( SIFT_desc_src, SIFT_desc_tar ) :
  SIFT_desc_src_t = SIFT_desc_src.copy()
  SIFT_desc_tar_t = SIFT_desc_tar.copy()
  bf = cv.BFMatcher()
  matched_keyPoints = bf.knnMatch(SIFT_desc_src_t,SIFT_desc_tar_t, k=2)

  return matched_keyPoints


def Prune_matches( match_list ) :
  keyPoints = match_list.copy()
  valid = []
  for kp in keyPoints :
    if kp[0].distance < 0.5*kp[1].distance:
      valid.append( kp )
  keyPoints = np.asarray( valid )

  return keyPoints
```

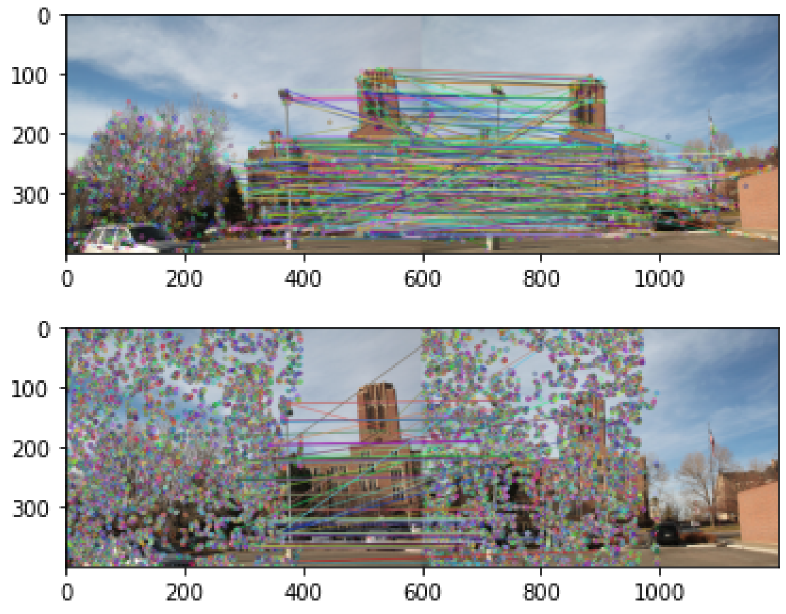
Figure 7: Feature matching on test set 1 for method 1 (above) and method 2 (below).

2.4 RANSAC Homography

```
def RANSAC_homography( match_list_keyPoints, SIFT_kp_src, SIFT_kp_tar ) :
  mm = match_list_keyPoints.copy()
```

```
    SIFT_kp_src_t = SIFT_kp_src.copy()
    SIFT_kp_tar_t = SIFT_kp_tar.copy()
    if len(mm[:,0]) >= 4:
        src = np.float32([ SIFT_kp_src_t[m.queryIdx].pt for m in mm[:,0] ]).reshape(-1,1,2)
        dst = np.float32([ SIFT_kp_tar_t[m.trainIdx].pt for m in mm[:,0] ]).reshape(-1,1,2)
        Transform, masked = cv.findHomography(dst, src, cv.RANSAC, 5.0)
    #print H
        return Transform, masked
    else:
        raise AssertionError('Insufficient Data')
```

2.5 Image Warping

```
def Warp_stitching( image_src, image_tar, H ) :
    '''
      Warps the target image to the source image.
    '''
    image_src_t = image_src.copy()
    image_tar_t = image_tar.copy()
    H_t = H.copy()
    dst = cv.warpPerspective(image_tar_t,H_t,(image_tar_t.shape[1] + image_src_t.shape[1], image_tar_t.
        shape[0]))

    #plt.subplot(122),plt.imshow(dst),plt.title('Warped Image')
    #plt.show()
    #plt.figure()

    for j in range(0, len(image_src[0])) :
        for i in range(0, len(image_src)) :
            if mean(image_src[i][j]) != 0 :
                dst[i][j] = image_src[i][j]

