Alineación y costura de imágenes.

J.F. Sánchez Castillo, ITNM.

I. Introducción.

En este documento se explica el procedimiento que se siguió para realizar la costura de un grupo de 10 imágenes tomadas consecutivamente.

El procedimiento básico cubre los puntos de Detección, emparejamiento de rasgos, el cálculo de la matriz de homografía, la transformación geométrica y la mezcla de imágenes. Estos procedimientos deben ser aplicados a cada par de mosaicos, que a su vez generara un mosaico compuesto, para posteriormente unir este mosaico compuesto a un mosaico de referencia. La secuencia como se procesa cada par de mosaicos hasta generar el panorama final se muestra a continuación:

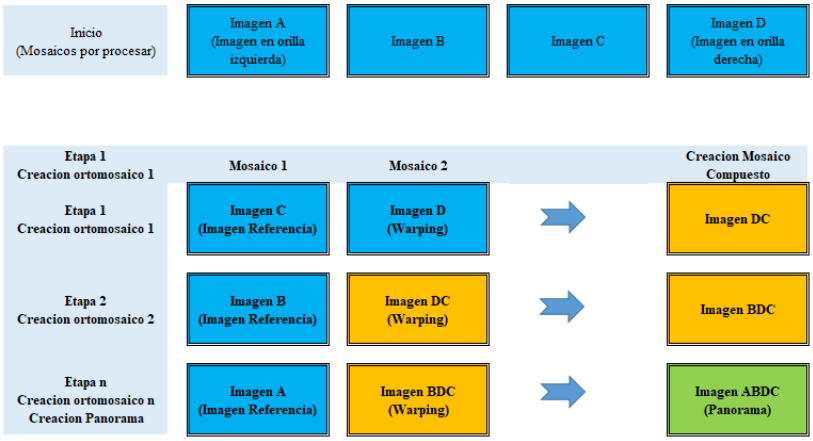

Figura 1. Secuencia para procesamiento de imágenes.

El proyecto se desarrolló con el lenguaje python3 y las librerías de OpenCV3. El software consiste en dos módulos: Modulo “Costura.py” se encarga de lectura de las imágenes de prueba, así como llamar a las funciones necesarias para la operación de costura, también muestra la imagen resultante del panorama creado. El modulo “Operaciones.py” contiene todas las funciones necesarias que son llamadas para realizar la costura como son cálculo de homografía, detección de rasgos, transformación geométrica entre otras. Todos estos

El programa le permite elegir al inicio si desea hacer el proceso de mezclado para determinada imagen, o desea mezclar todas las imágenes secuencialmente.

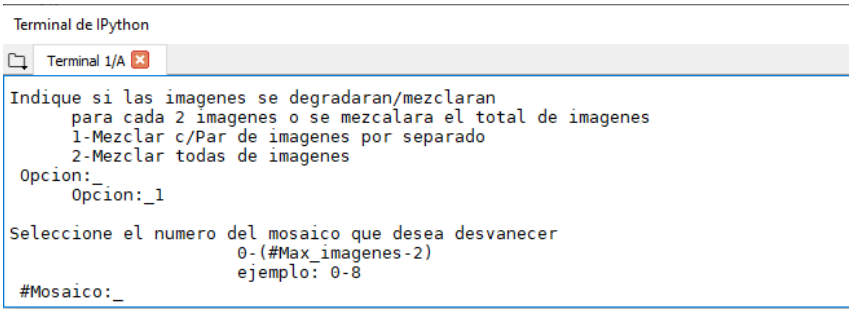

Figura 2. Menú inicio.

Posteriormente se cargan las imágenes que se encuentren en la carpeta ./images/, se recomienda renombrar los archivos de forma ascendente con letras, siendo la imagen ubicada a en la orilla izquierda el archivo "a.jpg" y la imagen ubicada en la orilla derecha el ultimo archivo en la secuencia por ejemplo si se trata de un grupo de diez imágenes el archivo se llamara "j.jpg". En la etapa final los panoramas generados se encontraran en los archios: Panorama.jpg y Panorma_blurred.jpg.

