HW 4 - Homographies

Oren Elmakis - 311265516

Alon Spinner - 305184335

1 Part 1: Theory

1.1 Q1.1

Question 1.1. Suppose two cameras fixate on a point P(see Figure 1) in space such that their principal axes intersect at that point. Show that if the image coordinates are normalized so that the coordinate origin (0,0) coincides with the principal point, the F33 element of the fundamental matrix is zero.

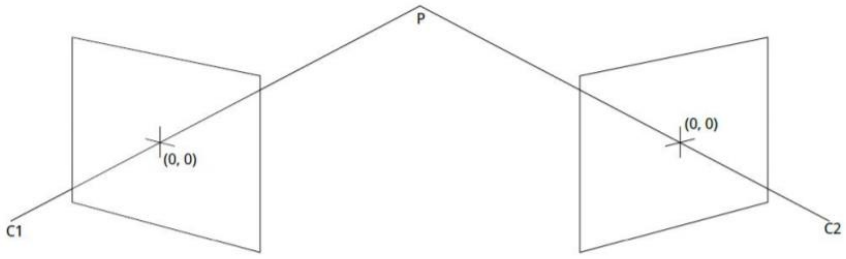


Figure 1: Figure for Q1.1. $C1$ and $C2$ are the optical centers. The principal axes intersect at point w (P in the figure).

The fundamental matrix holds:

$$x_1^{P\prime} F x_2^P = 0$$

Where x_i^P denotes the projection of the 3D point Ponto the image obtained in camera position i in homogenous coordinates.

$$x_i^P = [u_i, v_i, 1]$$

Writing off F explicitly $F = f_{ij}$:

$$[u_1, v_1, 1] \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = 0$$

Plugging in the values for u_i, v_i and compute the multiplication:

$$[0,0,1] \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 0$$

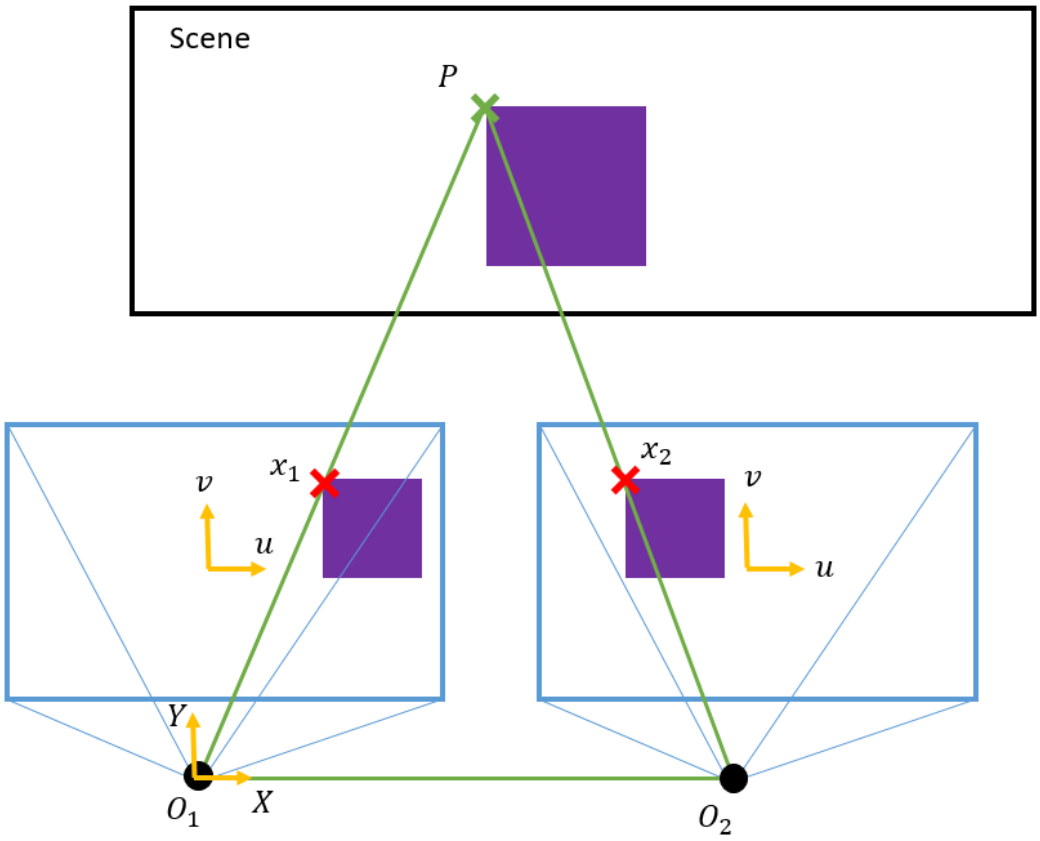
$$[0,0,1] \begin{bmatrix} f_{13} \\ f_{23} \\ f_{33} \end{bmatrix} = 0$$

