Computer Vision
Assignment 8

Alireza Moradi

November 15, 2020

1 Similarity Transform

For a certain number of iterations:

1. We choose some random points.(At least 2 in case of similarity transform)
2. Compute derivatives of cost function with respect to the parameters.
3. Find equations for each parameter by setting $\frac{\partial cost}{\partial param} = 0$.
4. Apply the transformation on all points.
5. Points which are near their true positions vote for the transformation.
6. If the new transformation is better than the previous best one, replace it.

After the loop, we have the best transformation, So we do the following:

1. Find all the points which have voted for the best transformation found in the previous section. The rest of the points are outliers.
2. Compute the final transformation with these points.

2 Implementations

2.1 Stitch

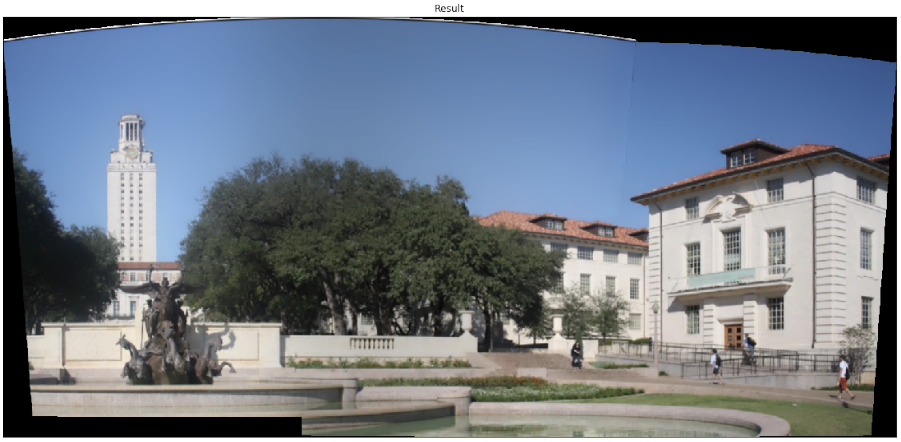
The algorithm used in `openCV` is similar to the method proposed by Brown and Lowe in their 2007 paper, Automatic Panoramic Image Stitching with Invariant Features which is insensitive to:

- Ordering of images
- Orientation of images
- Illumination changes
- Noisy images that are not actually part of the panorama.

The `cv2.Stitcher_create` is used to create and instance of the stitcher. To perform the actual image stitching we'll need to call the `.stitch` method which method accepts a list of input images , and then attempts to stitch them into a panorama, returning the output panorama image to the calling function.
The `status` variable indicates whether or not the image stitching was a success and can be one of four variables:

- `OK = 0` : The image stitching was a success.
- `ERR_NEED_MORE_IMGS = 1` : In the event you receive this status code, you will need more input images to construct your panorama. Typically this error occurs if there are not enough keypoints detected in your input images.
- `ERR_HOMOGRAPHY_EST_FAIL = 2` : This error occurs when the RANSAC homography estimation fails. Again, you may need more images or your images don't have enough distinguishing, unique texture/objects for keypoints to be accurately matched.
- `ERR_CAMERA_PARAMS_ADJUST_FAIL = 3` : It is related to failing to properly estimate camera intrinsics/extrinsics from the input images.