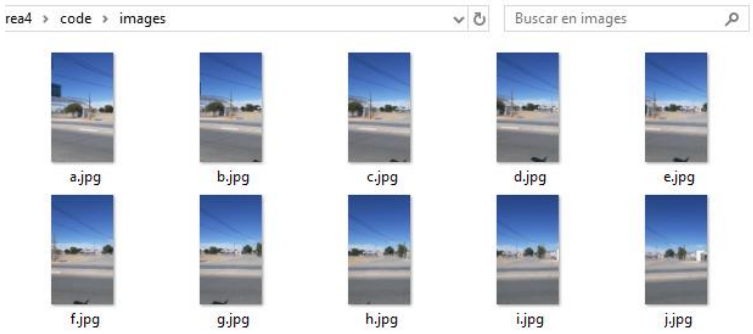

Figura 3. Imágenes a procesar ubicadas en carpeta ./images/

II. OPERACIONES

A. Rasgos detectados.

El proceso de detección de rasgos se utiliza el detector y descriptor de SIFT, luego los datos de los rasgos se almacenan en las variables "keypoints" y "Features". Las funciones utilizadas son:

```
descriptors=cv2.xfeatures2d.SIFT_create()
(Keypoints,features) = descriptors.detectAndCompute(image,None)
```

B. Emparejamiento de rasgos

Para el emparejamiento se toman los rasgos detectados y mediante el emparejador “Brute Force” se detectan los 2 mejores emparejamientos para cada par de rasgos, enseguida se define un radio de búsqueda y mediante la distancia de cada emparejamiento, se evalúa si está dentro de esta distancia (lowe radio), si cumple con esta codician el emparejamiento se almacena en el arreglo valid_matches y se pasa a la siguiente etapa. Las funciones utilizadas son:

Crea emparejador “Brut Force” y encuentra los dos mejores emparejamientos:

```
match_instance=cv2.DescriptorMatcher_create(“BurteForce”)
all_Matches=match_instance.knnMatch(featuresA,featuresB,2)
```

Si la distancia entre estos emparejamientos se encuentra entre un radio de búsqueda el emparejamiento se almacena.

```
For val in All Matches:
        If len(val)==2 and val[0].distance < val[1].distance*lower_ratio:
                valid_matches.append((val[0].trainIdx,val[0].queryIdx))
```

C. Refinamiento de rasgos (RANSAC)

En el cálculo de la matriz de homografía se procesan los puntos de los emparejamientos detectados en ambas imágenes, mediante el método estadístico RANSAC se calculan los parámetros de la matriz. La función utilizada es:

```
(H,status)=cv2.findHomography(pointsA,pointsB,cv2.RANSAC,max_Threshold)
```

D. Trasformación geométrica (Warping)

Con la matriz de homografía calculada, la imagen que se ubique en el extremo derecho será transformada geométricamente para coincidir con la imagen en el extremo izquierdo, este proceso se explicó en el capítulo 1, de esta forma ambas imágenes se concatenan para crear el mosaico compuesto. Las funciones que se utilizaron son:

Se define el tamaño que tendrán las dos imágenes a unirse:

```
val = imageA.shape[1] + imageB.shape[1]
```

Se hace la transformación geométrica a la “imagenA” (imagen derecha):

```
result_image = cv2.warpPerspective(imageA, Homography, (val , imageA.shape[0]))
```

Se concatena la imagen izquierda con la imagen derecha transformada:

```
result_image[0:imageB.shape[0], 0:imageB.shape[1]] = imageB
```

E. Mezcla de imágenes (Blending)

Después de la unión de cada par de imágenes queda un borde que hace aparente la discontinuidad de intensidad entre ambas, Por medio del proceso de mezclado se puede desvanecer esta discontinuidad, los pasos del proceso de forma general son: 1- Creación de pirámides de gauss y laplace de ambas imágenes, 2- Reconstrucción de imágenes mediante suma de imágenes en pirámides de laplace. A continuación de muestra las funciones utilizadas:

Se crea pirámide gaussianas de imágenes A y B, se crean copias de imágenes a seis escalas diferentes y se almacenan en arreglo gpA .

```
G=cv2.pyrDown(gpA)
gpA.append(G)
```

A partir de las piramides gaussianas se crean la piramides de laplace de A y B.
Se resta la imagen de escala inferior en gpA contra la imgen de escala superior inmediata en gpA, se tiene que incrementar escala de la primera para poder realizar la resta. Las piramides laplacianas se almacenan en arreglo lpA, lpB.

```
GE=cv2.pyrup(gpA[i])
L=v2.subtract(gpA[i-1]-GE
lpA.append(L)
```

Se concatenan las imágenes en la piramide laplaciana de imágenes A y B de acuerdo a su escala donde “y” representa la union donde se presentaba la diferencia de intensidades en la imagen original.

```
ls=np.hstack((la[:,0:y],lb[:,y:])
LS.append(ls)
```

Se incrementa escala para sumar todas las imágenes en la piramide laplaciana. El resultado es la imagen mezclada almacenada en ls_.

```
ls_=LS[0]
cv2.pyrUp(ls__)
ls_=cv2.add(ls_,LS[i])
```

Figura 4. Imagen con Mezcla real.