    return dst
```

2.5.1 SIFT Method 1:

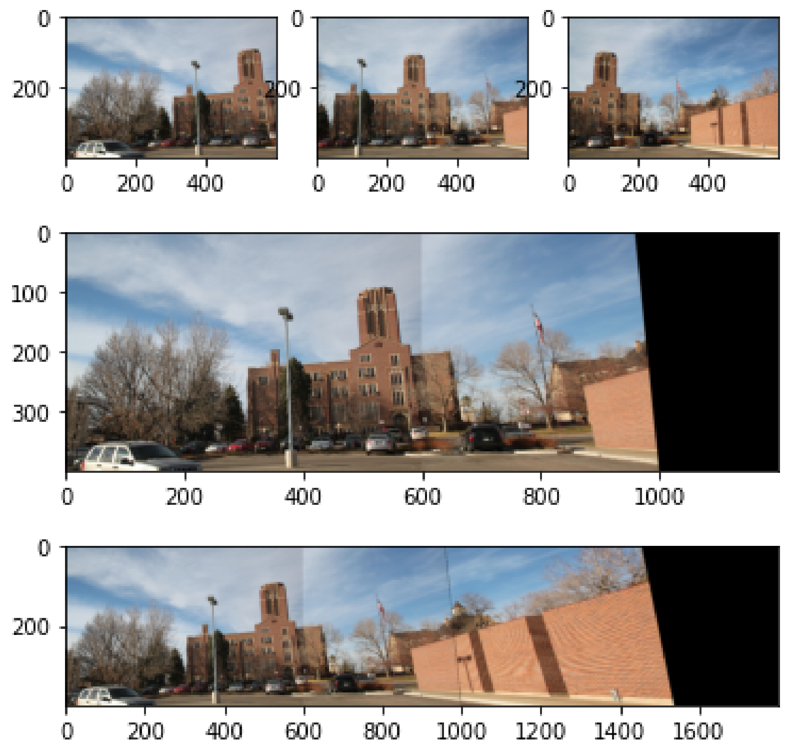

Figure 8: Method 1 Image without cropping or alpha blending

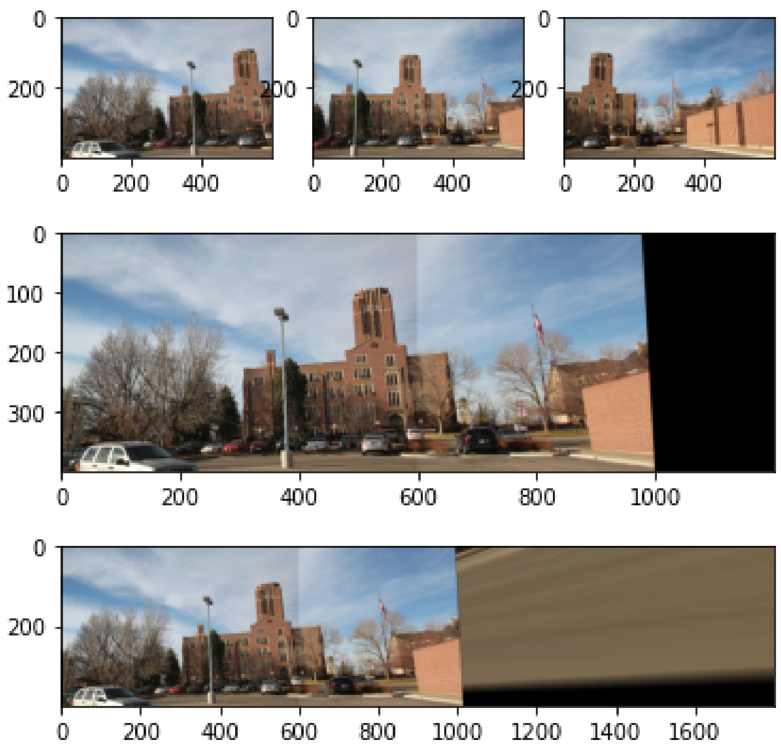

Figure 9: Method 2 test set 1 without cropping or alpha blending

2.5.2 Harris Corner Method 2:

2.6 Automatic Cropping

```python
def entropy( signal ) :
  lensig=signal.size
  symset=list(set(signal))
  numsym=len(symset)
  propab=[np.size(signal[signal==i])/(1.0*lensig) for i in symset]
  ent=np.sum([p*np.log2(1.0/p) for p in propab])
  return ent

def calc_entropy( image ) :
  N=5
  S=image.shape
  E=np.array(image)
  for row in range(S[0]):
    for col in range(S[1]):
      Lx=np.max([0,col-N])
      Ux=np.min([S[1],col+N])
      Ly=np.max([0,row-N])
      Uy=np.min([S[0],row+N])
      region=image[Ly:Uy,Lx:Ux].flatten()
      E[row,col]=entropy(region)
  return E

