Question A.1

For this Question, I'm following the notation of Prince computer vision book:

Let's assume there exist a transformation T that places the world coordinate system on the first camera. So that the extrinsic parameters of the first camera becomes $[I, 0]$,

$$\Pi_1 \xrightarrow{T} [I, 0]$$

where Π_1 is the extrinsic parameters of camera 1. Therefor the second camera has the general form of $[\Omega, \tau]$ as its extrisic matrix:

$$\Pi_1 \xrightarrow{T} [I, 0]$$
$$\Pi_2 \xrightarrow{T} [\Omega, \tau]$$

where Π_2 is the extrinsic parameters of camera 2
We can write following :

$$\lambda_1 \tilde{x}_1 = \Lambda_1 [I, 0] \tilde{w} \quad (1)$$
$$\lambda_2 \tilde{x}_2 = \Lambda_2 [\Omega, \tau] \tilde{w} \quad (2)$$

where $\tilde{w}$ is a 3D homogeneous coordinate of a scene point in a coordinate system that is centered on camera 1. $\tilde{x}_1$ & $\tilde{x}_2$ are the observed positions in camera 1 & 2, expressed in $^{2D}\Lambda$ homogeneous coordinates. λ_1 & λ_2 are scalers.
Λ_1 and Λ_2 are intrisic matrices of camera 1 and camera 2.
equation (1) simplifies to:

$$\lambda_1 \tilde{x}_1 = \Lambda_1 w \quad (3)$$

wher w doesn't have the fourth element of $\tilde{w}$ (it's 3x1)
similarly equation (2) gives,

$$\lambda \Lambda_2^{-1} \tilde{x}_2 = \Omega w + \tau \quad (4)$$

Subsituting ω from 3 to 4 results in:

$$\lambda_2 \Lambda_2^{-1} \tilde{x}_2 = \lambda_1 \Omega \Lambda_1^{-1} \tilde{x}_1 + \tau \quad (5)$$

Equation (5) presents a constrain on the corresponding points $\tilde{x}_1$ & $\tilde{x}_2$. We take the cross product of both sides of 5 with vector τ:

$$\lambda_2 \tau_x \Lambda_2^{-1} \tilde{x}_2 = \lambda_1 \tau \times \Omega \Lambda_1^{-1} \tilde{x}_1 \quad (6)$$

Then we take an inner product of both sides with $\Lambda_2^{-1} \tilde{x}_2$ the left hand side of 6 disappears because $\tau_x \Lambda_2^{-1} \tilde{x}_2$ is $\perp$ to $\Lambda_2^{-1} \tilde{x}_2$.

$$0 = \tilde{x}_2^T \Lambda_2^{-1} \tau_x \Omega \Lambda_1^{-1} \tilde{x}_1 \quad (7)$$

Note that λ_1 & λ_2 are dropped as LHS is zero now.
Also the cross product can be written in matrix form:

$$\tau_x = \begin{bmatrix} 0 & -\tau_z & \tau_y \\ \tau_z & 0 & -\tau_x \\ -\tau_y & \tau_x & 0 \end{bmatrix}$$

Equation (7) is indeed: $\tilde{x}_2^T \Lambda_2^{-1} E \Lambda_1^{-1} \tilde{x}_1 = 0$
where E is the essential matrix.

Thus, we define fundamental matrix as:

$$\boxed{F = \Lambda_2^{-1} E \Lambda_1^{-1}} \quad (8)$$

Note the transformation $[T^{-1}]$ is required to derive Ω & τ from extrinsic matrices. This transformation that includes a rotation & a translation can written in the form of a 4x4 matrix that transforms homogenous coordinates. Since we have only rotation & translation, T has a form of:

$$T = \begin{bmatrix} R'_{3\times3} & \tau' \\ 0\ 0\ 0 & 1 \end{bmatrix}$$ ← this is a Euclidean transformation that preserves (lengths) and (angles)

According to Equation (15.4) of Prince book.

$$\Pi_1 T = [I, 0]$$

$$\begin{bmatrix} R_1 & \tau_1 \\ {\scriptstyle 3\times3} & {\scriptstyle 3\times1} \end{bmatrix} \begin{bmatrix} R' & \tau' \\ {\scriptstyle 0_{1\times3}} & 1 \end{bmatrix} = [I, 0]$$

↳ Extrinsic matrix Π_1 has the form $[R_1 \tau_1]$. (R_1: rotation, τ_1: translation)

$$\Rightarrow R_1 R' = I \longrightarrow R' = R_1^{-1}$$

$$R_1 \tau' + \tau_1 = 0 \Rightarrow \tau' = -R_1^{-1}\tau$$

$$\Rightarrow T = \begin{bmatrix} R_1^{-1} & -R_1^{-1}\tau \\ 0_{1\times3} & 1 \end{bmatrix} \quad (9)$$

$$\Rightarrow [\Omega, \tau] = \Pi_2 T^{-1} \quad (10)$$

Continue to next page:

CSC2503 Fall 2018: Homework 2

Niloufar Afsariardchi

November 8, 2018

1 Part A-1

(continuation of previous pages)
So the algorithm to find F_0 is

1. Derive the transformation T from Equation 9
2. Derive Ω and τ from Equation 10
3. Compute F_0 from Equation 8

This is the F_0 that I obtained:

$$F_0 = \begin{bmatrix} 0 & -0.0031623 & 0 \\ -0.0031623 & 0 & -0.94868 \\ 0 & 0.94868 & 0 \end{bmatrix}$$

2 Part A-2

For this part, we need to compute the perpendicular distance between a point $p = (x, y, 1)$ and a line $l_1 = (a, b, c)$. First, I derive the line parameters for a line l_2 that is perpendicular to l_1 that passes through p. We know that the normal to the line l_1 is $n_1 = (a, b)$. We know that the normal to l_2, which we call it n_2 is also normal n_1, therefore $n_1^T n_2 = 0$. We can have another constrain on n_2 by considering $||n_2|| = 1$. Therefore solving these two equations gives $n_2 = (a_2, b_2) = (b, -a)/\sqrt{a^2 + b^2}$. Now we should compute the third line parameter of l_2, c_2. We know l_2 passes through p, therefore $a_2 x + b_2 y + c_2 = 0$, which gives $c_2 = (-bx + ay)/\sqrt{a^2 + b^2}$. Thus we have found all line parameters for l_2 as

$$l_2 \quad = \quad (b, -a, -bx + ay) \tag{1}$$

Note that I normalized everything by a factor of $\sqrt{a^2 + b^2}$. Having line parameters for both l_1 and l_2, we can now easily find where the lines intersect. I call

their intersection point q,

$$
\begin{aligned}
q &= l_1 \times l_2 && (2)\\
&= \begin{bmatrix} 0 & -c & b \\ c & 0 & -a \\ -b & a & 0 \end{bmatrix} \begin{bmatrix} b \\ -a \\ -bx + ay \end{bmatrix} && (3)\\
&= \begin{bmatrix} \frac{-ca - b^2x - aby}{a^2+b^2} \\ \frac{-cb - a^2y - abx}{a^2+b^2} \\ 1 \end{bmatrix} && (4)
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
d &= ||\vec{p} - \vec{q}|| && (5)\\
&= \left((\frac{-ca + b^2x - aby}{a^2 + b^2} - x)^2 + (\frac{-cb + a^2y - abx}{a^2 + b^2} - y)^2 \right)^{0.5} && (6)\\
&= \frac{|ax + by + c|}{\sqrt{a^2 + b^2}} && (7)\\
&= \frac{|l_1^T p|}{||n_1||} && (8)
\end{aligned}
$$

The maximum perpendicular distance to uncropped lines would occur on one of the end points of the cropped lines. Therefore I compute the perpendicular distance on the end points of the line and then take the maximum. Moreover, the line parameters could be found from fundamental matrix according to $l = x_2^T F$ or $l = x_1^T F^T$ depending the camera.

After taking the maximum of perpendicular error for each line end, I took the maximum of these error for all grid points and set it to the error associated with F matrix estimation. I varied the grid size between 10*10–100*100 and realized that the error would change for about 20%, but after this the grid size would only minimally changes the error, so I set the grid size to 100*100. Here you can see the epipolar lines in one image but derived considering both F_0 and F,

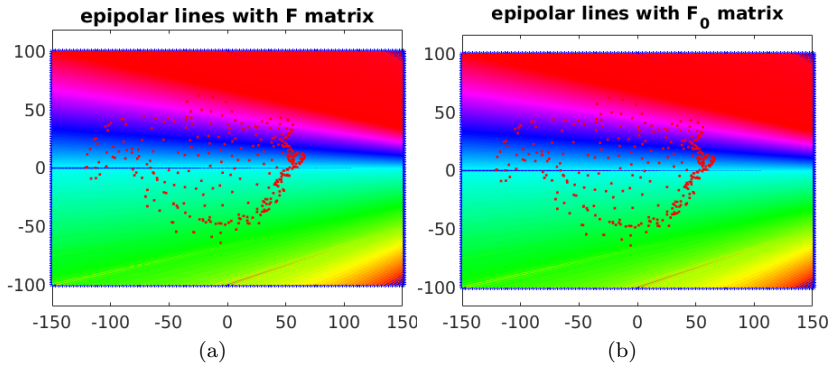


Figure 1: Epipolar lines obtained with F_0 and F for a grid with 100*100 points

Since F_0 and F are very close, we cannot differentiate the above results. The total perpendicular error that I got is $d_{\max} = 1.14 \times 10^{-12}$.

