Computer Vision – 046746

Hands-On Assignment

#04

Authors:

Gonen Weiss – 312347982

Alex Balabanov – 312775364

Due date:

July 2nd, 2020

Part 1 – Theory

Q1:

Let x and x' be the projections of P onto the two image planes in each's image coordinates. As a property of the fundamental matrix, we know that:

$$x'^T F x = 0$$

However, it is given that the coordinates origin (0,0) coincide with each principal points x, x', meaning that

$$x = x' = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Hence, we get:

$$F_{33} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = x'^T F x = 0$$

Q2:

We know that the Essential matrix E equals:

$$E = R[t_x]$$

Where R and t are the rotation matrix and translation vector between the two cameras' coordinate systems, and $[t_x]$ corresponds to the skew-symmetric matrix form of the cross-product with t:

$$[t_x] = \begin{bmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{bmatrix}$$

However, it is given that the seconds camera differs from the first by a pure translation parallel to the x-axis. Hence, we derive that:

$$R = I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}; t = \begin{bmatrix} d \\ 0 \\ 0 \end{bmatrix}$$

And the essential matrix becomes:

$$E = R[t_x] = I[t_x] = [t_x] = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -d \\ 0 & d & 0 \end{bmatrix}$$

Let us observe the second camera. Every epipolar line l' can be expressed as $l' = Ex$ for a certain point x in the first camera's plane. For a general point x we get:

$$x = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \Longrightarrow l' = Ex = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -d \\ 0 & d & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0 \\ -dZ \\ dY \end{bmatrix}$$

And such line is parallel to the x-axis, because, for instance, the two x_1, x_2 below define a line uniquely, both located on l' because $x_1^T l' = x_2^T l' = 0$ and their translation is obviously parallel to the x-axis.

$$x_1 = \begin{bmatrix} 1 \\ Y \\ Z \end{bmatrix}, x_2 = \begin{bmatrix} 2 \\ Y \\ Z \end{bmatrix}$$

The same arguments go for the first camera, by symmetry.

Q3:

Let us assume without loss of generality that there exists a universal coordinate system, toward which we all Ri-s and ti-s refer.

Rrel is the relative rotation from time i coordinate system to time j coordinate system. It can be achieved by inverse-rotating from time i to the universal, and then forward rotating to time j:

$$R_{rel} = R_j R_i^{-1} = R_j R_i^T$$

trel is the relative translation from time i coordinate system to time j coordinate system. It can be achieved by inverse-translating from time i to the universal, and then forward translating to time j:

$$t_{rel} = t_j - t_i$$

Now, given those two expressions and the camera's intrinsic matrix K, we know that the Essential and Fundamental matrices equal:

$$E = R_{rel}\left[t_{rel_x}\right]$$

$$F = K^{-T} E K$$

Where $[a_x]$ corresponds to the skew-symmetric matrix form of the cross-product with a:

$$[a_x] = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}$$

Further substitutions will not provide any neater results.

Part 2 – Planar Homographs

Q2.0:

Given N corresponding points {p, q} we can estimate a H matrix that projects a homogeneous coordinated of qi to pi.

Observe the calculation matrix

$$\begin{matrix} p_i = (x_i, y_i, 1) \\ q_i = (u_i, v_i, 1) \end{matrix} \quad H = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix}$$

$$p_i^t \equiv H q_i^t$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \gamma \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \cdot \begin{bmatrix} u \\ y \\ 1 \end{bmatrix}$$

After breaking it down we get

$$x = \gamma(a \cdot u + b \cdot v + c)$$

$$y = \gamma(d \cdot u + e \cdot v + f)$$

$$1 = \gamma(g \cdot u + h \cdot v + 1)$$

From the last equation we can find a value of γ

$$\gamma = (g \cdot u + h \cdot v + 1)^{-1}$$

$$x \cdot (g \cdot u + h \cdot v + 1) = a \cdot u + b \cdot v + c$$

$$y \cdot (g \cdot u + h \cdot v + 1) = d \cdot u + e \cdot v + f$$

We will move everything to 1 side and extract the matrix coefficients

$$u \cdot a + v \cdot b + c - xu \cdot g - xv \cdot h - x = 0$$

$$d \cdot u + e \cdot v + f - yu \cdot g - yv \cdot h - y = 0$$

For every pair of points {p, q} we get

$$\begin{bmatrix} u & v & 1 & 0 & 0 & 0 & -xu & -xv & -x \\ 0 & 0 & 0 & u & v & 1 & -yu & -yv & -y \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ 1 \end{bmatrix} = \bar{0}$$

Let us define the equation as:

$$A \cdot h = 0$$

Where A is a $9 \times 2N$ matrix consisting of point coordinates, and h is a vector made from matrix H values.

