Dated: 5th October, 2023 **Homework # 2**
Submitted by: Shahram Najam Syed (snsyed)

Q1.1:

Ans: For the given question, we are asked to prove the existence of a homography matrix "H" such that it can transform point "x1" from camera 1 to "x2" from camera 2 given the projection matrices P1 and P2 respectively for both the cameras. The assumption over here is that the points lie on a plane Π and the plane does not lie at infinity. Mathematically, we are asked to prove that for the following there is a valid "H":

$$x_1 \equiv H\, x_2$$

The projection matrices P1 and P2 can be written in the form:

$$P_1 = K_1\, [R_1 \mid t_1]$$

$$P_2 = K_2\, [R_2 \mid t_2]$$

where K_i is the intrinsic matrix, R_i is the rotation matrix, and t_i is the translation matrix, ∀ i=1,2.

And a 3D point X on plane Π is represented in homogenous coordinates as:

$$X = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

The projection of point "X" from the real world onto the image planes of Camera 1 & Camera 2 respectively can be defined as:

$$x_1 = P_1\, X$$

$$x_2 = P_2\, X$$

Intuitively, the homography matrix "H" can be understood as a compact representation of the following transformation:

a) Unproject the 2D point from the image plane of camera 1 back to the plane Π in 3D.
b) Transform the point on the plane Π from the coordinate frame of camera 1 to camera 2.
c) Project the 3D point from plane Π onto the image plane of camera 2.

Although there is no explicit 2D to 3D unprojection, rather the homography matrix H provides a direct 2D to 2D mapping of points from one image to another.

Hence for the assumption of a planar scene with all points of interest lying on a 3D plane, homography matrix "H" does indeed exists and provides a geometric and photometric relationship between the two image planes of camera 1 and camera 2.

Q1.2:

1.

Ans: Since we are given that:

$$x_1^i = H\, x_2^i$$

Such that in the vectorized form:

$$h = \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix}$$

Since vector h has 9 elements, and the homography matrix is defined up to a scale factor, hence the **<u>DOF for h is 8</u>**.

2.

Ans: For each set of points, we can get 2 independent linear equations i.e. one for the abscissa (x-coordinate) and the other for the ordinate (y-coordinate), and there are 8 unknowns/ DOF in h, hence we require a minimum of 4 point pairs to solve for h.

3.

Ans: Since, we are given the homography equation as:

$$x_1^i = H\, x_2^i$$

Expanding this in matrix form:

$$\begin{bmatrix} x_{i1} \\ y_{i1} \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_{i2} \\ y_{i2} \\ 1 \end{bmatrix}$$

Expressing these in algebraic form:

$$x_{i1} = h_{11}x_{i2} + h_{12}\, y_{i2} + h_{13} ------ (a)$$

$$y_{i1} = h_{21}x_{i2} + h_{22}\, y_{i2} + h_{23} ------ (b)$$

$$1 = h_{31}x_{i2} + h_{32}\, y_{i2} + h_{33} ------ (c)$$

Now, multiplying eq (c) by eq(a) and eq(b) respectively:

$$x_{i1}(h_{31}x_{i2} + h_{32}\, y_{i2} + h_{33}) = h_{11}x_{i2} + h_{12}\, y_{i2} + h_{13}$$

$$y_{i1}(h_{31}x_{i2} + h_{32}\, y_{i2} + h_{33}) = h_{21}x_{i2} + h_{22}\, y_{i2} + h_{23}$$

Simplifying:

$$x_{i1}h_{31}x_{i2} + x_{i1}\, h_{32}\, y_{i2} + x_{i1}h_{33} - h_{11}x_{i2} - h_{12}\, y_{i2} - h_{13} = 0$$

$$y_{i1}\, h_{31}x_{i2} + y_{i1}h_{32}\, y_{i2} + y_{i1}h_{33} - h_{21}x_{i2} - h_{22}\, y_{i2} - h_{23} = 0$$

Converting to matrix format to achieve the format $A_i h = 0$:

$$\begin{bmatrix} -x_{i2} & -y_{i2} & -1 & 0 & 0 & 0 & x_{i1}x_{i2} & x_{i1}y_{i2} & x_{i1} \\ 0 & 0 & 0 & -x_{i2} & -y_{i2} & -1 & x_{i2}y_{i1} & y_{i1}y_{i2} & y_{i1} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = 0$$

Where,

$$A_i = \begin{bmatrix} -x_{i2} & -y_{i2} & -1 & 0 & 0 & 0 & x_{i1}x_{i2} & x_{i1}y_{i2} & x_{i1} \\ 0 & 0 & 0 & -x_{i2} & -y_{i2} & -1 & x_{i2}y_{i1} & y_{i1}y_{i2} & y_{i1} \end{bmatrix}$$

4.

Ans: A trivial solution in terms of linear algebra is the uninformative solution set provided by a set of linear combination where all variables are set to zero. In the context of the equation:

$$A_i h = 0$$

The trivial solution for h would be a zero vector (9x1). Although being a mathematically valid solution, it does not represent a valid homography.

No, the matrix A is not full rank (i.e. rank deficient) if h has a non-trivial solution. If matrix A does have a full rank, the only solution for $A_i h = 0$ would be a trivial solution as mentioned above.

Due to matrix A not being a full rank and having a non-trivial solution means that the $A^T A$ will have at least one eigenvalue that is zero (since h is in the null space of A). And thus, the eigenvector corresponding to the zero eigenvalue is the one that is of use to us, which gives us the homography matrix H.

In practice this is usually achieved by using SVD to decompose matrix A as U and V to solve for the non-trivial solution of h, where h is the column of V corresponding to the singular value of zero.