def calc_seam( Energy_Image ) :
  Eimg = Energy_Image.copy()

  seams = []
  for start in range(0 , len(Energy_Image[0])) :
    j = start
    i = 0
    min_path = [ [i,j] ]
    while i != len(Energy_Image)-1 :
      adj_list = []
      if j+1 < len(Energy_Image[0]) and i+1 <= len(Energy_Image) :
        adj_list += [[i+1, j+1]]
      if j-1 < len(Energy_Image[0]) and i+1 <= len(Energy_Image) :
        adj_list += [[i+1, j-1]]
        adj_list += [[i+1 , j]]

      min_Energy =  float('inf')
      min_index = []
      for q in adj_list :
        if Eimg[q[0],q[1]] <= min_Energy :
          min_Energy = Eimg[q[0],q[1]]
          min_index = q
      min_path += [ min_index ]
      i = min_index[0]
      j = min_index[1]
    seams += [ min_path ]
  return seams


def Seam_weight( H_image, seam ) :
  img = H_image.copy()
  sum = 0
  for q in seam :
    sum += img[q[0],q[1]]
  return sum

def auto_crop( images ) :
  from PIL import Image as PI
  img = images.copy()
  out_image = np.array(images.copy())
  gray = PI.fromarray(img)
  gray = gray.convert('L')
  gray = np.array(gray)

  entr = calc_entropy( gray )

  ss = calc_seam( entr )

  for i in range(len(ss)-1 , 0, -1 ) :
    if Seam_weight(entr, ss[i]) == 0 :
      #Delete column.
      out_image = np.delete(out_image, i, axis = 1)