3 Part A-3

For this part, I repeated the process for each σ_n 10 times and took the median of 10 values as an estimate of perpendicular error per σ_n. Note that I used linEstF for estimating F. Increasing the number of trial for each σ_n reduces the scatter in the final $d_{\max}$ vs σ_n plot. Next, I varied σ_n using `sigma_n=logspace(-5,1.6,80)` where I changed σ_n from 10^{-5} to 40 with logarithmic spacing. I did not go above $\sigma_n = 40$, because 40 is about 40% of image size and must results in a significant error. Here you can see how $d_{\max}$ changes for different σ_n values.

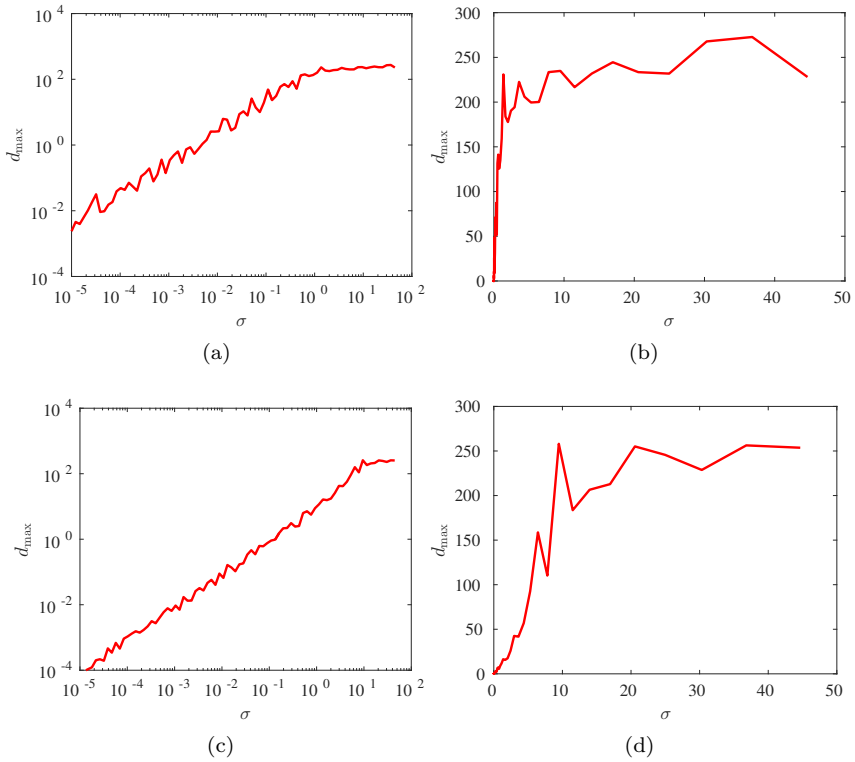


Figure 2: Maximum perpendicular error $d_{\max}$ per variance of Gaussian noise σ_n a) in log log plot b) the same but in linear axis c) result with RANSAC algorithm in log log plot d) result with RANSAC algorithm in linear axis

As can be seen above, the error increases very slowly for $\sigma_n < 0.1$ with a linear relationship in log log plot (panel (a) of figure 2), which means a power law relationship in a plot with linear spacing. Figure 2 (b) shows that for $\sigma_n \simeq 5$ the error increases rapidly for large σ_n. It seems the error increases exponentially for σ_n in the range ($\sim$1, $\sim$3). For $\sigma_n = 10$ the error is about 225 which is even larger than the x-axis range of the grid! For σ_ns larger than 3, the mean error does not change much, but it has a lot of scatter. At this point, it looks like the error is already very large and has been saturated and thus the estimate for F is broken down. To choose a maximum acceptable range for σ_n, I would choose $\sigma_n \simeq 0.1$, because it results in the error of $\sim$20 which is about 10% of

x-axis range of the image. After this, the error would increase rapidly and our estimate of F would be very inaccurate.

Since the question was not cleared about the algorithm we should use here, I repeated above with RANSAC-based 8 point method as well (Figure 2 panels (c) and (d)). RANSAC treats outliers and therefore the results are much better. The error with RANSAC is more than a factor of 10 smaller and it gets saturated at $\sigma_n = 10$ which is larger than $\sigma_n = 3$ for the case without RANSAC.