$$f_{33} = 0$$

1.2 Q1.2

Question 1.2. Consider the case of two cameras viewing an object such that the second camera differs from the first by a pure translation that is parallel to the x-axis. Show that the epipolar lines in the two cameras are also parallel to the x-axis. Backup your argument with relevant equations.

The configuration described can be drawn as follows:



The Essential matrix E can be decomposed to the rotation and translation between the cameras written in the XYZ axes system:

$$E = [t]_x R$$

We know there is no rotation between the cameras, and that the translation is in the X-axis only. Hence:

$$R \equiv I$$

$$t = [s, 0, 0]$$

$$\rightarrow E = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -s \\ 0 & s & 0 \end{bmatrix}$$

To obtain the implicit equations for an epipolar line seen on the image from camera 1, one would solve:

$$l_1 = Ex_2$$

Where x_2 represents some point on the image plane of camera 2 in its coordinates, and l_1 represents a corresponding epipolar line on the image plane of camera 1 in its coordiantes.

Plugging in the constraints for E and solving for l_1:

$$l_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -s \\ 0 & s & 0 \end{bmatrix} \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = [0, -s, +sv_2]$$

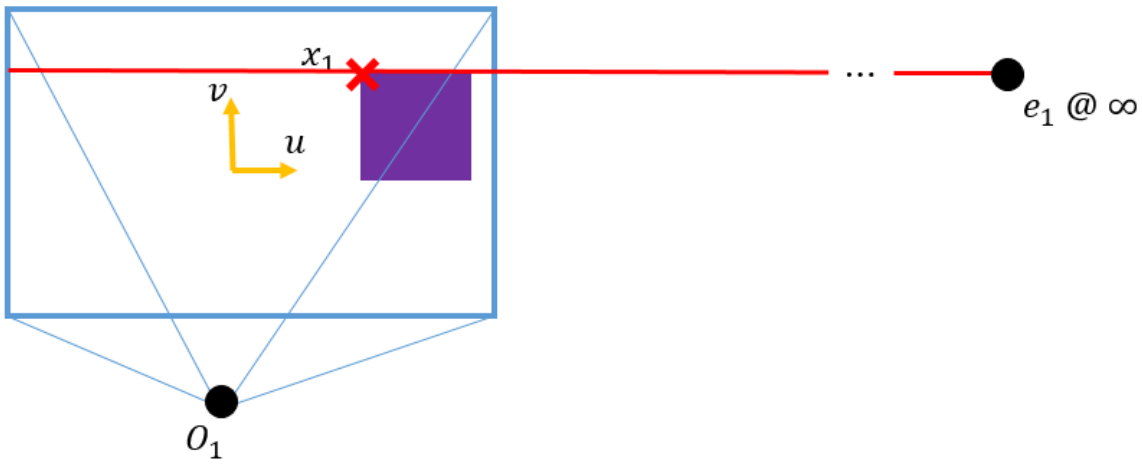
Or in "other words":

$$x \cdot 0 - s \cdot y + sv_2 = 0$$

$$\rightarrow y = v_2$$

The symmetric nature of the problem will promise us the same result if looking on epipolar lines on camera 2.

To obtain the intuition for the algebraic solution we drew the epipole of camera 1.
e_1 must be at infinity as it is the place where all the epipolar lines meet and they are parallel to the X-axis.