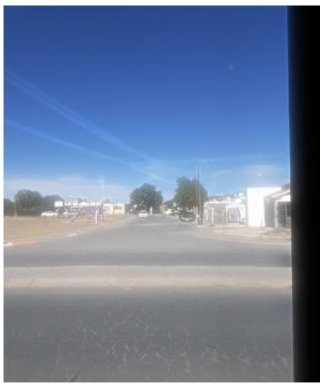

Figura 5. Imagen con mezcla desvanecida.

F. Corrección de desviaciones (Bundle adjustment)

Una vez que son procesados todos los mosaicos y se tiene el panorama final, se procede a eliminar las desviaciones que podrían haber sido causadas durante la etapa de transformación geométrica, causando bordes sobrantes o incompletos. En esta etapa se binariza la imagen, por medio de una función se ubican los contornos presentes en la imagen, y por medio de sus puntos se reconstruye un rectangulo vertical que indicara los limites dentro de los cuales la imagen no presenta desviaciones. Las funciones utilizadas son:

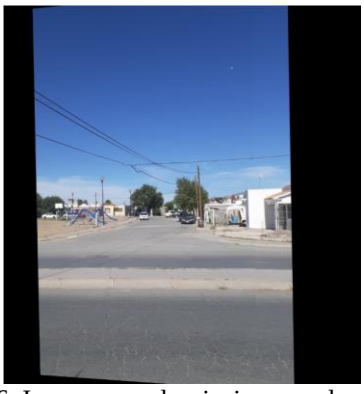

Figura 6. Imagen con desviaciones en los bordes.

Se detectan los contornos presentes en la imagen:

```
cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
c = max(cnts, key=cv2.contourArea)
```

Se reconstruye un rectángulo vertical por medio de los puntos que conforman los contornos.

```
(x, y, w, h) = cv2.boundingRect(c)
```

Se crea una imagen con el tamaño de la imagen original, y solo se dibuja el rectángulo reconstruido detectado con valores de blanco.

```
mask = np.zeros(thresh.shape, dtype="uint8")
cv2.rectangle(mask, (x, y), (x + w, y + h), 255, -1)
```

Se hace una resta entre la imagen original y la imagen del rectángulo blanco. El resultado dará las coordenadas donde se ubique el rectángulo, que contiene las partes de la imagen que no presentan desviación.

```
sub = cv2.subtract(minRect,thresh)
(x, y, w, h) = cv2.boundingRect(c)
result = result[10:y + h, 10:x + w]
```

III. RESULTADOS

A continuación, se muestran los resultados del proceso de costura, se muestra el proceso resumido solo para la costura del ultimo mosaico si se desea ver con mas detalle el proceso que siguió cada mosaico se puede consultar el archivo Resumen_costura.xlsx ubicado en la carpeta del código fuente.

Figura 7. Imagen para proceso de costura

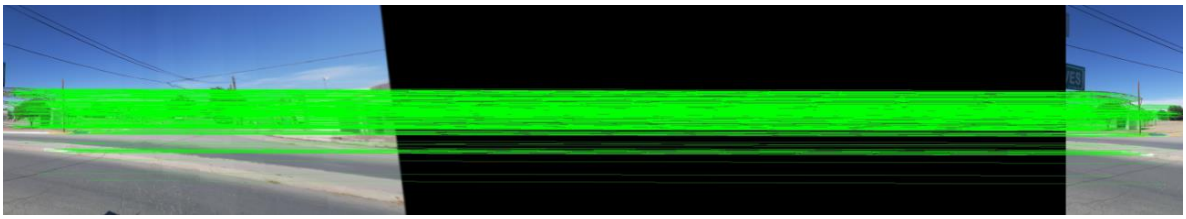

Figura 8. Detección y emparejamiento de rasgos en mosaico actual y mosaico por unir.