Looking and the equation it is clear that every pair of points sets 2 constrains, with the 8 DOF that matrix H has there should be 4 linearly independent pair of points to find a solution.

Q2.1

Let us define the image that will be warped to as im1, and the image that we want to project on im1 as im2.

Taking 2 pictures as different subplots, side by side, we have implemented a manual feature matcher that enables choosing point matches between the two images in the following order: START -> Left -> Right -> Left -> Right -> … -> Left -> Right → END

The sequence is:

- Im1 – pick a feature in the image
- Im2 – find the corresponding feature

Repeat N times

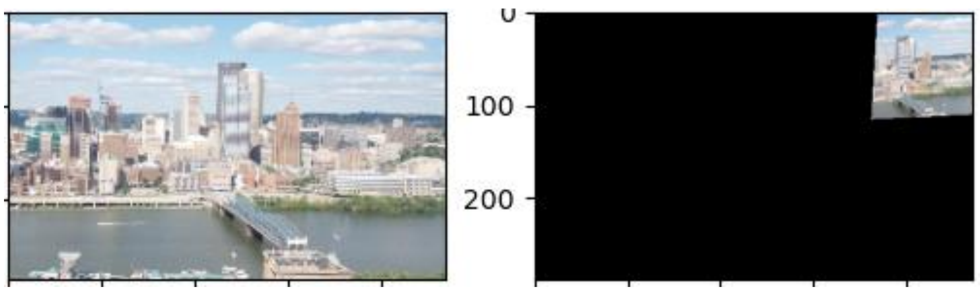
Q2.2

Looking back at Q2.0, it is possible to see a method to find the matrix H, after taking the lists of corresponding points, we will create matrix A.

Matrix A will not be a square matrix, so we will take SVD approach as seen in tutorial 8.

Using SVD is beneficial in this instance, since there can be a strong noise factor on the points we had found.

Below is an example of warping for arbitrary chosen points, as requested.



Q2.3

Image warping should be a straightforward task, it is a per pixel function.

Taking each pixel in the input image (img) changing it into homogeneous coordinated, applying the transformation matrix and getting the new coordinated.

Unfortunately, pixel coordinated are discrete, and there might be pixels in the new image that are missing information. Those "holes" won't be similar to "salt & papper" noise that can be removed with a median filter. They come in "paths" that correspond with the transformation so dealing with it is much harder. Similarly, the wrapped pixels do not form a square, so using an interpolation probably will not work and take longer to calculate. Therefore, we used a reverse approach, creating a blank warped (warp) image and working backwards. Taking each pixel coordinates in warp and checking where it should be on the original (img), using a reverse transform of H. The resulting coordinates are checked if in bounds and interpolated with img. This method ensures getting a continuous warped image.

Also, we have used the LAB color space and the linear interpolations. Comparison, as we were asked, is discussed below.

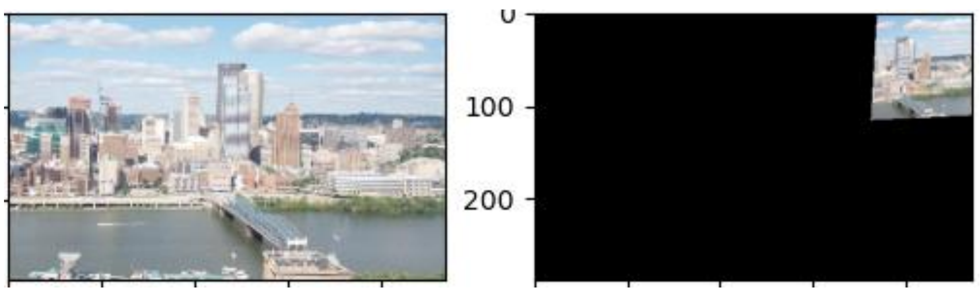
LAB vs. RGB:

> LAB is a color-space that is designed to approximate human vision. For that reason, using the LAB color space while applying interpolations produces more natural results to the human eye, since midpoints differences will contain more perceptible information. However, we should remark that the difference in results is not very significant.