Usually, we could describe l_1 as the projection of PO_2 on the image plane of camera 1, or equivalently, the line connecting x_1 and e_1 on camera 1 image plane,
but as e_1 is at infinity it provides for the direction of line l_1, which together with the point x_1 completes the definition.

1.3 Q1.3

Question 1.3. Suppose we have an inertial sensor which gives us the accurate positions(Ri and ti, where R is the rotation matrix and t is corresponding translation vector) of the robot at time i. What will be the effective rotation (Rrel) and translation (trel) between two frames at different time stamps? Suppose the camera intrinsics (K) are known, express the essential matrix(E) and the fundamental matrix (F) in terms of K, Rrel and trel.

We are asked to compute $F = F(K, R_{rel}, t_{rel})$ and $E = E(K, R_{rel}, t_{rel})$

Let $X = [x, y, z]^T$ be a point in the world (3D) and $x_i = [u_i, v_i, 1]^T$ be the projection of X on the camera, in camera coordinates.

We established the transformation between world and camera coordinates in the lecture:

$$x_i = K(R_i X + t_i)$$

$$x_{i+1} = K(R_{i+1} X + t_{i+1})$$

Extracting X from the first equation:

$$X = R_i^{-1}(K^{-1}x_i - t_i) = R_i^T(K^{-1}x_i - t_i)$$

Plugging X in the second equation:

$$x_{i+1} = K(R_{i+1}R_i^T(K^{-1}x_i - t_i) + t_{i+1})$$

$$x_{i+1} = \underbrace{KR_{i+1}R_i^T K^{-1}}_{R_{rel}} x_i + \underbrace{(-KR_{i+1}R_i^T t_i + Kt_{i+1})}_{t_{rel}}$$

And so:

$$R_{rel} = KR_{i+1}R_i^T K^{-1}$$

$$t_{rel} = -KR_{i+1}R_i^T t_i + Kt_{i+1}$$

The essential matrix is defined:

$$E = [t_{rel}]_x R_{rel}$$

And the fundamental matrix (using the same camera):

$$F = K^{-T} E K^{-1}$$

$$F = K^{-T}[t_{rel}]_x R_{rel}\ K^{-1}$$

2 Part 2 – Planar Homographies

2.1 Q2.0

Given a set of homogenous points p_i in an image taken by camera C_1, and corresponding homogenous points q_i in an image taken by C_2, suppose there exists a homography H such that:

$$p_i = Hq_i$$

a) Given N correspondence we are to derive a set of $2N$ independent linear equations in the form $Ah = 0$

We denote: $H = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix}$, $p_i = [x_i, y_i, 1]$, and $q_i = [u_i, v_i, 1]$

$$\begin{bmatrix} [x_1, y_1, 1] \\ \vdots \\ [x_N, y_N, 1] \end{bmatrix}^T = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} [u_1, v_1, 1] \\ \vdots \\ [u, v_N, 1] \end{bmatrix}^T$$

Solving for $i = 1$:

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix}$$

Opening up the terms:

$$\begin{cases} x_1 = h_1u_1 + h_2v_1 + h_3 \\ y_1 = h_4u_1 + h_5v_1 + h_6 \\ 1 = h_7u_1 + h_8v_1 + h_9 \end{cases}$$

Plugging the third equation into the first two:

$$\begin{cases} (h_7u_1 + h_8v_1 + h_9)x_1 = h_1u_1 + h_2v_1 + h_3 \\ (h_7u_1 + h_8v_1 + h_9)y_1 = h_4u_1 + h_5v_1 + h_6 \end{cases}$$

Rearranging:

$$\begin{cases} -u_1h_1 - v_1h_2 - h_3 + u_1x_1h_7 + v_1x_1h_8 + x_1h_9 = 0 \\ -u_1h_4 - v_1h_5 - h_6 + u_1y_1h_7 + v_1y_1h_8 + y_1h_9 = 0 \end{cases}$$

Each pair of matching points i will provide two equations. Hence, the lot of points will provide the $2N$ equations we were asked to find.