Figura 9. Transformación geométrica a mosaico en lado derecho.

Figura 10. Imágenes unidas por medio de método de costura.

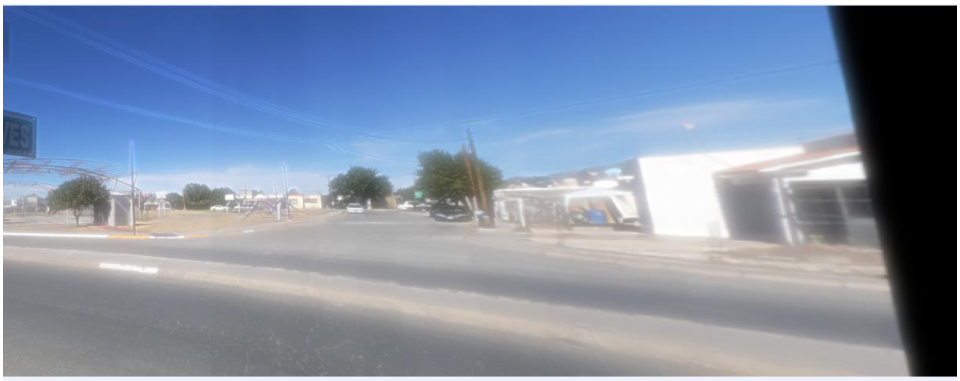

Figura 11. Método de desvanecimiento aplicado a mosaico anterior con respecto a el mosaico unido.

Figura 12. Panorama Generado sin método de desvanecimiento.

Figura 13. Panorama Generado con método de desvanecimiento aplicado solo a una unión, imagen (2).

Figura 14. Panorama Generado con método de desvanecimiento aplicado a cada imagen que se unió.

IV. Conclusiones

En esta práctica se desarrolló un método para costura de varias imágenes, fue posible visualizar cada etapa del proceso, se observó la desviación que se genera durante la transformación geométrica, algo que con una función prediseñada no es posible dado que solo mostraría el panorama final. Una de las oportunidades detectadas se encuentra en el proceso de desvanecimiento, por tratarse de la unión de varias imágenes, cada imagen provoca un factor de desvanecimiento en el panorama, así cada imagen que se continúe agregando agregara un factor adicional, esto se puede visualizar en el panorama resultante con desvanecido completo (figura 14), así como el resultado al aplicar este proceso solo a la unión de un par de mosaicos (figura 13). También se generó un mosaico sin aplicar el proceso de mezclado donde se puede encontrar las líneas verticales provenientes del proceso de costura (figura 12)

V. Código Fuente

A. Costura.py

```
from Operaciones_Rev4 import *
import imutils
from imutils import paths
import cv2

#############################################################################################
#
################################Costura de imagenes#########################################
#############################################################################################
#
print("El siguiente programa realiza la costura de\n\
    una serie de imagenes que seran colocadas en la carpeta ./images/\n\
    solo renombre las imagenes de forma incremental con las letras del abecedario\n\
    siendo -a- la imagen izquierda superior -b- la imagen adyacente inmediata\n")
opcion=int(input("Indique si las imagenes se degradaran/mezclaran\n\
    para cada 2 imagenes o se mezcalara el total de imagenes\n\
    1-Mezclar c/Par de imagenes\n\
    2-Mezclar el total de imagenes\n Opcion:_"))

print("Procesando...\n")



#Carga de imagenes de folder ./images

imagePaths = sorted(list(paths.list_images('./images/')))
images = []

for imagePath in imagePaths:
        image = cv2.imread(imagePath)
        images.append(image)

no_of_images=len(images)
counter=no_of_images
```

```
#Se ajustan las dimenciones de las imagenes para que todas sean iguales

for i in range(no_of_images):
    images[i] = imutils.resize(images[i], width=1000)
    #pyramida(images[8])
for i in range(no_of_images):
    images[i] = imutils.resize(images[i], height=1000)


if no_of_images==2:
    x=no_of_images-2
    (result, matched_points) = image_stitch([images[0], images[1]],True,x,opcion)

else:
    #Se procesan las dos primeras imagenes que serviran como el mosaico de referencia
    x=no_of_images-2
    (result, matched_points) = image_stitch([images[no_of_images-2], images[no_of_images-1]],
True,x,opcion)
    cv2.imwrite('Reference_image.jpg',images[0])
    cv2.imwrite('Matches'+str(x)+'.jpg',matched_points)
    cv2.imwrite('result'+str(x)+'.jpg',result)

    #Se procesan las imagenes adyacntes de forma secuencial y se van uniendo al mosaico de referencia
    for i in range(no_of_images - 2):
        x=no_of_images-i-3
        (result, matched_points) = image_stitch([images[x],result],True,x,opcion)
        cv2.imwrite('Matches'+str(x)+'.jpg',matched_points)
        cv2.imwrite('result'+str(x)+'.jpg',result)

#Se muestra el ortomosaico sin ningun recorte
cv2.imwrite('No recortado.jpg',result)
#Se recortan las esquinas excedentes
result=cropping(result)


#Se muestran el Panorama completo con bordes recortados
cv2.imwrite("Panorama.jpg",result)
print("Proceso terminado, El Panorama generado es: Panorama.jpg \n")

cv2.waitKey(0)
cv2.destroyAllWindows()
```