Linear vs. Cubic:

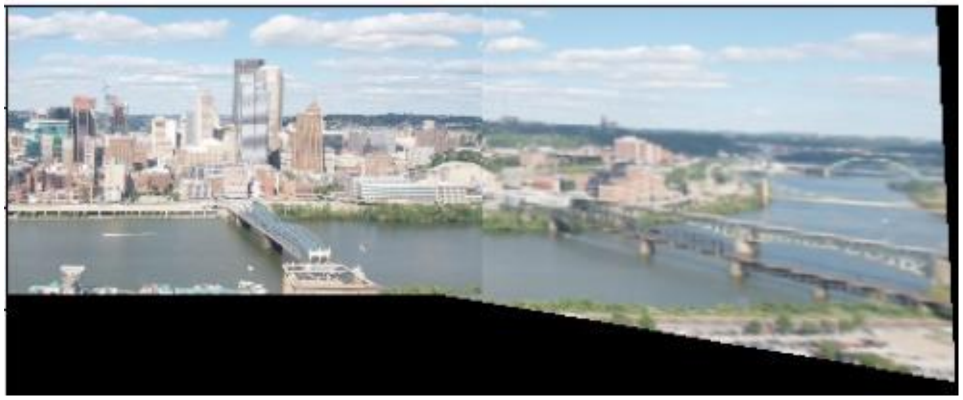
> Linear interpolation uses 4 nearest neighbors for prediction, while cubic uses 16. As such, it is more cubic results would often produce smoother results, with the cost of higher computation complexity. Most of the time the difference between the two is not very significant, and linear interpolation is a fine tradeoff of complexity and continuity.

The same example as before is provided, for the demonstration of *warpH* functionality.



Q2.4

Panorama stitching includes 2 main actions, finding a mask for both images and blending nicely. Since img1 was not wrapped it contains less interpolation artifacts so we will use all of it. Wrap_img2 needs to be masked to the information that does not overlap with img1 and added to the image.



Seemingly we have received nice stitching results, with slight incontinuity between images due to SIFT detecting noise.

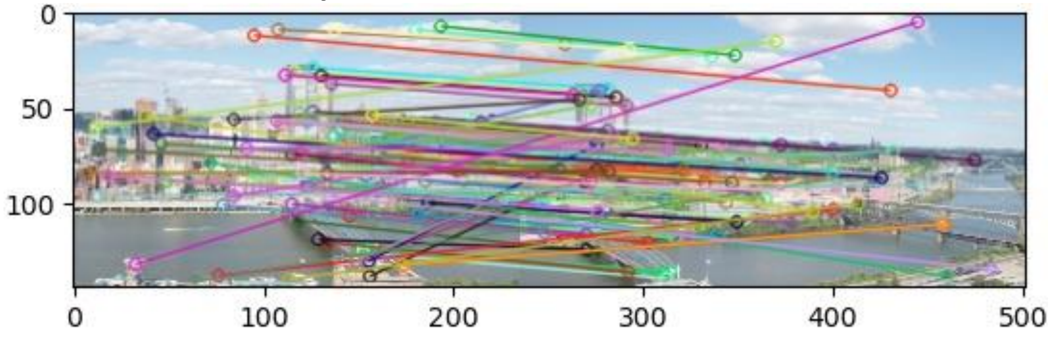
Q2.5

Autonomous stitching can be as good as its key point extractor and matcher. First we tried using ORB to get key points (KP) and descriptors (DS) but it didn't work well, so we used a SIFT algorithm from git hub as was recommended (https://github.com/rmislam/PythonSIFT).

SIFT works similarly to what we had implemented on Hw1, finding edges and corners with difference of Gaussian pyramid (DoG pyramid), using Sobel filter (differentials) to filter out edges and leave most corners, and finally using an anker filter to get a numerical descriptor of the corner.

Once both images are calculated we need to find matching KP, likely we have DS that should be mostly invariant, the matcher will take each both lists of KP, DS and match them one by one, therefore it is called brute force.

Not all matches are great so we will want to find as many good matches and no outliners. Distance can be really misleading measurement, even if we use KNN to have a estimation how sure is it. Therefore, the only manipulation we will do is to sort them by distance and use RANSAC as described in Q2.8



Q2.6

The two methods are essentially different. The manual approach is most of the time a more precise one, however it is much more costly in manpower, and often in time on general as well. However, SIFT detector is very time consuming as well, since the algorithm complexity is not negligible.

Below are few points of comparison between the SIFT and the manual approach:

- Automated vs. manual – SIFT allows embedding the algorithm in an automated script, performing more complex tasks automatically. Furthermore it can be implemented to work relatively fast, sufficient even for run-time programs.
- PC resources vs. manpower resources – SIFT runs on machine resources, which is often much less expensive.
- A lot of keypoints vs. little keypoints – SIFT finds a lot of keypoints most of the time. By that, it can reduce its own noise of inaccuracies, based on big numbers, compensating the human's edge in this manner (for a human this would take too long).
- Number of detections tradeoff flexibility – In SIFT it is harder to preform quick matching for small number of points, since most of the time it first finds keypoints and then filter out bad ones. For a human, little yet fast keypoints is the comfort zone. Cons:
- Entire mistakes vs. slight misplacing – SIFT quite often finds entirely wrong matches, while a human usually doesn't. The human's inaccuracies are most of the time a result of resolution, and the error is much lower.