Q1.3:

Ans: Since we are given that between the two cameras there is only pure rotation, and that:

$$x_1 = K_1\,[I \mid 0]\,X - - - - - - (a)$$

$$x_2 = K_2\,[R \mid 0]\,X - - - - - - (b)$$

Where,

- x1 and x2 are the points of the camera planes 1 & 2 respectively
- K_1&K_2 are the intrinsic matrices of camera 1 & 2 respectively
- I is the 3x3 identity matrix
- 0 is the 3x1 zero vector
- R is the 3x3 rotation matrix of the camera 2

From equation b, we can infer:

$$X = [R \mid 0]^{-1} {K_2}^{-1}\, x_2$$

Substituting the equation of X in equation (a):

$$x_1 = K_1\,[I \mid 0]\,[R \mid 0]^{-1} {K_2}^{-1}\, x_2$$

Which can be simplified to:

$$x_1 = K_1\, R^{-1} {K_2}^{-1}\, x_2$$

Comparing the above with:

$$x_1 \equiv H\, x_2$$

We get the homography matrix H as:

$$H = K_1\, R^{-1} {K_2}^{-1}$$

Hence proven that for two camera separated by pure rotation there exists a valid homography matrix H determining 2D-to-2D correspondence between the two image planes of the two cameras.

Q1.4:

Ans: In the previous question, we derived that:

$$H = K_1 \, R^{-1} {K_2}^{-1}$$

For a rotation of θ, we can denote this as:

$$H = K_1 \, {R_\theta}^{-1} {K_2}^{-1}$$

Given since K is constant,

$$H = K \, {R_\theta}^{-1} K^{-1}$$

$$H^2 = K \, {R_\theta}^{-1} K^{-1} \, K \, {R_\theta}^{-1} K^{-1}$$

$$H^2 = K \, {R_\theta}^{-1} \, {R_\theta}^{-1} K^{-1}$$

$K^{-1} \, K$ becomes an identity matrix.

Assuming rotation along the x-axis, we get:

$$R_\theta = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$${R_\theta}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}$$

$$({R_\theta}^{-1})^2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos^2\theta - \ sin^2\theta & 2 \ \sin\theta \ \cos\theta \\ 0 & -2 \ \sin\theta \ \cos\theta & cos^2\theta - \ sin^2\theta \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos 2\theta & \sin 2\theta \\ 0 & -\sin 2\theta & \cos 2\theta \end{bmatrix}$$

Which proves that for 2θ rotation we get a homography of H^2.

Q1.5:

Ans: For planar homography it is assumed that all points lie on a single plane in the 3D world, but this fails to account for depth variation or non-planar structure. Thus this becomes insufficient to account for the mapping between various viewpoints for depth induced distortions.

Q1.6:

Ans: Since we are given that for a 3D point in the real world, X = [X1, X2, X2, 1] (in the homogenous coordinates), the 2D projection is represented by x = PX, where P is the projection matrix and x is the homogenous coordinate of the projected point.

Now consider a line in the 3D space. For that consider two points, say, A and B, having homogenous coordinates of XA and XB respectively. The bounded locus (controlled by a scalar Ψ) between the homogenous coordinates XA and XB is algebraically defined as:

$$X = \Psi X_A + (1 - \Psi)\, X_B$$

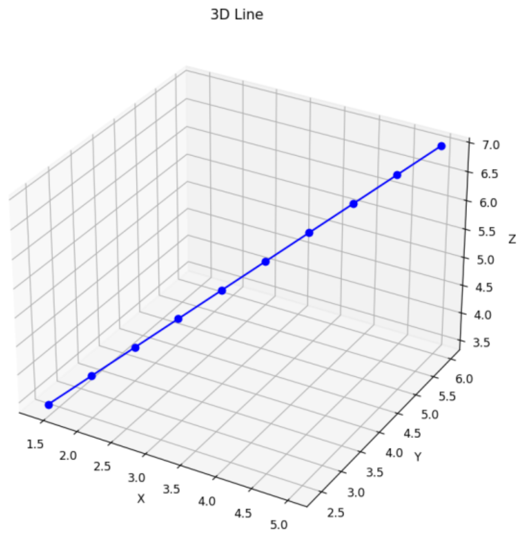
Now, projecting the 3D-line to 2D space, we apply perspective projection by multiplying projection matrix "P" on both sides:

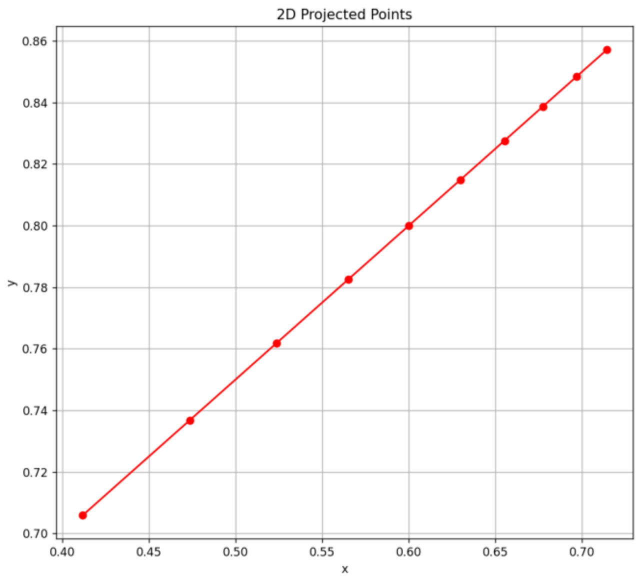
$$PX = P[\Psi\, X_A + (1 - \Psi)\, X_B]$$

$$x = \Psi\, x_A + (1 - \Psi)\, x_B$$

The equation above validates the fact that 2D locus of x varies linearly with Ψ, similar to the 3D point X. Hence, the entire line in 3D gets mapped to a line segment in 2D.

As an experiment, to verify this notion (on preservation of lines) mathematically, I randomly selected two 3D points, parametrically combined the two points and sampled multiple points along the bounded locus providing multiple 3D points lying on the same 3D line. I projected these 3D points into 2D space by taking the dot product of the 3D points with the matrix P (arbitrarily selected) to get the mapped 2D points. I plotted the original 3D line and the projected 2D points to verify that the line is truly preserved, and following are the results.





Q2.1.1:

Ans: Although both FAST and Harris corner detector operate by identifying points in the image with gradient change along various directions (definition of a corner), but in principle operate completely differently yielding varying computational performance, where a Harris corner detector uses geometric interpretable reasoning for corner detection while FAST makes use of intensity based testing.

In terms of Harris corner detector, it uses a small sliding window to compute a Harris matrix which is a second moment matrix to capture gradient information around each pixel in all directions. On the contrary, the FAST corner detector considers a circle of sixteen pixels around the pixel interest and classify a pixel as a corner if for a set of continuous pixels are either all darker or brighter than the center pixel.