For $i = 1 \dots N$:

$$\begin{cases} -u_i h_1 - v_i h_2 - h_3 + u_i x_1 h_7 + v_i x_i h_8 + x_i h_9 = 0 \\ -u_i h_4 - v_i h_5 - h_6 + u_i y_i h_7 + v_i y_i h_8 + y_i h_9 = 0 \end{cases}$$

b) Write down an expression for A

Writing the $i'th$ equation in matrix form:

$$\begin{bmatrix} -u_i & -v_i & -1 & 0 & 0 & 0 & u_i x_i & v_i x_i & x_i \\ 0 & 0 & 0 & -u_i & -v_i & -1 & u_i y_i & v_i y_i & y_i \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = 0$$

Concatenating for all i points:

$$\underbrace{\begin{bmatrix} -u_1 & -v_1 & -1 & 0 & 0 & 0 & u_1 x_1 & v_1 x_1 & x_1 \\ 0 & 0 & 0 & -u_1 & -v_1 & -1 & u_1 y_1 & v_1 y_1 & y_1 \\ & & & & \vdots & & & & \\ -u_N & -v_N & -1 & 0 & 0 & 0 & u_N x_N & v_N x_N & x_N \\ 0 & 0 & 0 & -u_N & -v_N & -1 & u_N y_N & v_N y_N & y_N \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix}}_{h} = 0$$

$$Ah = 0$$

c) How many elements are in h? How many point pairs are required to solve this system?

<u>In Short:</u>
There are 9 elements in h, but as homogenous coordinates can only be expressed up to scale, only 8 elements of h are relevant.

<u>The longer version:</u>
Homogenous coordinates can be expressed up to scale, that is:
$[x, y, 1]$ & $[xw, yw, w]$ both represent the same point.

Looking at the homography equation $p_i = Hq_i$

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = H \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix}$$

As q_i is homogenous coordinates, multiplying q_i by some factor w will not change the resulting p_i:

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = H \begin{bmatrix} wu_i \\ wv_i \\ w \end{bmatrix}$$

Taking w out of q_i and putting it next to H

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = wH \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix}$$

Hence any homography matrix H can be defined up to a scale and the equation will hold.

This allows us to normalize H by fixing one if its values to 1 (usually h_9) ensuring us 8 free parameters.

Another popular constraint is $|H| = 1$.

$$H = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & 1 \end{bmatrix}$$

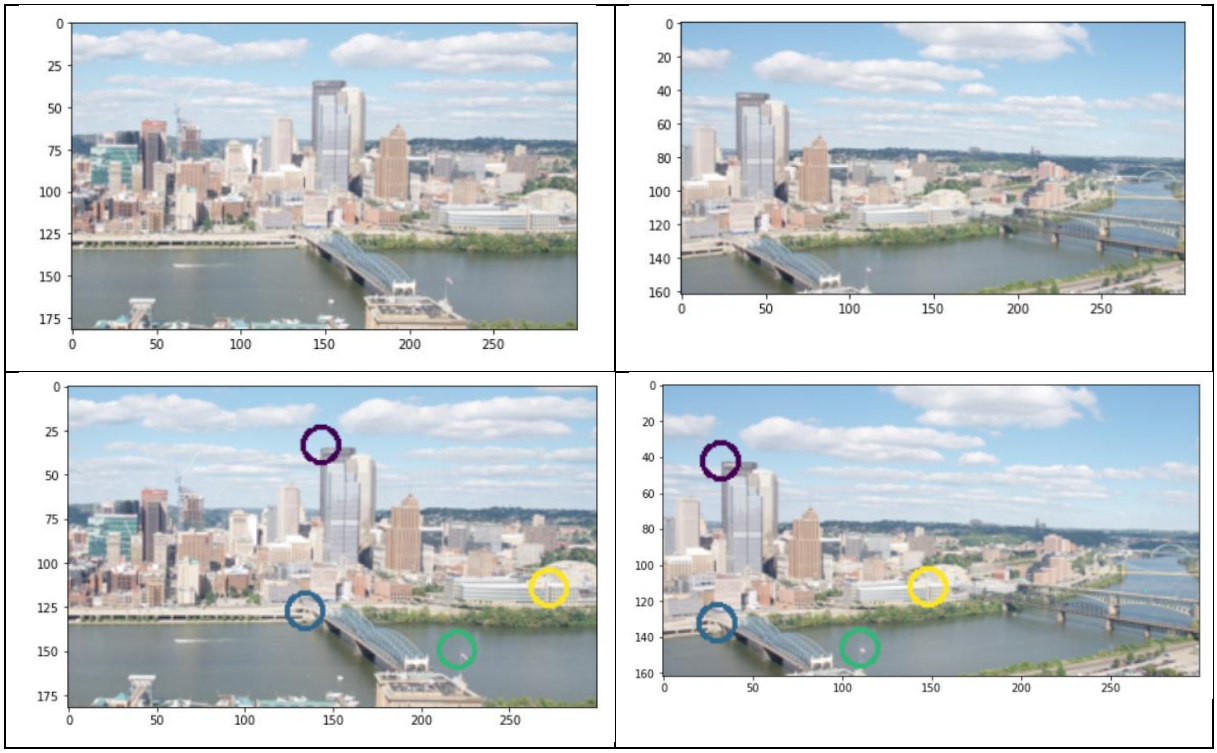
In order to solve for the 8 parameters, 8 equations are required.
Each point correspondence provides 2 equations, hence 4 correspondences are required as a minimum – that is 4 points in each image.