Q2.7

First, we will run both algorithms manually and automatically on both image sets, and compare the results.

As stated in Q2.5 stitching can be as good as it's KP finder, obviously a human can do the task better and the results are visibly better, however human can't go through huge data sets and that is the manual method downfall. Manual

Q2.8

RANSAC's advantage is not the random factor, but the iterative nature. Lists p1, p2 are sorted by distance, we will tap into it by using a weighted randomization. We will give higher probability for KP in the head of the lists, thus increasing the chance to find the few good pairs that include most other pairs. We picked 8 points at a time to filter out noise in the data.

Q2.11: Affine vs Projective - BONUS:

Given N corresponding points {p, q} we can estimate the A matrix of an affine transformation from qi to pi.

Observe the calculation matrix

$$\begin{matrix} p_i = (x_i, y_i, 1) \\ q_i = (u_i, v_i, 1) \end{matrix} \quad A = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

$$p_i^t \equiv A q_i^t$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \gamma \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} u \\ y \\ 1 \end{bmatrix}$$

Third equation would provide

$$1 = \gamma(0 \cdot u + 0 \cdot v + 1) = \gamma$$

Thus,

$$x = a \cdot u + b \cdot v + c$$

$$y = d \cdot u + e \cdot v + f$$

Recall that the variables are a, b, c, d, e, f. For every pair of points {p, q} the two equation can be described in the following matrix form:

$$\begin{bmatrix} u & v & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & u & v & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Let us define the equation as:

$$A \cdot x = b$$

Where A is a $6 \times 2N$ matrix consisting of point coordinates, and h is a vector made from matrix A values.

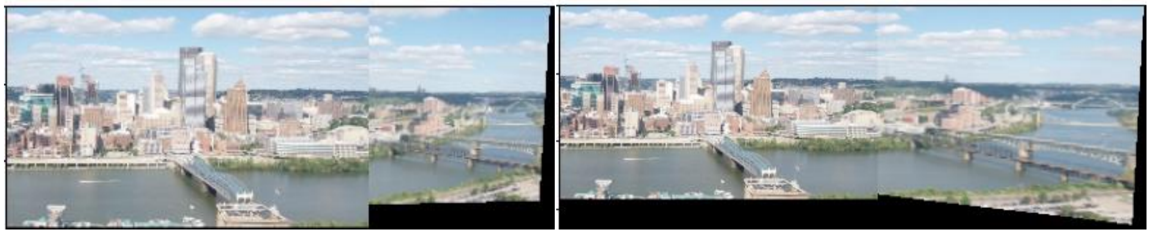
Observing the equation, every pair of points sets 2 constrains. With 6 DOF of matrix A, there should be at least 3 linearly independent pairs of points to have a unique solution.

This is a Linear Least Squares problem, since a solution should suffice $\|A \cdot x - b\|^2$ as close to 0 as possible. This minimization problem has a closed form solution, which is calculated using the Pseudo-inverse of A: $x = A^{+}b = (A^{T}A)^{-1}A^{T}b$

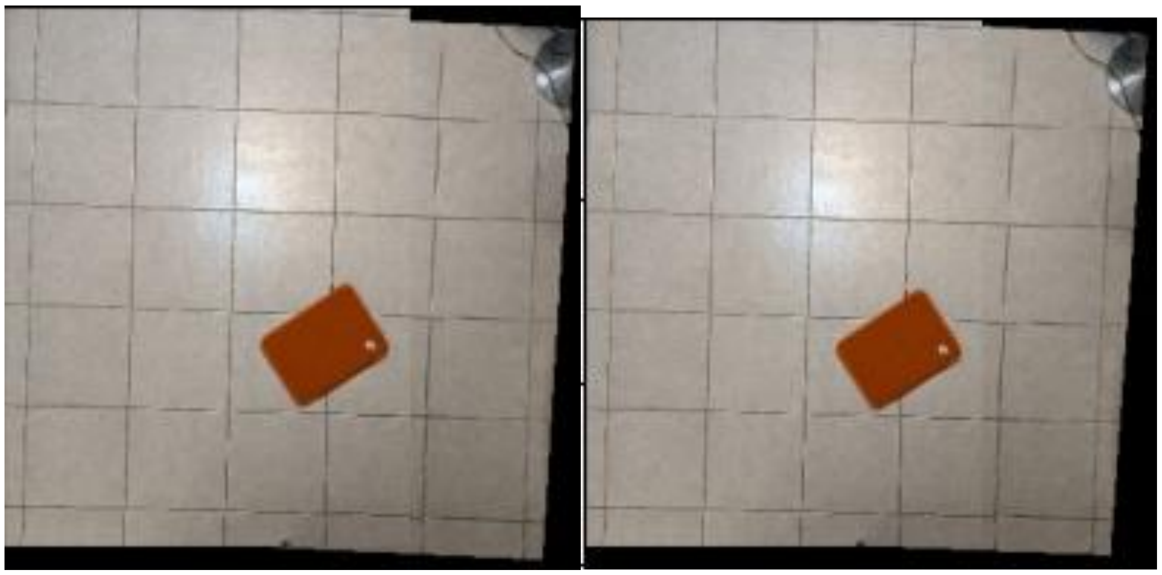
Pretty similarly to computeH's implementation, our solution follows the above description step-by-step inside the *computeA* function (see code).