Computationally, as mentioned above Harris corner detector operates using a sliding window for computing the gradients followed by an eigenvalue analysis for each pixel rendering it computationally expensive at a trade-off for robustness to noise. On the contrary, FAST as the name suggests is computationally fast since it uses a decision tree to filter out non-corners without explicitly testing all set of sixteen pixels centered around the pixel in question.

Q2.1.2:

Ans: A feature descriptor using BRIEF is constructed by generating binary strings using a binary test between pixels. On the other hand, the filter banks discussed in class (Gaussian, LoG, directional Gaussian, etc.) emphasize a specific type of feature within the image via convolution. Technically, filter banks can be used as feature descriptor of a particular region or structure, but since the responses of filter banks are higher dimensional it would be more compute extensive than using BRIEF, and would require encoding them as lower dimensional embeddings to be used effectively (which again would be compute extensive).

Q2.1.3:

Ans: Hamming distance between two strings (of equal lengths) quantifies the number of mismatches between the two strings corresponding to their respective positions. In the context of BRIEF which uses a binary string as descriptor, the Hamming distance is simply the XOR operation corresponding to the respective bits of the two binary strings. When two interest points are represented by binary strings, then a lower Hamming distance suggests that the interest points are potentially a match. Mathematically the Hamming distance is represented by:

$$H(a,b) = \sum_{i=0}^{n-1} (a_i \oplus b_i)$$

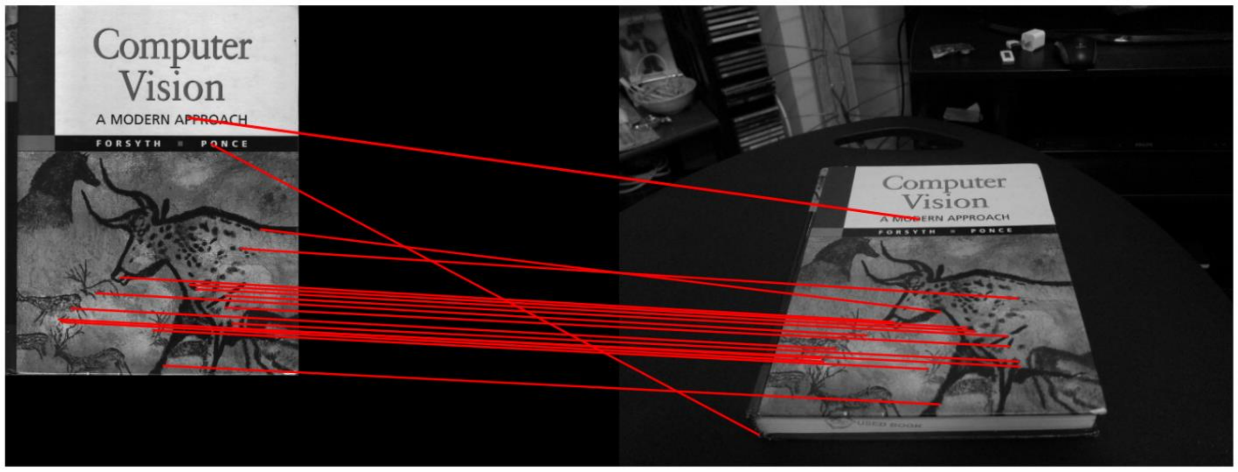
Nearest Neighbor Search is the process of finding the closest match for a query point within a set of data points. Once we have the binary representation of a query point, a distance metric, which in our case is a Hamming distance metric, is used to determine the distance of the query point to the binary representation of the binary descriptors of the reference image, and the binary descriptor with the minimum distance is selected as the potential closest match. To speed up the search I came across multiple techniques like Locality Sensitive Hashing or KD-Tree to optimize the search process.

The advantage of using Hamming distance over Euclidean distance as a similarity metric for BRIEF descriptors relies heavily on the computational efficiency:

1. Hamming distance simply uses bitwise XOR for the binary strings making it more efficient than mathematically involved Euclidean distances which involves float computation.
2. The binary nature of the binary strings makes it computationally efficient to use a binary function like in the case of Hamming Distance.
3. Quantification and storage of non-binary Euclidean distances is less intuitive as compared to the robustness of bit-wise matching for Hamming distance.
4. Computation and memory efficiency makes the usage of Hamming distance for BRIEF descriptor similarity suitable for real-time deployments or systems where scalability is paramount.

Q2.1.4:

Ans: Following is the result of matches of FAST corner features with the BRIEF descriptors for a pair of images "cv_cover.jpg" and "cv_desk.jpg" for default values of sigma = 0.15 and ratio = 0.7.

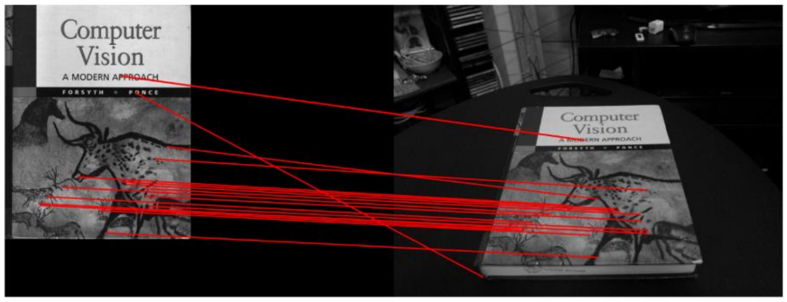
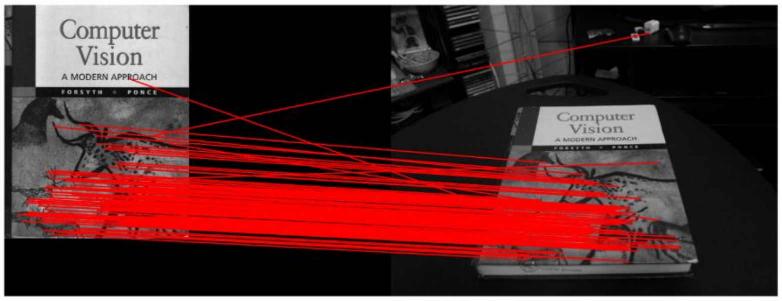
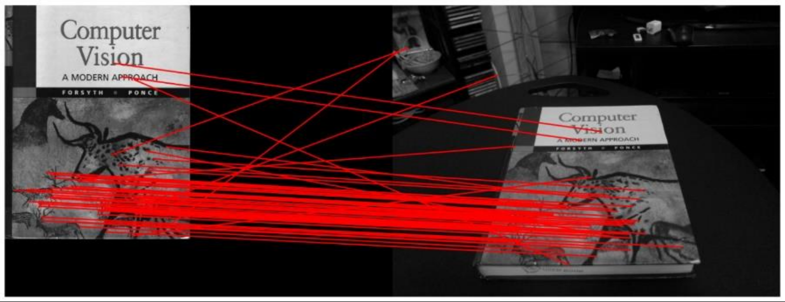