2.2 Q2.1 – Manual finding corresponding points

We were asked to find corresponding points in two images manually. We used the function $ginput()$ from $matplotlib$ on the *Incline* images.

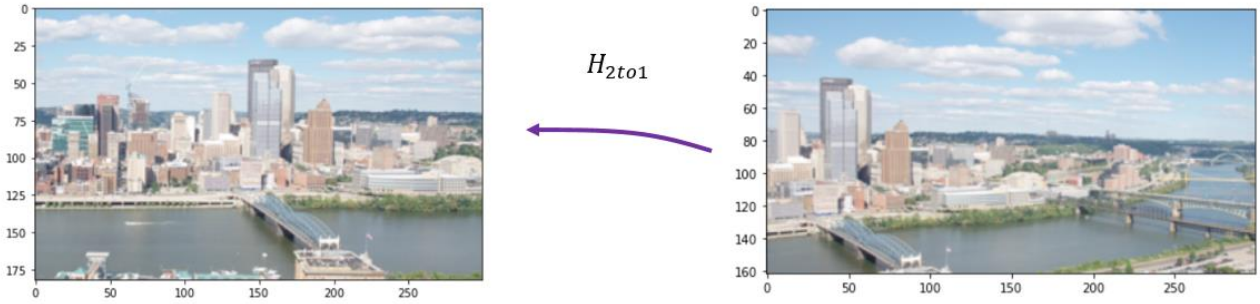
We go further in on how to pick said points in the following question $Q2.2$

Incline Left – image1	Incline Right – image 2



2.3 Q2.2 – Calculate transformation

We were asked to calculate the homography that transforms pixels in *Incline Right* to *Incline Left*. We scaled the resolution of the images down optimizing the time spent on this project.



We computed H_{2to1} by solving the LMS problem of the system of equation that is constructed in $Q2.0$ (see above) using EVD.

In other words:

$$Ah = 0 \rightarrow \ argmin_h \left|\left|Ah\right|\right|_2^2 s.t. \left|\left|h\right|\right|_2^2 = 1$$