Affine transformations can only transform planes into planes within the same depth in the scene (Z stays the same!). Their most strict limitation is the fact that affine transformations always preserve line parallelism. Hence it can be used to stitch two images, only if the objects in the image lie in the same depth, such as walls or very distant backgrounds.

Example for **unsuccessful stitch**, where the objects' depths difference is too big, is the incline-*.jpg pair. See comparison between affine transform (left) and general homography (right).



In the contrary, an example for **successful stitch** is where the scene is depth-wise flat, such as floor tiles in an image taken from a distance. See FloorEqualDepth-*.jpg images in my_data folder. A comparison between affine transform (left) and general homography (right) is displayed below.



Part 3 – Augmented Reality

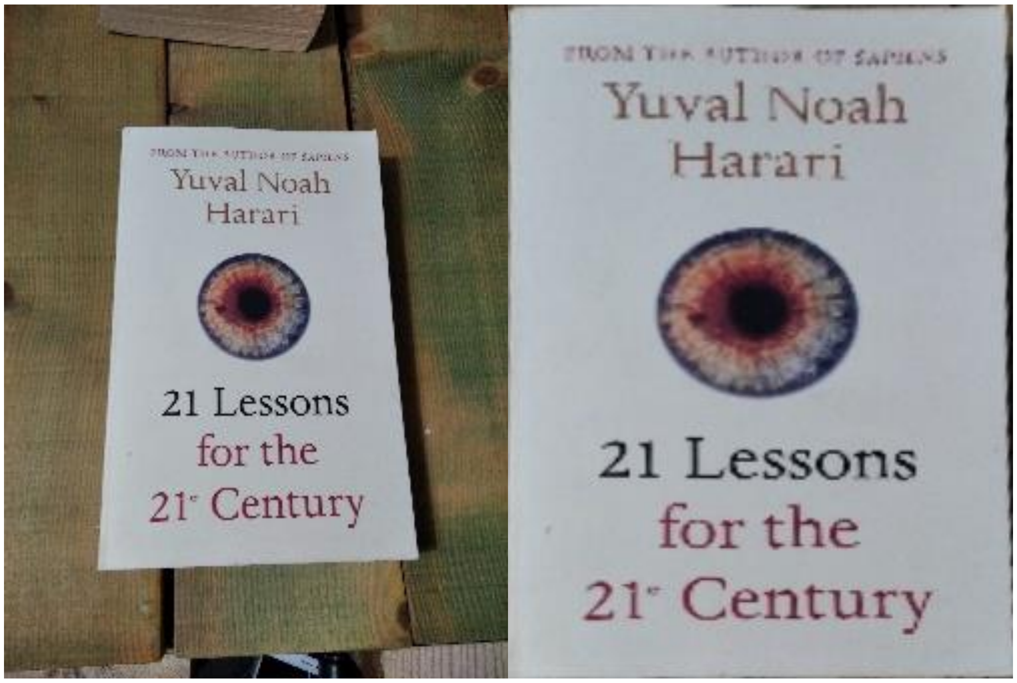
Q3.1

Reference model can be very helpful for tracking an object in video. We can get it out of every image of a book, as we saw previously although 4 points theoretically are enough, we take 6 points to be robust against noise. In the creation of the reference we will use a manual method.