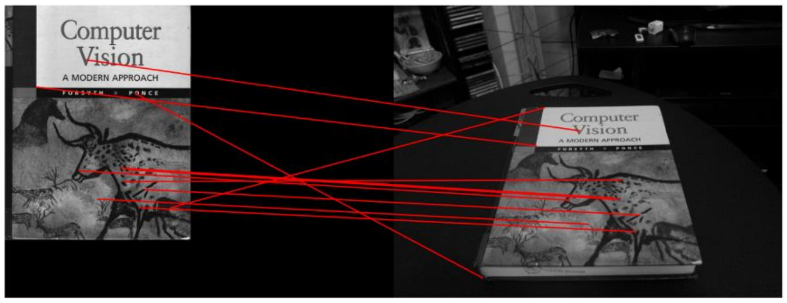
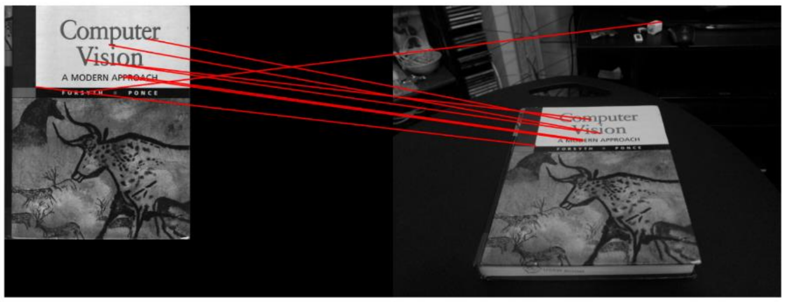
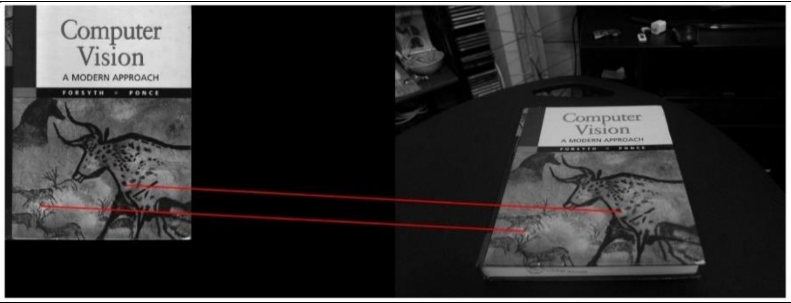
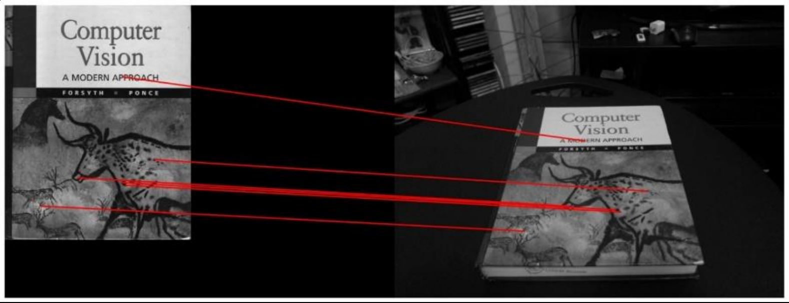
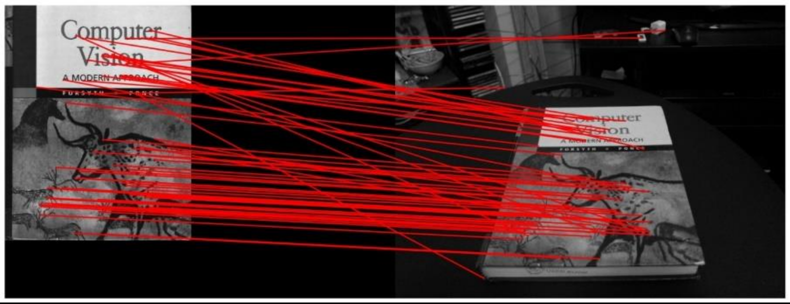
Q2.1.5:

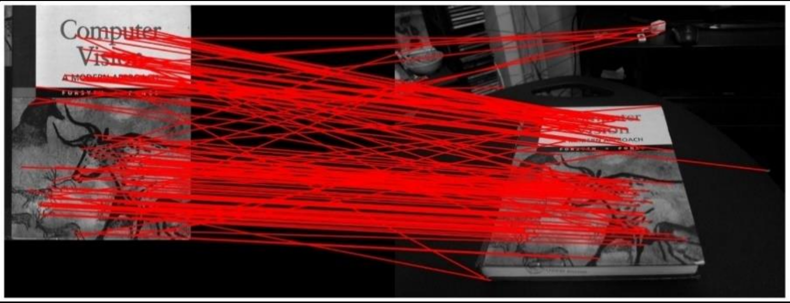
Ans: Varying the two tunable parameters, i.e. sigma (FAST) and the ratio (BRIEF), I observed the following results:

a) Sigma had an inverse relationship with the number of corners being detected. i.e. a higher sigma translated to only the most pronounced and prominent corners with the highest contrast difference being considered while filtering out the noisy or weak corners. Conversely, a lower sigma translated to including less prominent or weaker contrast differences, thus giving noisy or weak corners.
b) As for the ratio, a direct relationship was observed with the number of corner matches. i.e. for a higher value of the ratio, leniency is observed for the number of matches filtered thus potentially providing false matches. Consequently, a lower ratio provided more robust or matches but at a trade-off of potentially filtering out genuine matches.