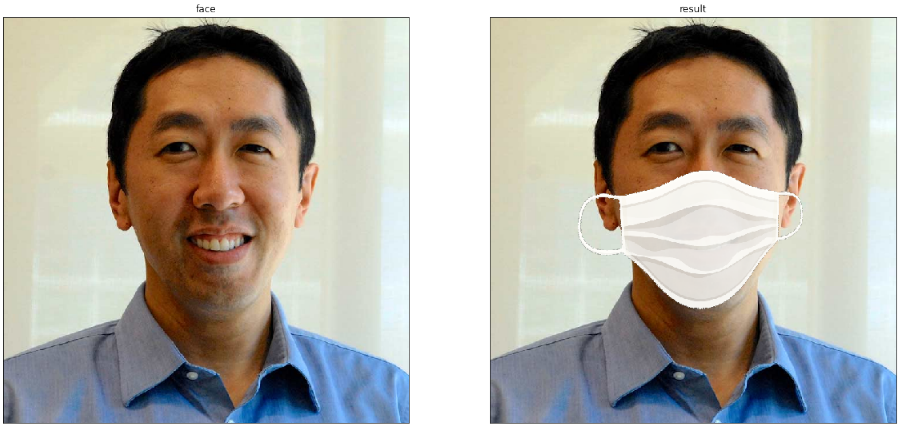
Source



```python
def stitch(image1, image2):
    '''
    Creates panorama image of two inputs.

    Parameters:
        image1 (numpy.ndarray): The first input image.
        image2 (numpy.ndarray): The second input image.

    Returns:
        numpy.ndarray: The result panorama image.
    '''

    out_img = None

    #Write your code here
    stitcher = cv2.Stitcher_create()
    status, out_img = stitcher.stitch([image1,image2])

    return out_img
```

2.2 Mask

`rect_to_bb` , short for “rectangle to bounding box”, accepts a single argument, `rect` , which is assumed to be a bounding box rectangle produced by a `dlib` detector (i.e., the face detector).
The `rect` object includes the *(x, y)-coordinates* of the detection.
However, in OpenCV, we normally think of a bounding box in terms of *(x, y, width, height)* so as a matter of convenience, the `rect_to_bb` function takes this `rect` object and transforms it into a 4-tuple of coordinates.
The `dlib` face landmark detector will return a shape object containing the *68 (x, y)-coordinates* of the facial landmark regions.
Using the `shape_to_np` function, I convert this object to a NumPy array.
Then we create a predictor by loading dlib’s pre-trained facial landmark detector.
I manually selected 12 landmarks on the mask photo using this and exported them as a `csv` file.
Then I used `findHomography` which finds a perspective transformation between two planes and `warpPerspective` which applies a perspective transformation to an image.
Source



```
def rect_to_bb(rect):
        x = rect.left()
        y = rect.top()
        w = rect.right() - x
        h = rect.bottom() - y
        return (x, y, w, h)

def shape_to_np(shape, dtype="int"):
        coords = np.zeros((68, 2), dtype=dtype)
        for i in range(0, 68):
                coords[i] = (shape.part(i).x, shape.part(i).y)
        return coords
```

```
def put_mask(face, mask):
    '''
    Adds mask image on face image.

    Parameters:
        face (numpy.ndarray): face image.
        mask (numpy.ndarray): mask image.

    Returns:
        numpy.ndarray: The result image.
    '''

    result = face.copy()
    result = result.astype(np.float32)
    result = result / 255.0

    detector = dlib.get_frontal_face_detector()
    gray_andrew = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY)
    predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
    rects = detector(gray_andrew, 1)

    shape = predictor(gray_andrew, rects[0])
    shape = shape_to_np(shape)

    h, w, c = mask.shape
    hf,wf,cf = face.shape
    dst = np.array([shape[15],
                   shape[14],
                   shape[12],
                    shape[11],
                    shape[10],
                   shape[8],
                    shape[6],
                    shape[5],
                    shape[4],
                    shape[2],
                    shape[1],
                    shape[30],
                   ], dtype = "float32")

    with open("mask.csv") as csv_file:
        import csv
        csv_reader = csv.reader(csv_file, delimiter=",")
        src = []
        for i, row in enumerate(csv_reader):
            try:
                src.append(np.array([float(row[1]), float(row[2])]))
            except ValueError:
                continue
    src = np.array(src, dtype="float32")

    M, _ = cv2.findHomography(src, dst)
    transformed_mask = cv2.warpPerspective(
        mask,
        M,
        (result.shape[1], result.shape[0]),
        None,
        cv2.INTER_LINEAR,
        cv2.BORDER_CONSTANT,)
    alpha_mask = transformed_mask[:, :, 3]
    alpha_image = 1.0 - alpha_mask
    for c in range(0, 3):
        result[:, :, c] = (
            alpha_mask * transformed_mask[:, :, c]
            + alpha_image * result[:, :, c])
    return result
```