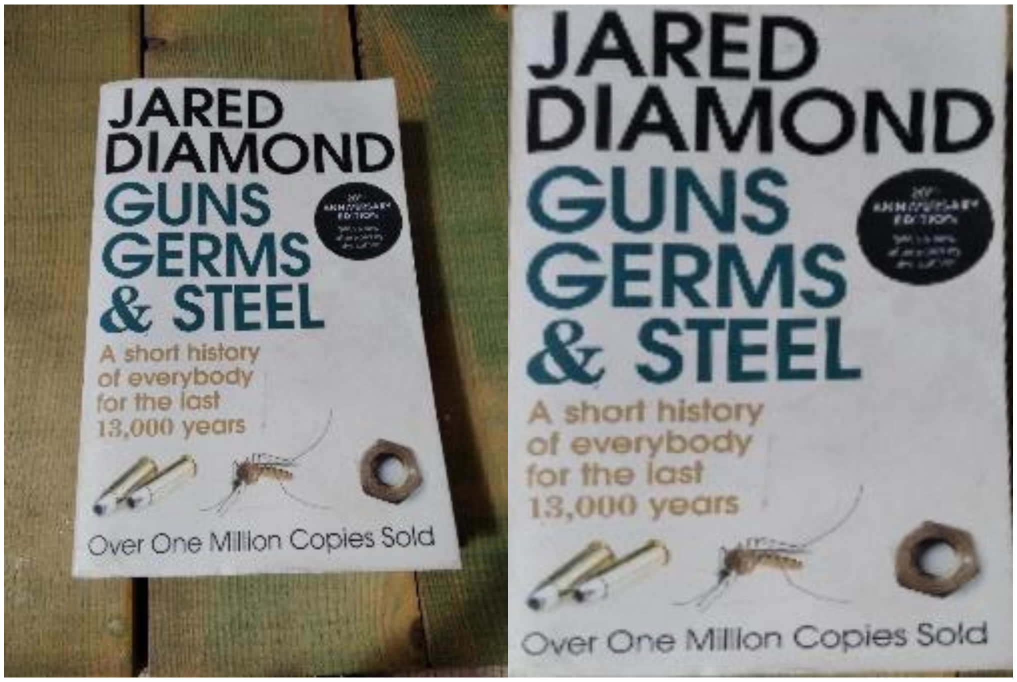
Again, order is important, we had to pick the points in the following order:

- Top left corner
- Top right corner
- Bottom left corner
- Bottom right corner
- Middle of top edge
- Middle of right edge

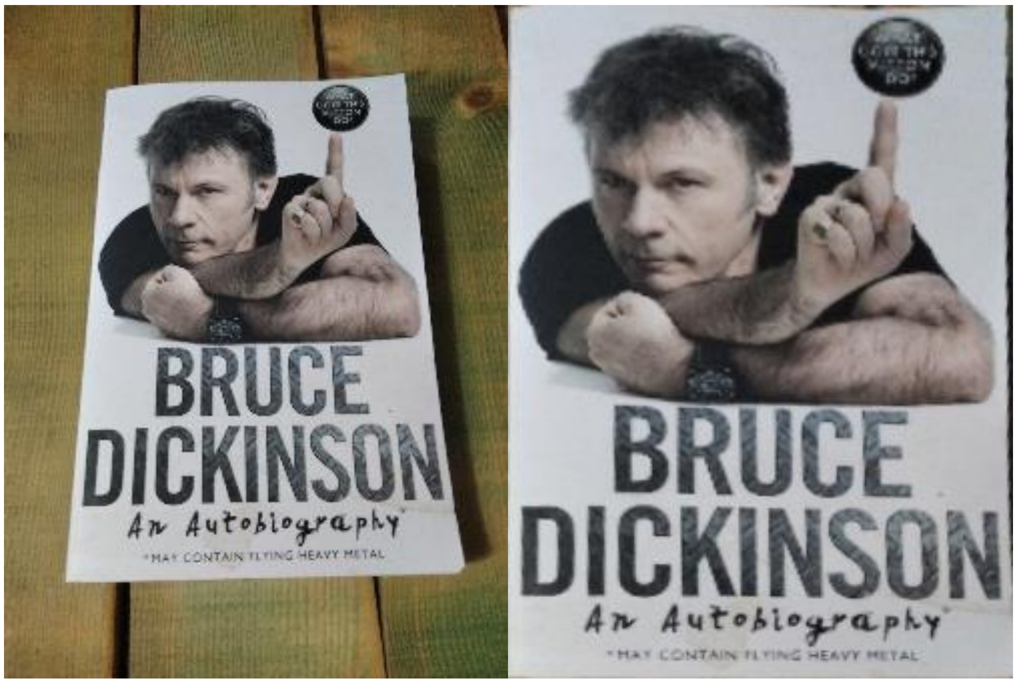
Examples:


FROM THE AUTHOR OF SAPIENS
Yuval Noah Harari
21 Lessons for the 21st Century

Book 1


JARED DIAMOND
GUNS GERMS & STEEL
A short history of everybody for the last 13,000 years
Over One Million Copies Sold