An ablation study was conducted, and following are the tabulated results:

Sigma	Ratio	Result
0.15	0.7	
0.05	0.7	
0.1	0.7	

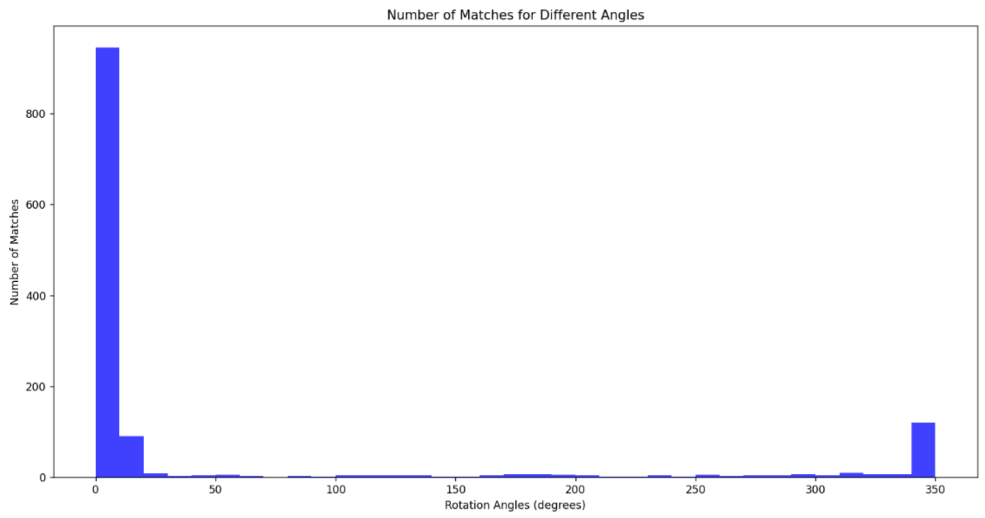
0.2	0.7	
0.3	0.7	
0.15	0.5	
0.15	0.6	
0.15	0.8	

0.15	0.9	

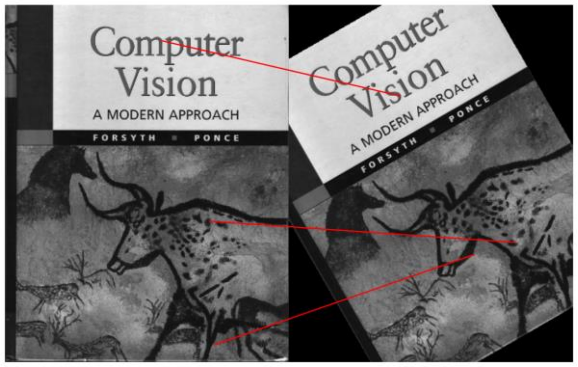
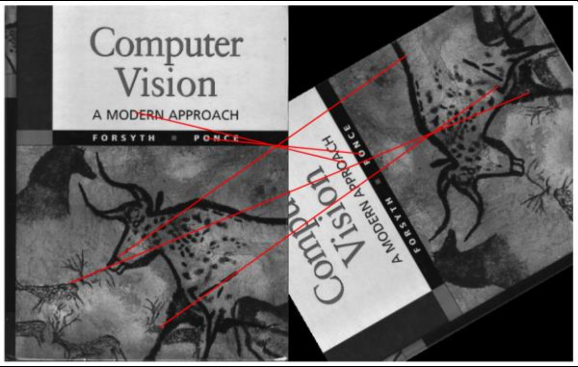
Q2.1.6:

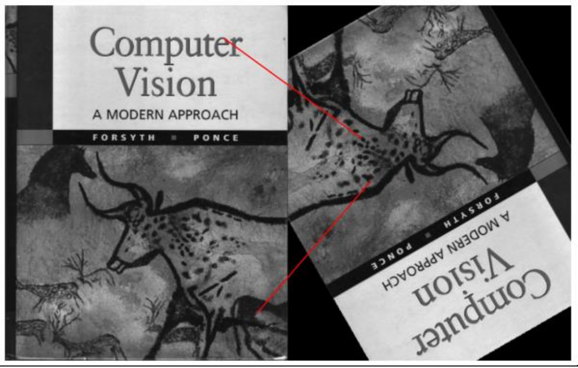
Ans: From the results below, it can be observed that rotation of the images yields fewer matches as you rotate the image, the reason behind this is that BRIEF descriptor is not rotation invariant. The reason behind this is that BRIEF compares the features in relative positions within the pair of images, but when the image is rotated, the feature's relative position changes thus causing degradation in the BRIEF performance.

Following is the histogram of the number of matches at different rotation angles:



And the following table tabularizes the matching result at three different orientations:

Rotation Angle	Matching result
30 degrees	
120 degrees	

210 degrees	

Q2.2.1:

Ans: Code submitted as a python file in the code submission.

Q2.2.2:

Ans: Code submitted as a python file in the code submission.

Q2.2.3:

Ans: Code submitted as a python file in the code submission.

Q2.2.4:

Ans: After implementing the HarryPotterize.py script and the compositeH() function, and executing the code with default hyperparameters of 500 max_iters and 2.0 inlier_tol, I got the following result. As expected, the image did indeed overlay in the correct position but didn't fill up the same space as the book. The reason behind this is that the image dimensions for hp_cover.jpg image is not the same shape as the cv_cover.jpg, for which the homography was computed for. Hence resizing it to the correct shape provides the correction.