Finding the Eigen values λ and their corresponding Eigen vectors v of $A^T A$

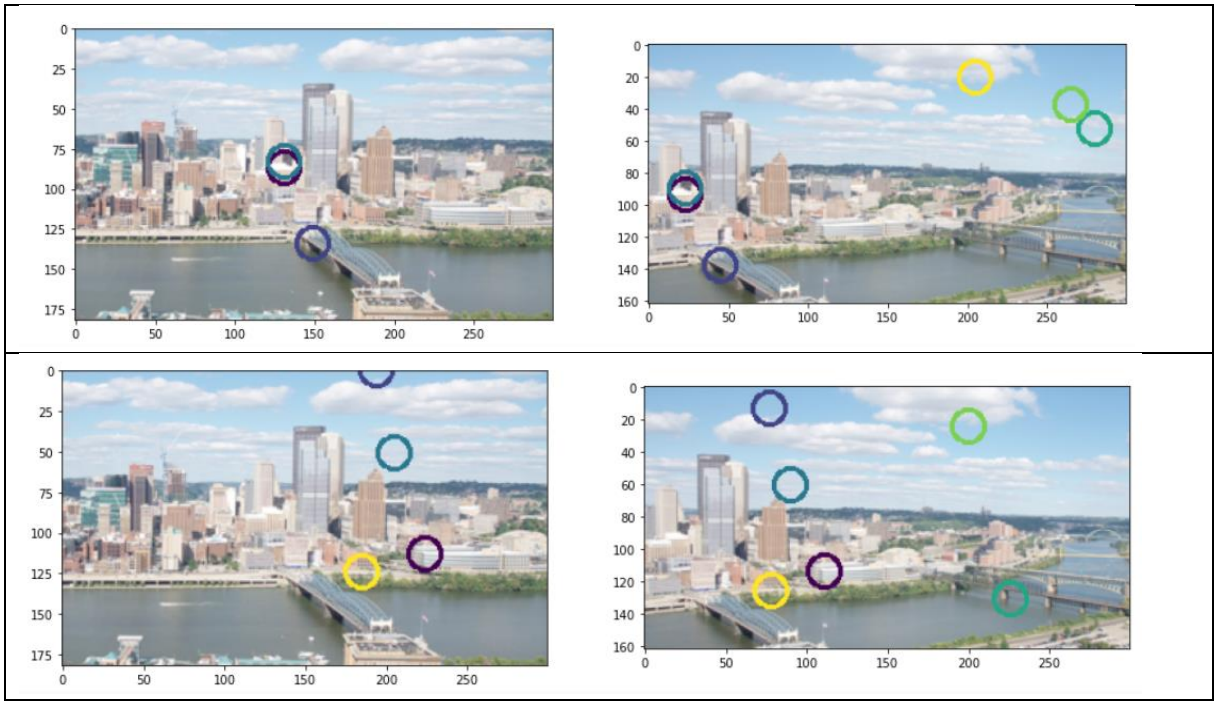
$$\lambda, v = np.linalg.eig(A^T A)$$

$$h = v(\min(\lambda))$$

$$H_{2to1} = reshape(h, (3,3))$$

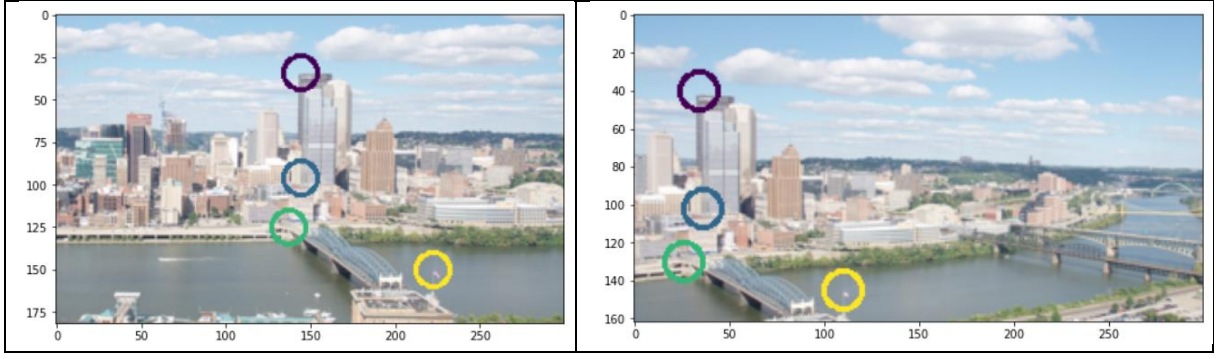
We found this method of solving the LMS problem superior to the SVD as it is quicker for a large point set.