  return out_image
```

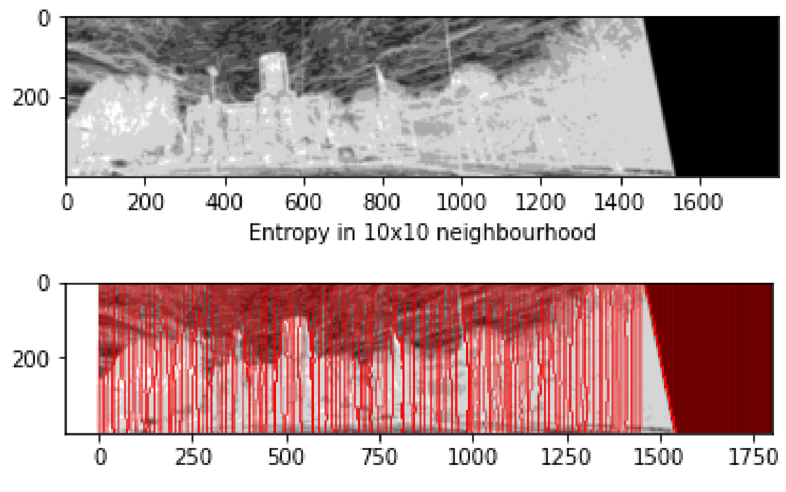

Figure 10: Energy map and seam map of test set 1.

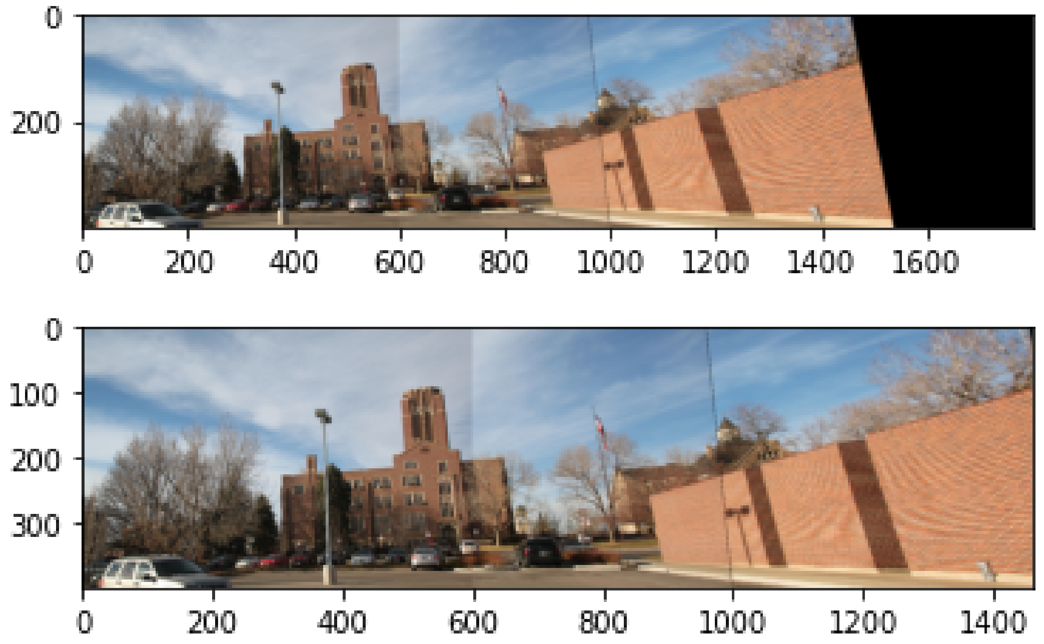

Figure 11: Automatically cropped final image.

2.7 Alpha blending

Three blending techniques are considered,

i) Overlay (no blending). This is the technique shown above, where the non warped image is pasted over the warped image. Notice seams between the images.