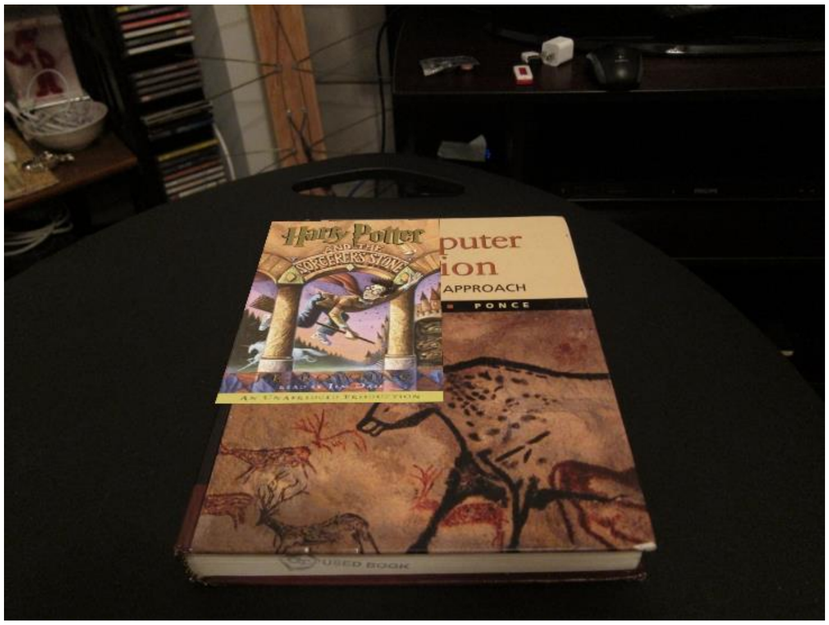


Figure: Warping without resizing

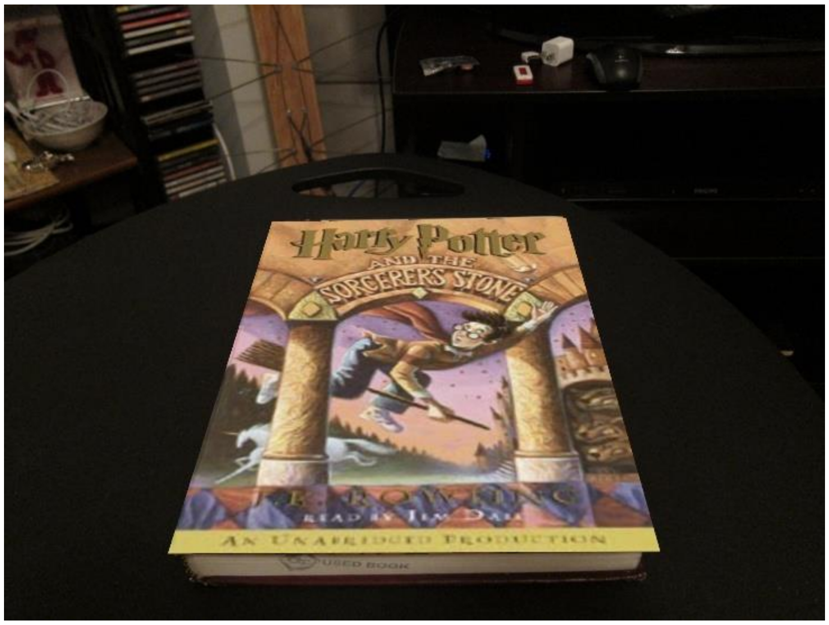
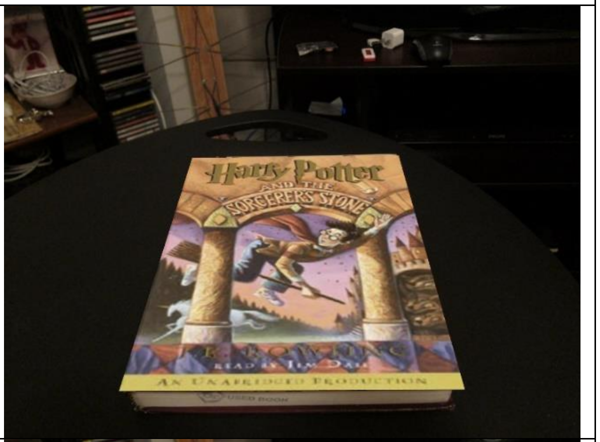
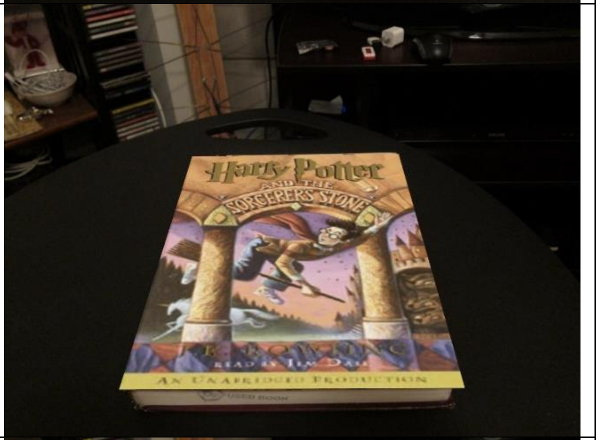
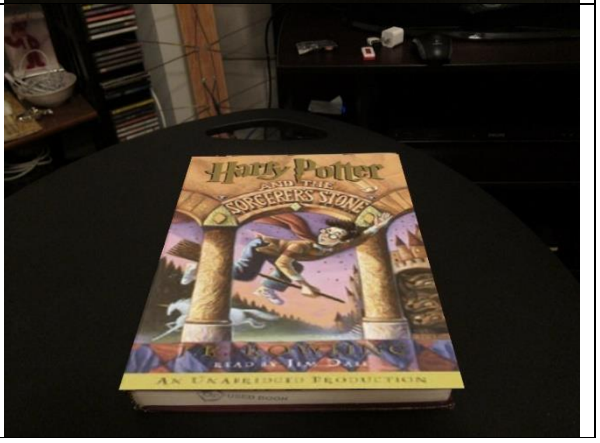
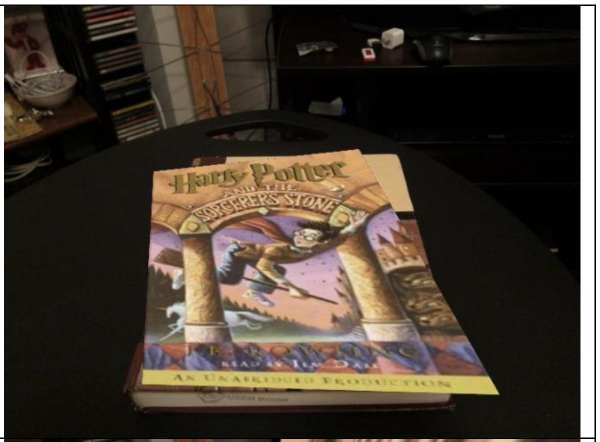
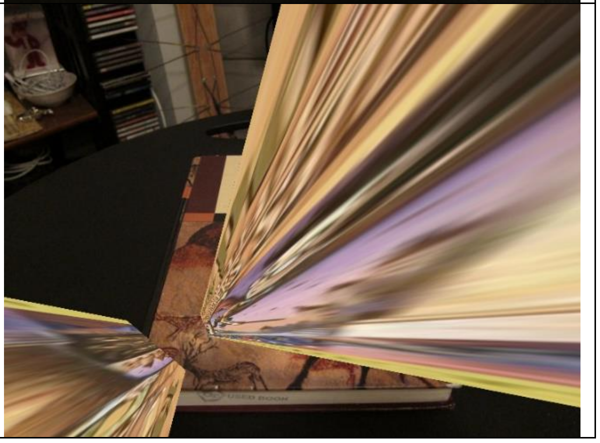