B. Operaciones.py

```
import numpy as np
import imutils
import cv2
from past.builtins import xrange


#Deteccion de rasgos en entre cada par de mosaicos
def Detect_Feature_And_KeyPoints(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

```
    # detect and extract features from the image
    descriptors = cv2.xfeatures2d.SIFT_create()
    (Keypoints, features) = descriptors.detectAndCompute(image, None)

    Keypoints = np.float32([i.pt for i in Keypoints])
    return (Keypoints, features)


#Calcula coincidencias etre los rasgo de los mosaicos
def get_Allpossible_Match(featuresA,featuresB):
    match_instance = cv2.DescriptorMatcher_create("BruteForce")
    All_Matches = match_instance.knnMatch(featuresA, featuresB, 2)
    return All_Matches

#Se obtienen las coincidencias entre rasgos que son validos
#y se encuentran dentro de un radio de busqueda (Lowe_radio)
def All_validmatches(AllMatches,lowe_ratio):
    valid_matches = []

    for val in AllMatches:
        if len(val) == 2 and val[0].distance < val[1].distance * lowe_ratio:
            valid_matches.append((val[0].trainIdx, val[0].queryIdx))

    return valid_matches


#Calcula la matriz de homografia, tomando entrada las coordenadas de puntos coincidentes
#en ambas imagenes
def Compute_Homography(pointsA,pointsB,max_Threshold):
    (H, status) = cv2.findHomography(pointsA, pointsB, cv2.RANSAC, max_Threshold)
    return (H,status)

#Llamadas a 3 process: Busqueda conincidencias, Busqueda coincidencias validas y Calculo de
#matriz de homografia
def matchKeypoints( KeypointsA, KeypointsB, featuresA, featuresB,lowe_ratio, max_Threshold):

    AllMatches = get_Allpossible_Match(featuresA,featuresB);
    valid_matches = All_validmatches(AllMatches,lowe_ratio)

    if len(valid_matches) > 4:
        # construct the two sets of points
        pointsA = np.float32([KeypointsA[i] for (_,i) in valid_matches])
        pointsB = np.float32([KeypointsB[i] for (i,_) in valid_matches])

        (Homograpgy, status) = Compute_Homography(pointsA, pointsB, max_Threshold)

        return (valid_matches, Homograpgy, status)
    else:
        return None

#Se realiza una proyeccion perspectiva a la imagen a utilizando la
#matriz de homografia calculada.
def getwarp_perspective(imageA,imageB,Homography):
    val = imageA.shape[1] + imageB.shape[1]
    result_image = cv2.warpPerspective(imageA, Homography, (val , imageA.shape[0]))
```

```
    return result_image

#Obtiene las dimensiones de una imagen, ancho y alto.
def get_image_dimension(image):
    (h,w) = image.shape[:2]
    return (h,w)
#crea matriz donde se almacenan los puntos coincidentes detectados
def get_points(imageA,imageB):

    (hA, wA) = get_image_dimension(imageA)
    (hB, wB) = get_image_dimension(imageB)
    vis = np.zeros((max(hA, hB), wA + wB, 3), dtype="uint8")
    vis[0:hA, 0:wA] = imageA
    vis[0:hB, wA:] = imageB

    return vis

#Dibuja los puntos coincidentes detectados entre dos imagenes en la matriz vis.
def draw_Matches(imageA, imageB, KeypointsA, KeypointsB, matches, status):