Book 2


BRUCE
DICKINSON
An Autobiography
*MAY CONTAIN FLYING HEAVY METAL
BRUCE
DICKINSON
An Autobiography
*MAY CONTAIN FLYING HEAVY METAL

Book 3

Q3.2

Now that we have a reference for a few books we can find them in a real image. We will implement an im2im() function

```
im2im(base, ref, item) -> img
```

It gets 3 images as an input

Base – an image with the reference somewhere in it (object), any size (H, W)

Ref – an image of the object we are looking for, any size

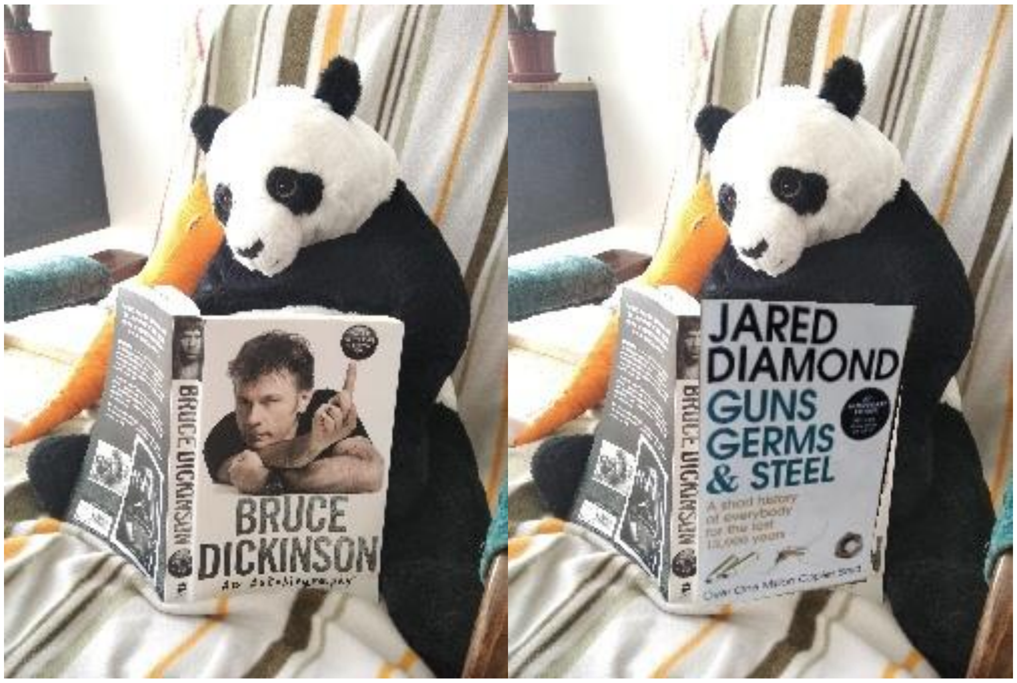
Item – an image that we will plant in the base image, any size

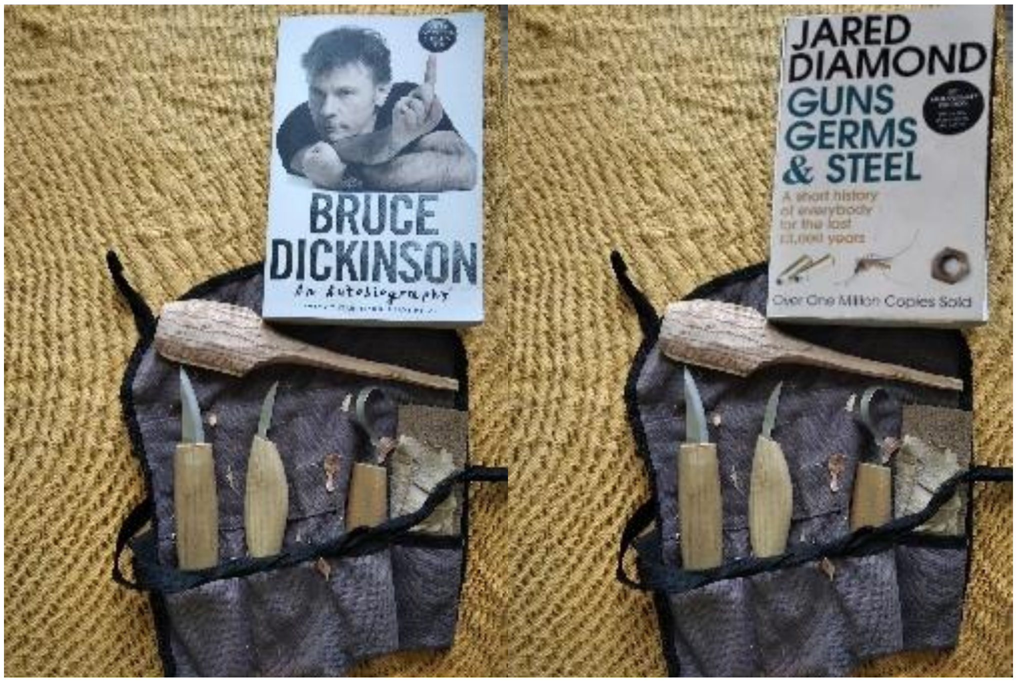
It outputs an image

Img – base image with the reference within it replaced with the item, size (H, W)