Figure: Warping with resizing

Q2.2.5:

Ans: Intuitively, increasing the number of iterations would yield an optimal homography with a trade-off of slower computation (and vice versa) but since in my case the default parameters gave near perfect homography and thus warping, hence no significant changes were observed by increasing the number of iterations. But on the other hand, increasing the inlier tolerance resulted in inclusion of outliers further away from the estimated transformation. Following is the table that summarizes the results:

max-iters	**Inlier-tol**	**Result**
500	2.0	
1000	2.0	
1500	2.0	

500	10.0	
500	50.0	

Q3.1:

Ans: Generated video of the Kungfu Panda video warped over the cv-cover in the book video is saved as ar.avi (as instructed) and submitted to be reviewed in the results folder. Following are the screenshots at 3 different time instances:

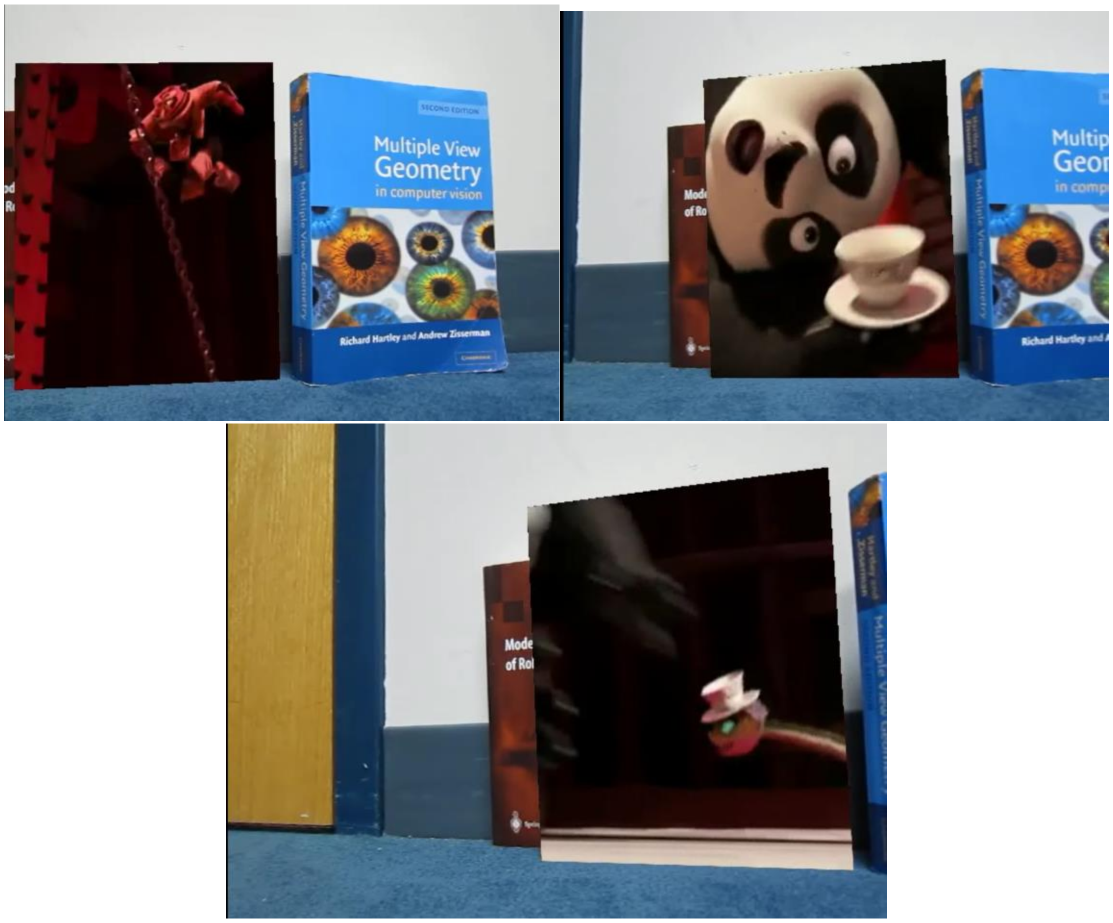


Figure: Top left (overlay to the left of the frame), Top right (overlay to the middle of the frame), Bottom center (overlay to the right of the frame)

Q4.1x:

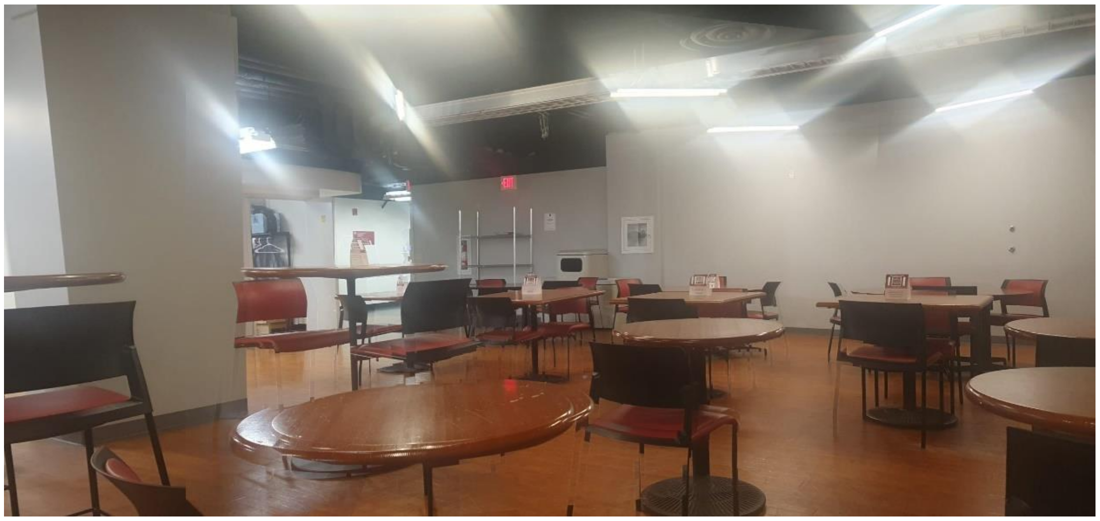
Ans: For this question, I am getting a runtime global FPS (total number of frames/total runtime) ~100 FPS - 150 FPS (depending on whether any other applications are running in my background, particularly videos on YouTube).