    (hA,wA) = get_image_dimension(imageA)
    vis = get_points(imageA,imageB)

    # loop over the matches
    for ((trainIdx, queryIdx), s) in zip(matches, status):
        if s == 1:
            ptA = (int(KeypointsA[queryIdx][0]), int(KeypointsA[queryIdx][1]))
            ptB = (int(KeypointsB[trainIdx][0]) + wA, int(KeypointsB[trainIdx][1]))
            cv2.line(vis, ptA, ptB, (0, 255, 0), 1)

    return vis

#Crea degradado e el borde de unione de dos imagenes que se unen
def blending(A,B,Yv):
    # Genera piramide guasiana para imagen A
    G = A.copy()
    gpA = [G]
    for i in xrange(6):
        G = cv2.pyrDown(gpA[i])
        gpA.append(G)

    # Genera piramide laplaciana para imagen A
    lpA = [gpA[5]]
    for i in xrange(5,0,-1):
        size = (gpA[i-1].shape[1], gpA[i-1].shape[0])
        GE = cv2.pyrUp(gpA[i], dstsize = size)
        L = cv2.subtract(gpA[i-1],GE)
        lpA.append(L)


    # Genera piramide gausiana para imagen B
    G = B.copy()
    gpB = [G]
    for i in xrange(6):
        G = cv2.pyrDown(gpB[i])
        gpB.append(G)
```

```
    # Genera piramide laplaciana para imagen B
    lpB = [gpB[5]]
    for i in xrange(5,0,-1):
        size = (gpB[i-1].shape[1], gpB[i-1].shape[0])
        GE = cv2.pyrUp(gpB[i], dstsize = size)
        L = cv2.subtract(gpB[i-1],GE)
        lpB.append(L)


    #Concatena las imagenes creadas a partir de piramides laplacianas
    #he indica las cordendas de union en la variable Yv.
    LS = []
    for la,lb in zip(lpA,lpB):
        rows,cols,dpt = la.shape
        y1=Yv
        ls = np.hstack((la[:,0:y1], lb[:,y1:]))
        LS.append(ls)

    #Reconstruye La imagen a partir de las piramides calculadas y
    #la almacena en la variable ls_
    ls_ = LS[0]
    for i in xrange(1,6):
        size = (LS[i].shape[1], LS[i].shape[0])
        ls_ = cv2.pyrUp(ls_, dstsize = size)
        ls_ = cv2.add(ls_, LS[i])


    #Muestra la mezcla directa de la imagenes
    y2=Yv
    real = np.hstack((A[:,:y2],B[:,y2:]))
    cv2.imwrite('a-Pyramid_blending.jpg',ls_)
    cv2.imwrite('a-Direct_blending.jpg',real)
    return ls_

#Se realiza la costura de imagenes, para eso es necesario llamar a procesos: deteccion y emparejamiento
#de rasgos, proyeccion perspectiva y degradado de imagenes.
def image_stitch(images, match_status,x,opcion):
    lowe_ratio=0.75
    max_Threshold=4.0

    (imageB, imageA) = images
    (KeypointsA, features_of_A) = Detect_Feature_And_KeyPoints(imageA)
    (KeypointsB, features_of_B) = Detect_Feature_And_KeyPoints(imageB)

    #Llama a funcion para obtener emparejamientos de rasgos
    Values = matchKeypoints(KeypointsA, KeypointsB,features_of_A, features_of_B, lowe_ratio,
max_Threshold)

    if Values is None:
        return None

    #Llama a funcion para  obtener la proyeccion perspectiva de imagen A
    (matches, Homography, status) = Values
    result_image =getwarp_perspective(imageA,imageB,Homography)
    cv2.imwrite("warped_image"+str(x)+'.jpg',result_image)
```

```
    #Concatena la imagen de referencia con la imagen proyectada y los une por
    #los extremos con relacion a sus puntos coincidentes (costura).
    result_image[0:imageB.shape[0], 0:imageB.shape[1]] = imageB

    #Realiza el desvanecimiento (mezcla) de las imagenes concatenadas.
    if opcion==1:
       lsblending=blending(result_image,result_image,imageB.shape[1])
       cv2.imwrite("blend"+str(x)+".jpg",lsblending)

    if opcion==2:
       result_image=blending(result_image,result_image,imageB.shape[1])


    if match_status:
       vis = draw_Matches(imageA, imageB, KeypointsA, KeypointsB, matches,status)

       return (result_image, vis)

    return result_image

def cropping(result):