4 Part A-4

I repeated the above question, this time with turning off the Hartley normalization. As shown In Figure 3 (panel (a) and (b)), there is not any significant difference between F estimation with and without Hartely normalization using 8-point `linEstF` algorithm. This is probably because this algorithm does not treat the potential noise in the point measurements as oppose to gold-standard or MLESAC algorithms, therefore it is not sensitive to the normalization of image positions. For this reason, I did the above experiment again this time using RANSAC-based algorithm that depends on `linEstF` and is implemented in `dinoTestF.m` script (instead of pure 8-point algorithm). Figure 3 (panel c and d) shows the results of RANSAC-based algorithm for two cases with Hartley normalization and without it. First of all, I notice that tolerance of **both** cases (with and without Hatley Normalization) is greatly improved for RANSAC algorithm compared to pure 8-point algorithm. However, there are two major differences in the result of RANSAC algorithm with and without Hatley Normalization: 1) while the error of two cases are similar for $\sigma_n \lesssim 0.2$, they quickly deviate after this value. The error of the case without Hartley Normalization saturates at $\sigma_n \simeq 0.7$ as oppose to $\sigma_n \simeq 10$ for the case with Hartley Normalization. 2) The scatter for the case without Hartley normalization is higher than that of with Hartley normalization. Overall, the results of Figure 3 (panel c and d) highlights the importance of normalization for noisy image correspondences for algorithms that take into account the noisy image measurements (e.g., RANSAC). This is because with first two elements of the homogeneous image coordinate can take large values but the third coordinate is one and thus the scaling is poor.

Without normalization, I would pick the same $\sigma_n \simeq 0.1$ as the maximum value for obtaining an acceptable F matrix using pure 8-point algorithm.

5 Part A-5

For this part, I repeated the process of part 3 but for a dino with smaller size along z-axis, i.e., ScZ=0.1. I checked F_0, Mext, and Mint matrices before running the scrip and noted that they did not change from part 3 because these parameters are independent of the scene and solely depend on the cameras and their position. The results are shown in Figure 4. As can be seen, the estimate of F matrix is less tolerant for the Gaussian noise of image positions when ScZ=0.1 than the original ScZ=1. In the case of ScZ=0.1 the error is a factor of ~ 10 higher than the original case for $\sigma_n \simeq< 0.1$. For large $\sigma_n > 1$, the error of two cases are similar (Figure 4 panel (b)). This shows that when the dino is

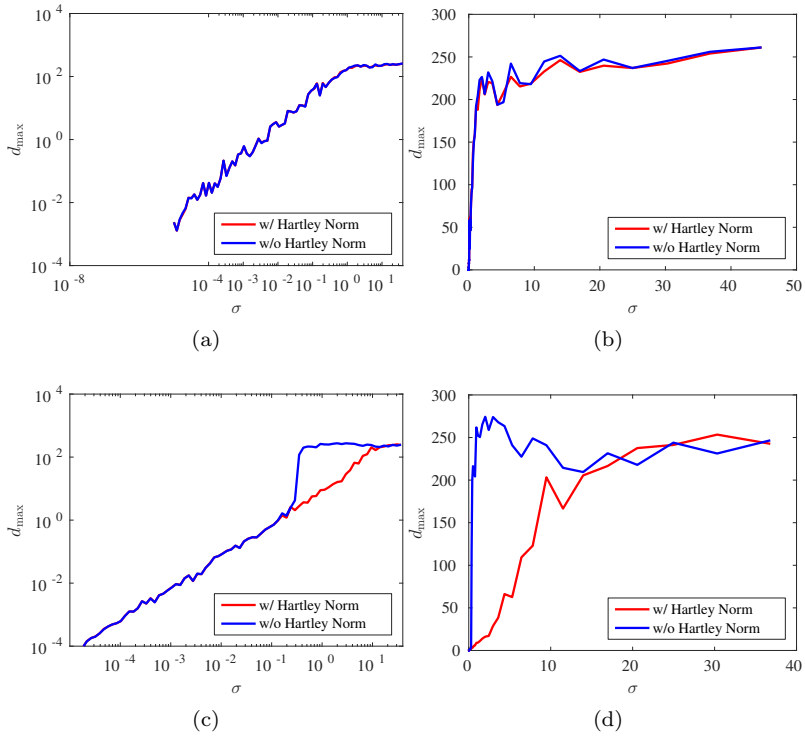


Figure 3: Maximum perpendicular error $d_{\max}$ per variance of Gaussian noise σ_n for twos cases: with Hartley Normalization and without it. a) in a log log plot b) the same in linear axis. c) the comparison using **RANSAC-based** 8 point estimation in log log plot d) same as panel (c) but in linear axis

flatter along z-aixs, it is more difficult to estimate F matrix. This is because the points are physically more closer to each other than the case of ScZ=1 and hence are less tolerant to the position noise. For example in the extreme case of ScZ=0 the dino is completely planer and Homography matrix is more suitable than F matrix for relating the image correspondences. I conclude that objects flatter along z-axis are less tolerant to the noise in image correspondences for estimating F matrix. For ScZ=0.1, I would pick $\sigma_n \simeq 0.001$ as the maximum value for obtaining an acceptable F matrix, because this value results in the error of ~ 20 which is 10% of x-axis scale.