For making the AR code run in real time, I first implemented the complete code using OpenCV functions leveraging the highly optimized code and functions already available. The following are the changes I made to the approach that we were using in Q 3.1:

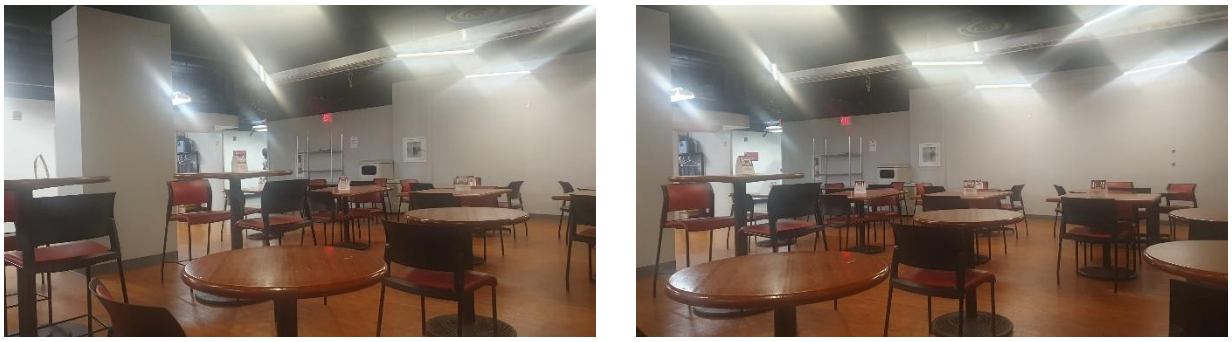
a) Changed the BRIEF descriptor with ORB descriptor
b) Vectorized the operations to optimize the runtime complexity
c) To enhance processing speed, I employed a frame subsampling technique where keypoints and descriptors are computed once every 10 frames and using the same results for the intermediate frames. This is a cheap heuristic and works well for the provided videos since the motion in the video is relatively smooth and gradual rather than abrupt.

Q4.2x:

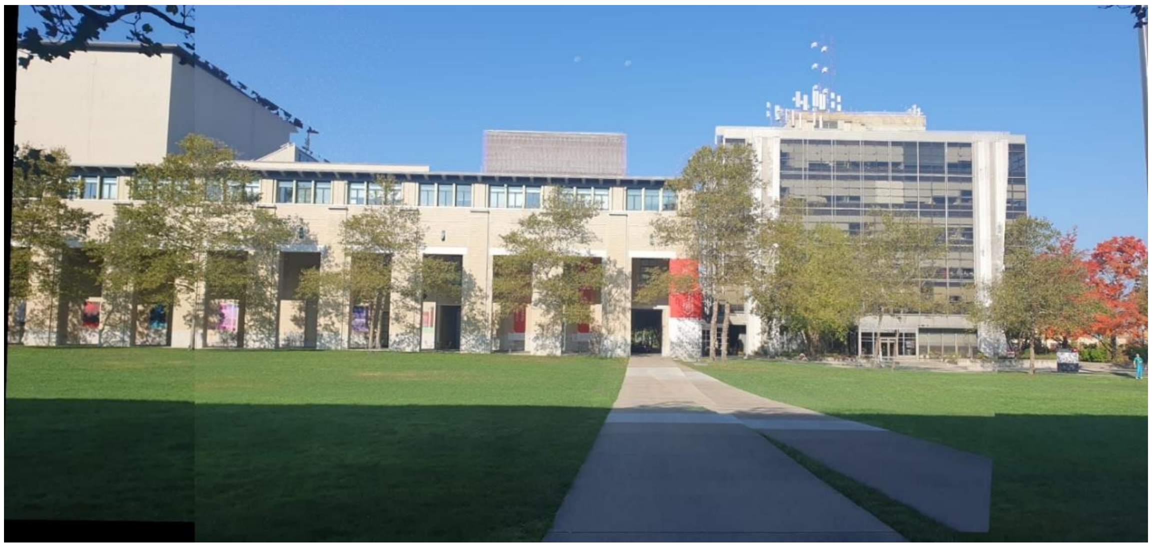
Ans: Following is the result of the panorama for pure rotation:



For the following set of images:



And following is the result of the panorama for rotation + translation (out of curiosity):



For the following set of images

References:

[1] Hartley, Richard, and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[2] Szeliski, Richard. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[3] Strang, Gilbert. *Introduction to linear algebra*. Wellesley-Cambridge Press, 2022.

[4] https://docs.opencv.org/4.x/d9/dab/tutorial_homography.html

[5] https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga4abc2ece9fab9398f2e560d53c8c9780

[6] https://docs.opencv.org/3.4/d1/de0/tutorial_py_feature_homography.html

[7] https://stackoverflow.com/questions/4655334/ransac-algorithm

[8] https://cs.stackexchange.com/questions/70434/how-is-ransac-used-to-estimate-a-homography-given-the-descriptors-of-both-images

[9] https://stackoverflow.com/questions/21827594/raise-linalgerrorsvd-did-not-converge-linalgerror-svd-did-not-converge-in-m

[10] https://www.youtube.com/watch?v=l_qjO4cM74o

[11] https://www.youtube.com/watch?v=lU4zgDe1x6Y