First the function will run a SIFT algorithm with base and ref, if the ref isn't occluded in the image and close enough SIFT can easily find and set a matrix to transpose it on it. Once the best matrix is calculated with ransecH (Q2.8) we resize the item to fit ref. Resize in necessary to have the item to cover the object perfectly. Perform wrap on the item with matrix H we found for ref. The new wrapped item is the same size as base image and in the proper place, with a big black background. Lastly, we mask the wrapped item and plant it on the image.

Examples:


BRUCE DICKINSON
An Autobiography
JARED DIAMOND
GUNS GERMS & STEEL
A short history of everybody for the last 13,000 years
Over One Million Copies Sold


BRUCE DICKINSON
An Autobiography
JARED DIAMOND
GUNS GERMS & STEEL
A short history of everybody for the last 13,000 years
Over One Million Copies Sold

Q3.3

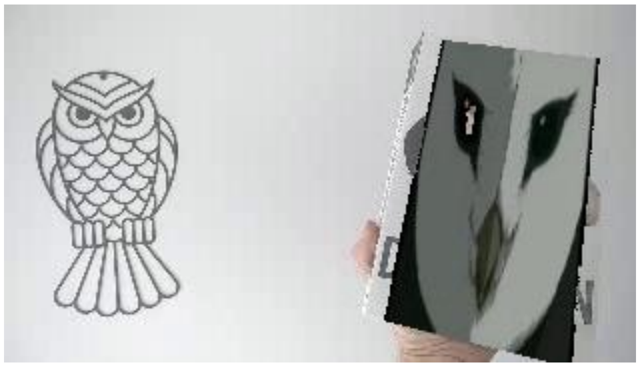
```
vid2vid(main, ref, side) -> out
```

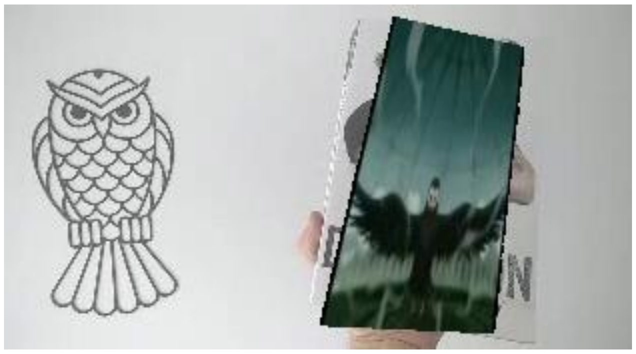
main – list of images to main video, length any size (N, H, W)

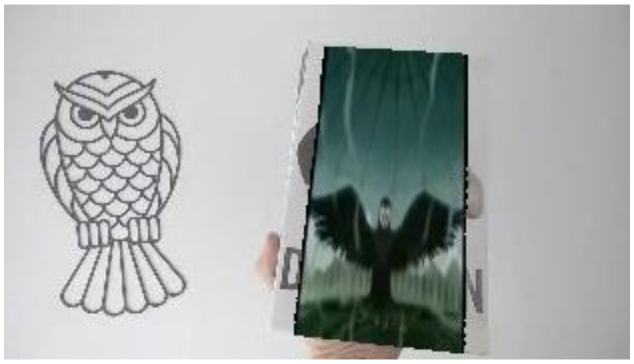
ref – reference image

side – list of images to be planted in main video, any size (M, a, b) out – list of planted images, size(min{N, M}, H, W)

Image planting can be done to any image basically, it doesn't has be another reference image, and it can be done again and again even if the object moves. Using this principle, we can implant a video within a video. It is surprisingly easy, take 2 videos Iterate over both of them simultaneously, and on each pair take the side and plant it in base with the ref and im2im(). There is no reason to change the implementation within im2im() for this function, since each pair are just 2 images. im2im() can get any size of image as input and it resizes automatically so when the perspective on the object changes im2im() can handle it.







Q3. 4

Same as vid2vid() takes a video into separate images and does im2im() on them, we can take 2 videos that are results of vid2vid() and stitch them together image by image.

This is what we have done, and to make it a little spooky we will implant the right video on the left one and wise versa.