Since the question was not cleared about the algorithm we should use here, I repeated above with RANSAC-based 8 point method as well (Figure 4 panels (c) and (d)). RANSAC treats outliers and therefore the results are much better for both ScZ=0.1 and ScZ=1 case. The error with RANSAC is more than a factor of 10 smaller than the corresponding case without RANSAC. Also the error gets saturated at a larger σ_n compared to the the corresponding case without RANSAC.

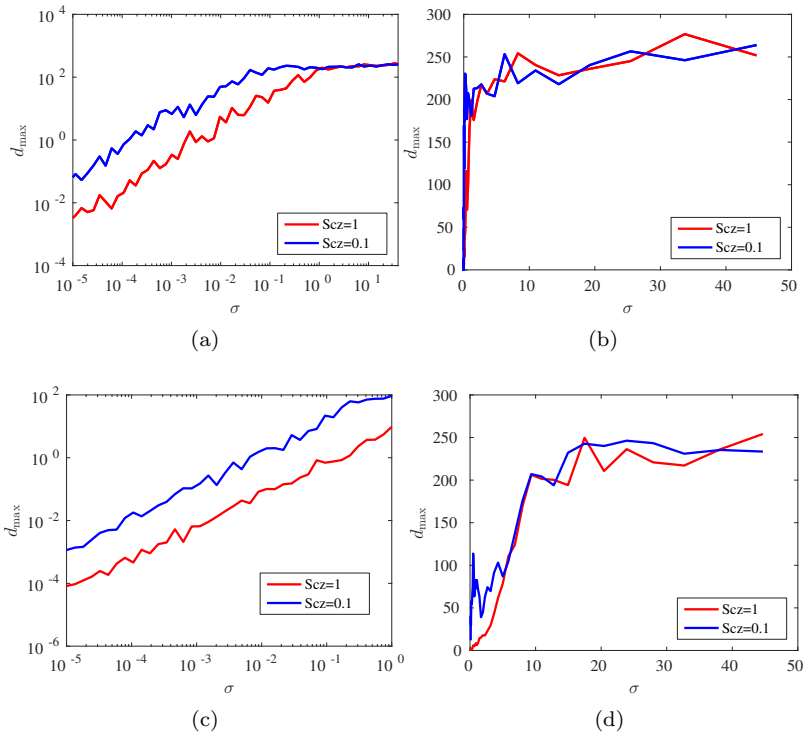


Figure 4: Maximum perpendicular error $d_{\max}$ per variance of Gaussian noise σ_n for twos cases: original $Scz = 0.1$ and $ScZ = 1$ a) in a log log plot b) the same in linear axis c) the comparison using **RANSAC-based** 8 point estimation in log log plot d) same as panel (c) but in linear axis

6 Part A-6

When I set the $d = (\pm 5, 0, -150)$, I got following F_0 matrix that is different from the original case:

$$F_0 = \begin{bmatrix} 0 & -3.3315e-05 & 0 \\ -3.3315e-05 & 0 & -0.099944 \\ 0 & 0.099944 & 0 \end{bmatrix}$$

Figure 5 compared the results of smaller camera separation to the original case. As can be see, for small $\sigma_n < 1$, the error of the original case is smaller by a factor 2, but as σ_n increases two errors gets close. However, after $\sigma_n > 2$ the smaller camera separation case slightly outperforms the original case. Therefore, I conclude that for small $\sigma_n < 1$ the error of the original case is smaller and after this there is not meaningful difference between two errors but for very large $\sigma_n > 2$ the error of smaller camera separation is smaller than the original case. This makes sense because when two cameras are very close two of F_0 matrix elements are very small as can be seen above, so in order to capture this small number, the F estimation is less tolerant to the error compared to original case. But this error gets saturated more quickly and for smaller perpendicular error