We tested our computation with a function that randomly selects points on *Incline Right*, passes them through the homography and prints them on to *Incline Left*. Here are two results:

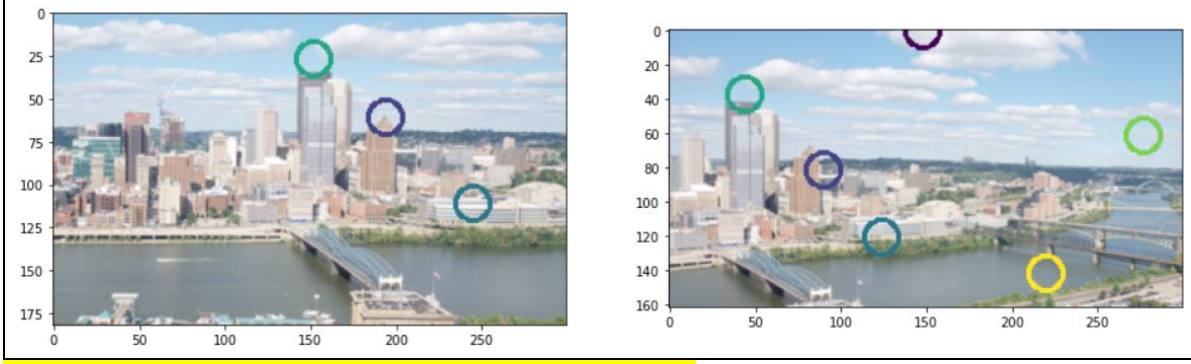


Note:
Calculating a homography when 3 points are lying on the same line in the source image is a terrible-terrible mistake. Remember, homography transformation takes two parallel lines and makes them unparalleled. How can it know to do that if it doesn't accept two edges to begin with?
A good intuition for choosing 4 points in the source image is to try and maximize the area of their convex hull holds. This also has the advantage of minimizing the error in homography gradients.

Chosen Points for a bad homography



Testing the bad homography with random points

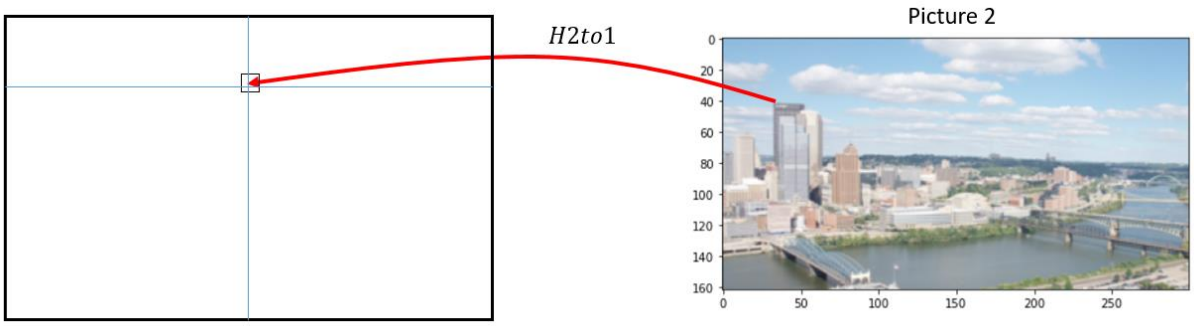


We can see a shift in the purple and dark blue circles.

2.4 Q2.3 - Image Warping

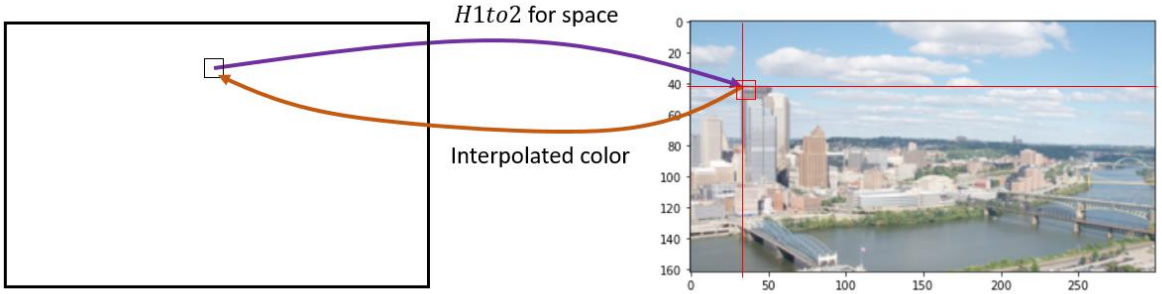
We were asked to create a function which accepts an image, *im,* a homography, *H*, and an *out_size*, and computes the transformation of *im* via *H*.
In addition, we were to compare *linear* vs *cubic* color interpolations using *LAB* and *RGB* color spaces.