    # Se crea un rectangulo con bordes de 2 pixeles
    print("Reduciendo contornos ...")
    result = cv2.copyMakeBorder(result, 2, 2, 2, 2,
            cv2.BORDER_CONSTANT, (0, 0, 0))

    #Convierte la imagen panorama a blanco y negro.

    gray = cv2.cvtColor(result, cv2.COLOR_BGR2GRAY)
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY)[1]


    #Se buscan los contornos en la imagen, el contorno externo es un
    #cuadrado
    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
            cv2.CHAIN_APPROX_SIMPLE)

    cnts = imutils.grab_contours(cnts)
    c = max(cnts, key=cv2.contourArea)

    #se crea una imagen de mascara con la misma dimension
    #que la imagen binarizada, con valores cero
    mask = np.zeros(thresh.shape, dtype="uint8")
    #Se genera el rectangulo mas pequeño que se puede generar
    #verticalmente con los contornos detectados
    (x, y, w, h) = cv2.boundingRect(c)
    #Se dibuja el rectangulo (relleno de blanco, "-1") mas pequeño con los parametros
    #identificados
    cv2.rectangle(mask, (x, y), (x + w, y + h), 255, -1)

    #Se crean dos copias de la imagen mascara
    minRect = mask.copy()
    sub = mask.copy()
```

```
#Se resta el rectangulo minimo (ceros) menos la imagen binarizada,
#esto vacia el interior del area de la imagen, y solo con serva el
#contorno.
while cv2.countNonZero(sub) > 0:
        #se erosiona rectangulo minimo, para facilitar
    #la operacion de substraccion
    minRect = cv2.erode(minRect, None)
    sub = cv2.subtract(minRect,thresh)

#Se detectan los contornos del rectangulo minimo
cnts = cv2.findContours(minRect.copy(), cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
c = max(cnts, key=cv2.contourArea)
#Se detectan una vez mas el rectangulo mas pequeño que se puede calcular con
#los puntos detectados
(x, y, w, h) = cv2.boundingRect(c)

#Muestra los limites de la imagen resultante, utilizando como limite
#el rectangulo calculado.
result = result[10:y + h, 10:x + w]
return result
```

VI. BIBLIOGRAFÍA.

[1] pyimagesearch, Image Stitching with OpenCV and Python, https://www.pyimagesearch.com, https://www.pyimagesearch.com/2018/12/17/image-stitching-with-opencv-and-python/, [online], Ultimo acceso: 2 junio 2020.

[2] pyimagesearch, OpenCV panorama stitching, www.pyimagesearch.com, https://www.pyimagesearch.com/2016/01/11/opencv-panorama-stitching/, Ultimo acceso: 2 junio 2020.

[3] OPENCV, samples/python/stitching.py, www.opencv.org, https://docs.opencv.org/master/d5/d48/samples_2python_2stitching_8py-example.html, [online], 2 junio 2020.

[4] learnopencv, Image Alignment (Feature Based) using OpenCV (C++/Python), www.learnopencv.com, https://www.learnopencv.com/image-alignment-feature-based-using-opencv-c-python/, [online], Ultimo acceso: 2 junio 2020.

[5] OPENCV, Feature Matching + Homography to find Objects, opencv-python-tutroals.readthedocs.io, https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_feature_homography/py_feature_homography.html, [online], Ultimo acceso: 2 junio 2020.

[6] OPENCV, Geometric Image Transformations, www.opencv.org, https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html#void%20warpPerspective(InputArray%20src,%20OutputArray%20dst,%20InputArray%20M,%20Size%20dsize,%20int%20flags,%20int%20borderMode,%20const%20Scalar&%20borderValue), [online], Ultimo acceso: 2 junio 2020.

[7] OPENCV, Image Pyramids, www.opencv.org, https://docs.opencv.org/3.1.0/dc/dff/tutorial_py_pyramids.html, [online], Ultimo acceso: 2 junio 2020.

[8] UNIVERSITY OF WISCONSIN–MADISON, Panoramic Image Mosaic, http://pages.cs.wisc.edu/, http://pages.cs.wisc.edu/~csverma/CS766_09/ImageMosaic/imagemosaic.html, [online], Ultimo acceso: 2 junio 2020.