ii) Averaging between images. This technique finds the overlap between the warped and nonwarped image after the non-warped image is overlaid, and averages the intensity values at each pixel location.

$$0.5 \cdot I_1 + 0.5 \cdot I_2$$

iii) Linear Parameterized Gradient method, assigns a alpha gradient in the direction of the warped image from the non warped image, and linearly adjusts the average between the images.

$$\alpha \cdot I_1 + (1 - \alpha) \cdot I_2$$

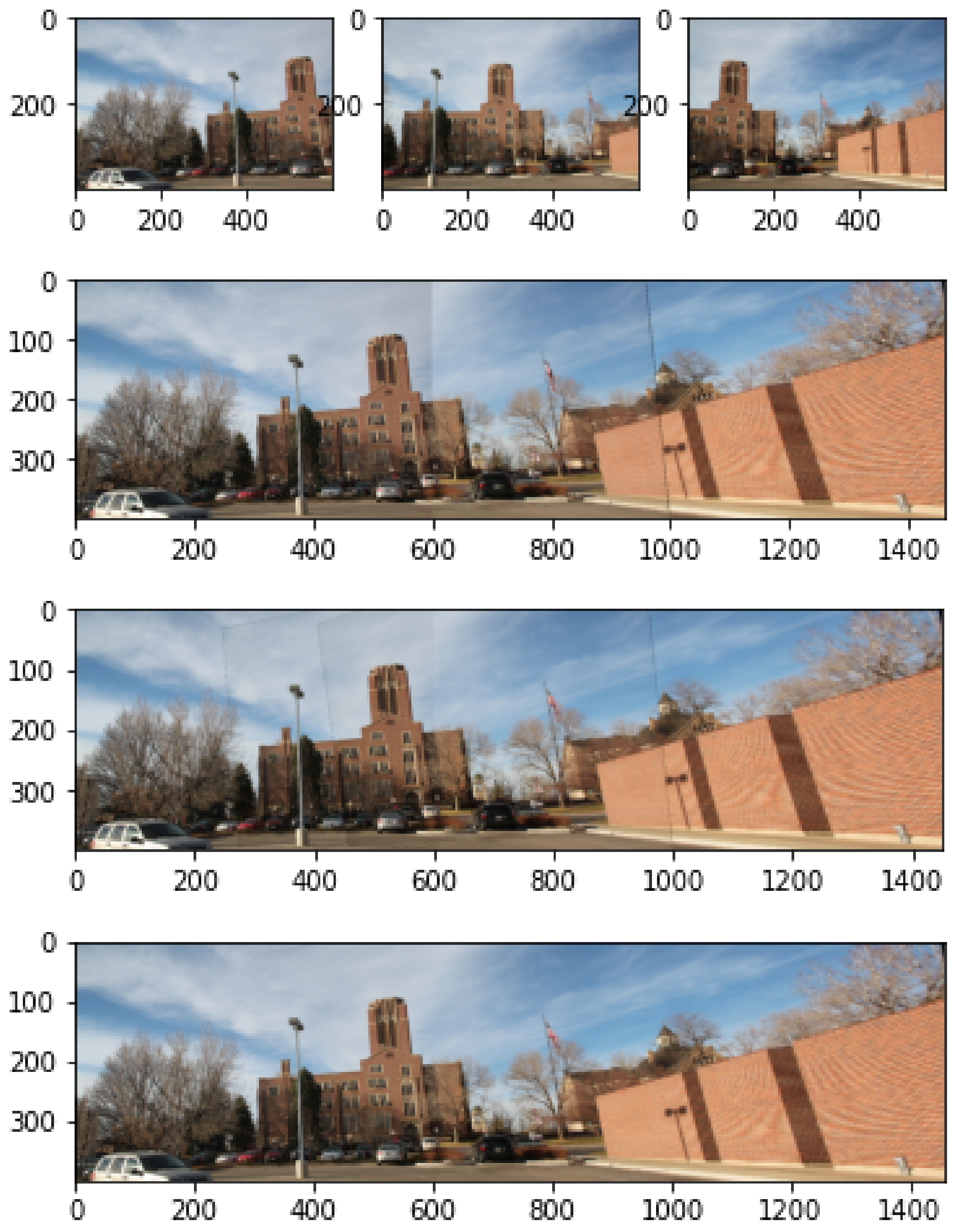

Figure 12: Alpha Blending for SIFT model

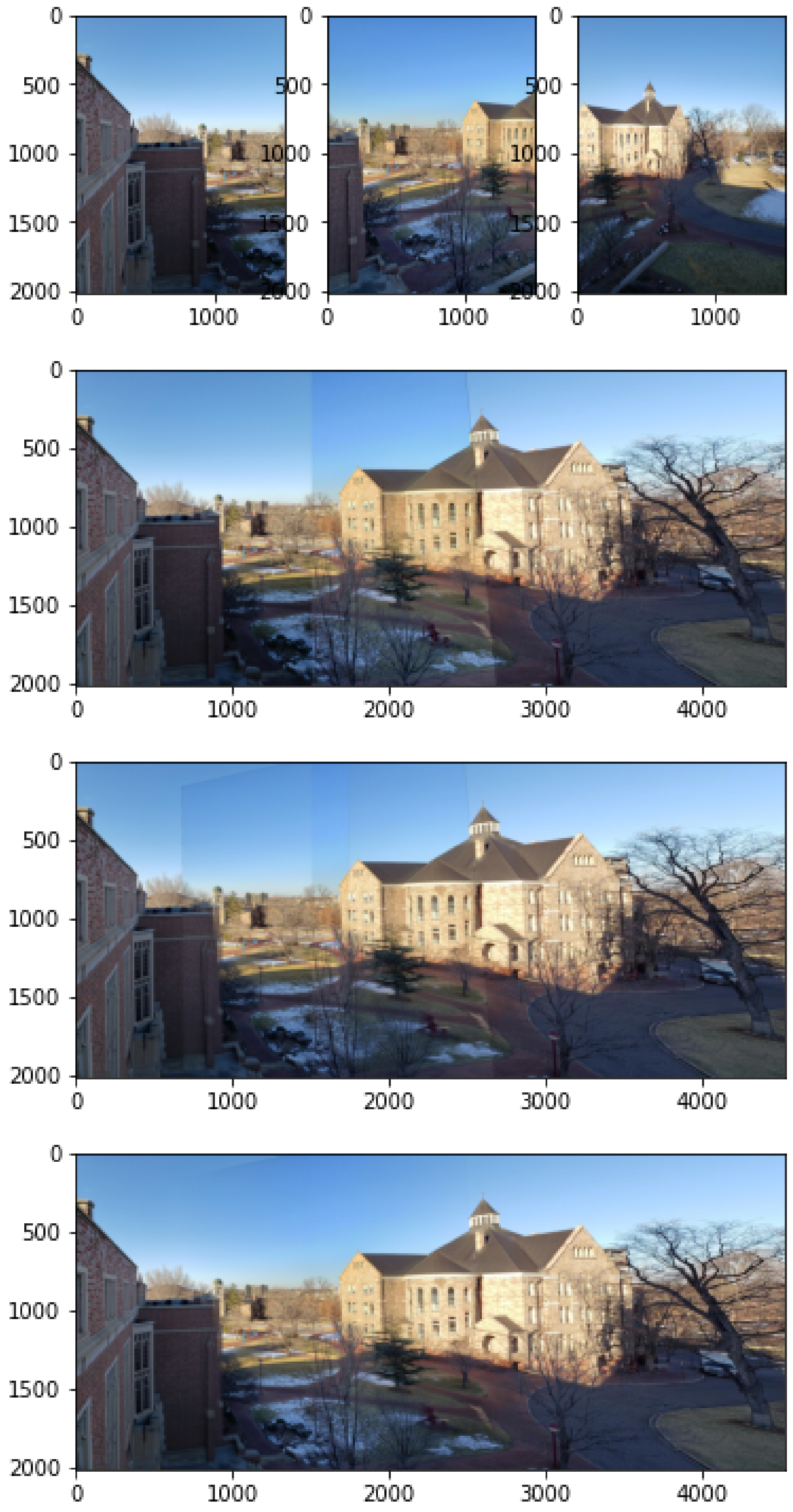

Figure 13: Alpha Blending for SIFT model

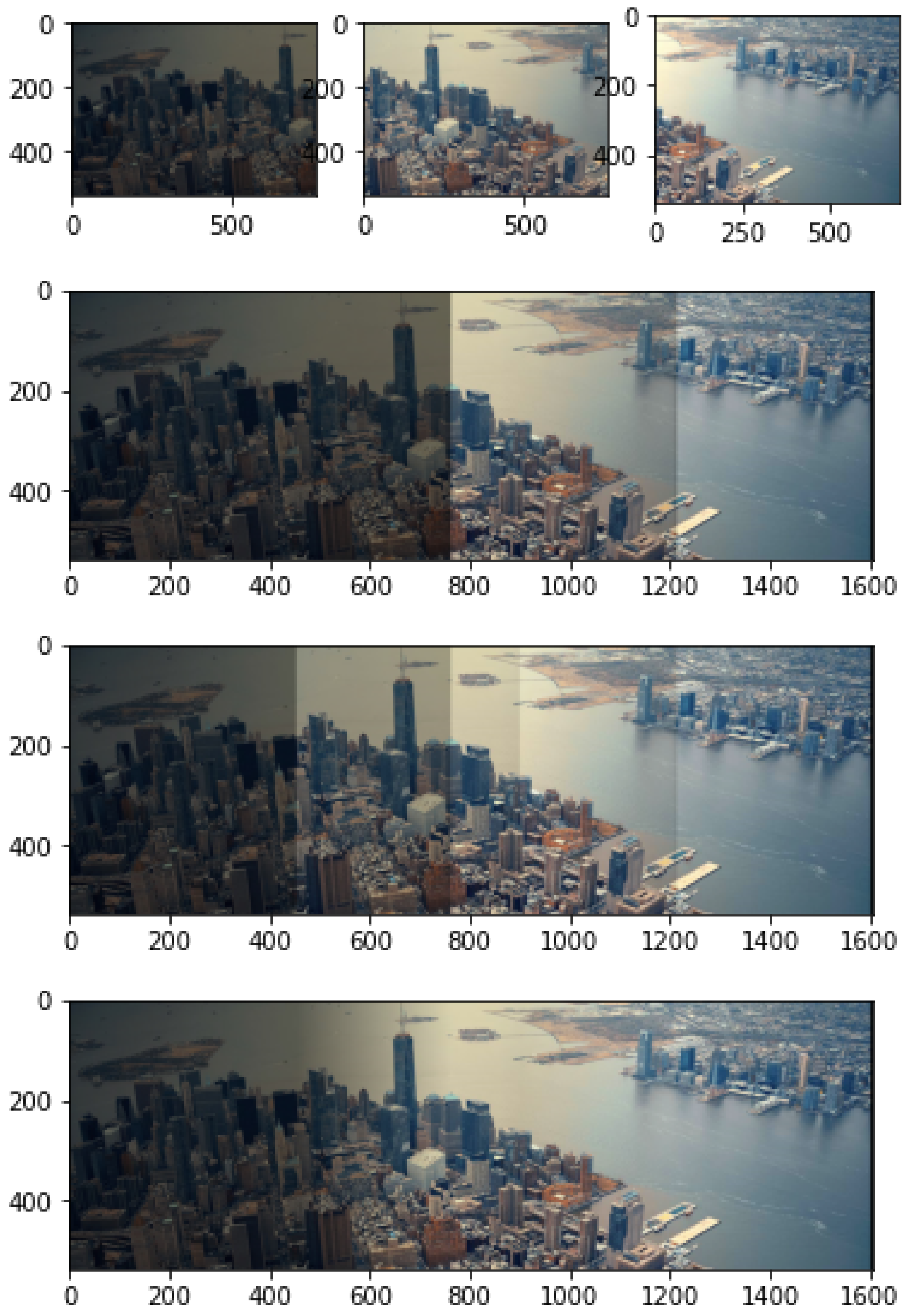

Figure 14: Alpha Blending for SIFT model