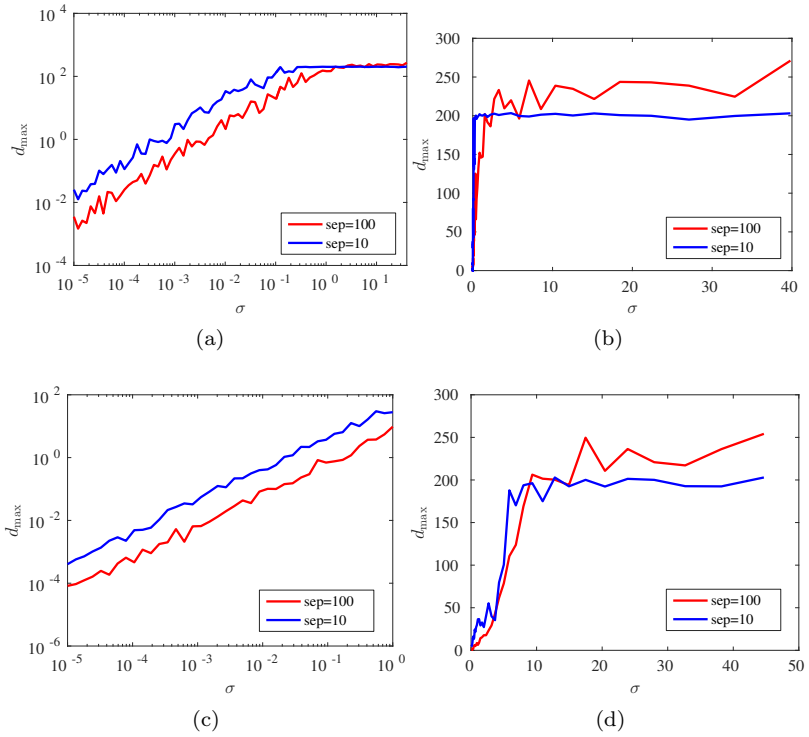


Figure 5: Maximum perpendicular error $d_{\max}$ per variance of Gaussian noise σ_n for twos cases: original $d = (\pm 50, 0, -150)$ (i.e. Separation of 100) and $d = (\pm 5, 0, -150)$ (i.e. Separation of 10). a) on a log log plot b) the same in linear axis c) the comparison using **RANSAC-based** 8 point estimation in log log plot d) same as panel (c) but in linear axis

(figure 5 panel (b)). To choose a maximum acceptable σ_n for camera separation of 10, I would choose $\sigma_n \simeq 0.05$, because it results in the error of $\sim$20 which is about 10% of x-axis range of the image.

Since the question was not cleared about the algorithm we should use here, I repeated above with RANSAC-based 8 point method as well (Figure 5 panels (c) and (d)). RANSAC treats outliers and therefore the results are much better for both Sep=10 and Sep=100 cases. The error with RANSAC is more than a factor of 10 smaller than the corresponding case without RANSAC. Also the error gets saturated at a larger σ_n compared to the the corresponding case without RANSAC.

Question B1

Based on equation (2) of the handout, we have,

$$\vec{q}_k \times (H\vec{p}_k) = 0$$

this means

$$\begin{bmatrix} 0 & -1 & +q_{k2} \\ 1 & 0 & -q_{k1} \\ -q_{k2} & q_{k1} & 0 \end{bmatrix} \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} p_{k1} \\ p_{k2} \\ 1 \end{bmatrix} = 0$$

$$\Rightarrow \begin{bmatrix} 0 & -1 & q_{k2} \\ 1 & 0 & -q_{k1} \\ -q_{k2} & q_{k1} & 0 \end{bmatrix} \begin{bmatrix} h_1 p_{k1} + h_2 p_{k2} + h_3 \\ h_4 p_{k1} + h_5 p_{k2} + h_6 \\ h_7 p_{k1} + h_8 p_{k2} + h_9 \end{bmatrix} = 0$$

Note that the third constrain above is a linear combination of the first two constrain, so we only consider the first two equation here:

$$\begin{cases} -h_4 p_{k1} - h_5 p_{k2} - h_6 + h_7 q_{k2} p_{k1} + h_8 q_{k2} p_{k2} + h_9 q_{k2} = 0 \\ h_1 p_{k1} + h_2 p_{k2} + h_3 - h_7 q_{k1} p_{k1} - h_8 q_{k1} p_{k2} - h_9 q_{k1} = 0 \end{cases}$$

So in the matrix form:

$$\begin{bmatrix} 0 & 0 & 0 & -p_{k1} & -p_{k2} & -1 & p_{k1} q_{k2} & p_{k2} q_{k2} & q_{k2} \\ p_{k1} & p_{k2} & 1 & 0 & 0 & 0 & -p_{k1} q_{k1} & -p_{k2} q_{k1} & -q_{k1} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = 0$$

Therefore in short:

$Ah = 0$ where A is the matrix of $\vec{q}_k$ & $\vec{p}_k$ in the above form and h consists of all H element.

In order least square estimation of H, we can write error as:

$$O(e) = \frac{1}{2} e^T e = \sum_{k=1}^{K} \sum_{j=1}^{2} [\vec{q}_k \times H\vec{p}_k]$$

$$= (Ah)^T (Ah)$$

$$= \frac{1}{2} h^T A^T A h$$

$$\Rightarrow \frac{\partial O(e)}{\partial h} = A^T A h = 0$$

Above equation is a homogeneous linear system of equation that can be solved using SVD decomposition.

Her I show that a colum of V is the eigenvector of A^TA, and eigenvector with eigenvalue zero is a solution to above equation.

$A = U\Sigma V^T$ where U & V are unitary matrices.
& U & V have orthogonal basis.

$$A^T A = V \Sigma^T U^T U \Sigma V^T$$

since U is orthogonal matrix, $U^T = U^{-1}$, therefor

$$A^T A = V \Sigma^T \Sigma V^T$$

now assume we multiply both sides with e.g., first colum of V, $\vec{v}_1$:

$$A^T A v_1 = V \Sigma^T \Sigma V^T \vec{v}_1$$

$$= V \Sigma^T \Sigma \begin{bmatrix} \|\vec{v}_1\| \\ 0 \\ 0 \\ \vdots \end{bmatrix}$$ since v_i is orthogonal to all other columns of V

$$= V \begin{bmatrix} \sigma_1^2 & & & \\ & \sigma_2^2 & 0 & \\ & & \ddots & \\ & 0 & & \sigma_N^2 \end{bmatrix} \begin{bmatrix} \|\vec{v}_1\| \\ 0 \\ 0 \\ \vdots \end{bmatrix}$$ where Σ is a diagonal matrix with element σ_i along diagonal.

therefore

$$A^TA v_1 = V \begin{bmatrix} \sigma_1^2 \|\vec{v}_1\| \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$= \sigma_1^2 \|\vec{v}_1\| \vec{v}_1 = \lambda \vec{v}_1$$

Therefore I showed a column of V is indeed an eigenvector of A^TA. So the column associated with eigenvalue zero is the solution of $A^TAh = 0$

Similar to linEstF, normalization of image corresponces helps the stability of the algorithm. Therefore the corresponderces will be multiplied by T_1 & T_2 depending on the camera:

$$T_p = \begin{bmatrix} s & 0 & b_1 \\ 0 & s & b_2 \\ 0 & 0 & 1 \end{bmatrix} \qquad T_q = \begin{bmatrix} s' & 0 & b_1' \\ 0 & s' & b_2' \\ 0 & 0 & 1 \end{bmatrix}$$

Therefore:

$$\vec{P}_k = (p_{1k}, p_{2k}, 1) \xrightarrow{T_p} \tilde{P}_k = (sp_{1k} + b_1, sp_{2k} + b_1, 1)$$

$$\vec{q}_k = (q_{1k}, q_{2k}, 1) \xrightarrow{T_q} \tilde{q}_k = (s'q_{1k} + b_1', s'q_{2k} + b_2', 1)$$

$$\alpha_k T_q \vec{q}_k = H' T_p \vec{P}_k \quad , \quad \alpha_k \vec{q}_k = H \vec{P}_k$$

$$\Rightarrow \alpha_k \vec{q}_k = T_q^{-1} H' T_p \vec{P}_k$$

$$\Rightarrow H = T_q^{-1} H' T_p$$

this is the output of normalization

7 Part B-2

For this question, I first came up with a new error meature. I chose below error metric:

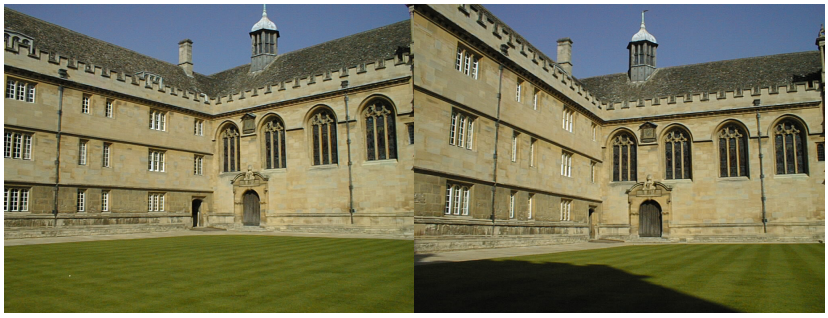
$$\hat{w}_k = Hp_k \tag{9}$$

$$\hat{q}_k = \frac{\hat{w}_k}{\hat{w}_k(3)} \tag{10}$$

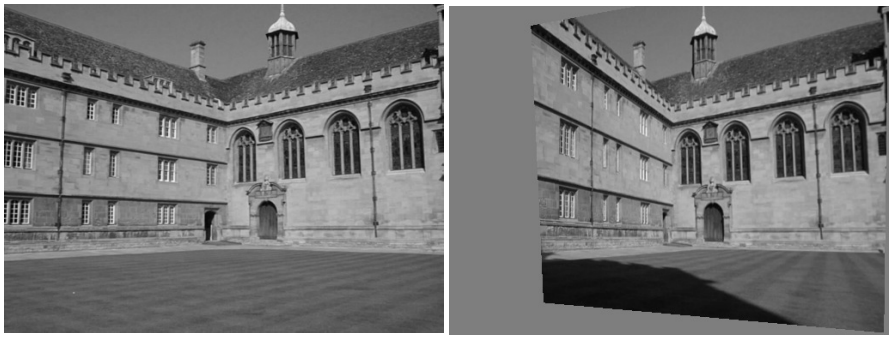
$$e = ||\hat{q}_k - q_k|| \tag{11}$$

where e is the error between the real correspondence and the estimated value in the left image. I define my error metric this way because it has a distance dimension and therefore we can easily compare it to σ value. Also, it is very similar to what has been implemented for F estimation and since that worked well (see section 4 above), I have chosen to follow a similar method. This error would be used in RANSAC algorithm and is compared against a default $\sigma = 2$ to detect outliers and inliers. I also made following changes to the grappleFmatrix and saved as grapple2DHmatrix: 1) I used linEstH instead of linEstF, 2) changed the number of point for input linEstH to 4, 3) the scripts now plot warped left image as well as right image. Note that I realized the correspondences were swapped in corrPntsX files, so I had to inverse H to get the expected results, I fixed this issue by swapping back the columns of corrPntsX.

The results for all image correspondences provided (001-002 / 001-003 / 001-004 / 001-005) are provided below. For all results I turned Hartley normalization on. I am comparing WarpedLeft–RightImage and WarpedRight–LeftImage for all image pairs. As can be seen below, I am getting reasonable results for all pairs except for 001-003. I notice that for this pair the camera seem to be very far away from each other and they are viewing the scene from different angles. Therefore it is possible that the error in the position correspondences are high, which results in poor Homorgraphy. Also it could be the case that for large camera separations our method of estimating H is less sensitive to noise (of course this hypothesis has to be tested for a picture pair that the ground truth is known). Finally, I stress that Homography is a reasonable method for planar scenes, i.e., pair images of a plane, but here we used it on 3D scenes (Wadham college), therefore one should not expect it to work perfectly.



(a) (b)



(c) (d)

(e) (f)

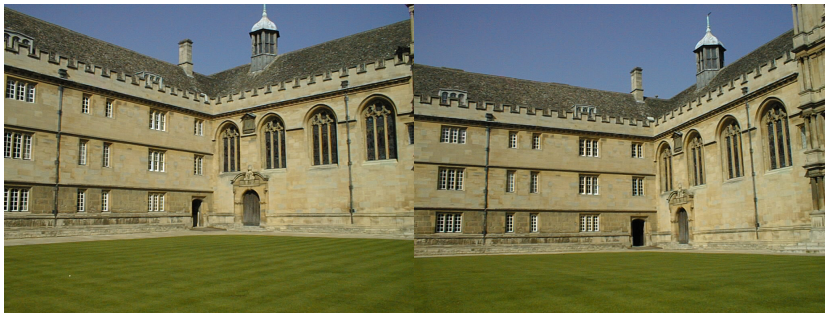
Figure 6: (a) image 001 (b) image 005 (c) Left image (d) Warped Right (e) Right image (f) Warped right

(a) (b)

(c) (d)

(e) (f)

Figure 7: (a) image 001 (b) image 002 (c) Left image (d) Warped Right (e) Right image (f) Warped right

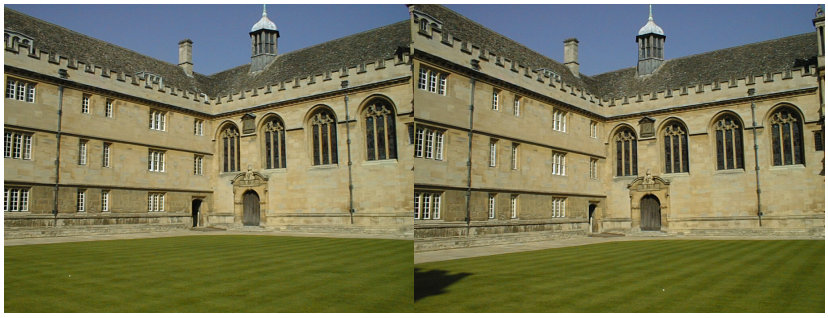


(a) (b)

(c) (d)

(e) (f)

Figure 8: (a) image 001 (b) image 003 (c) Left image (d) Warped Right (e) Right image (f) Warped right



(a) (b)

(c) (d)

(e) (f)

Figure 9: (a) image 001 (b) image 004 (c) Left image (d) Warped Right (e) Right image (f) Warped right