At first glance, one would like to apply H_{2to1} to each pixel in picture 2 and paint the corresponding pixel in a template image.

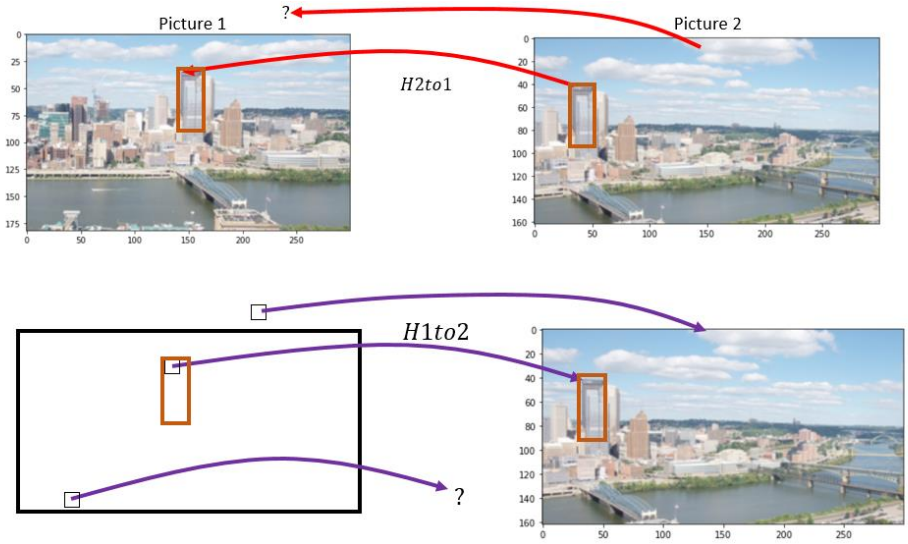


Alas, the transformed pixels from picture 2 are not inclined to fall on a grid, to avoid holes a technique called inverse homography is used:

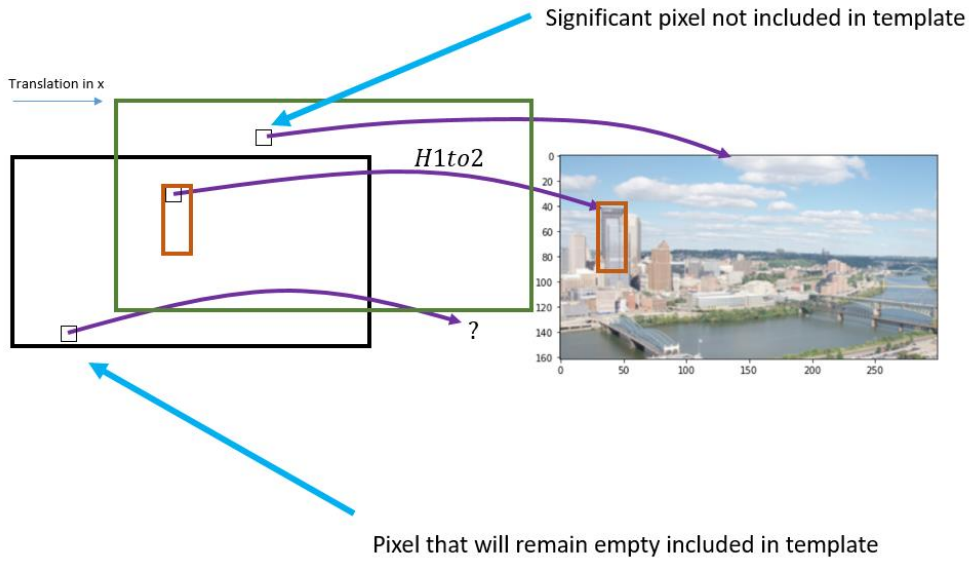
1. Create an empty template of the transformed picture 2
2. fill the template by applying the inverse homography on the pixels, and then draining the color from picture 2



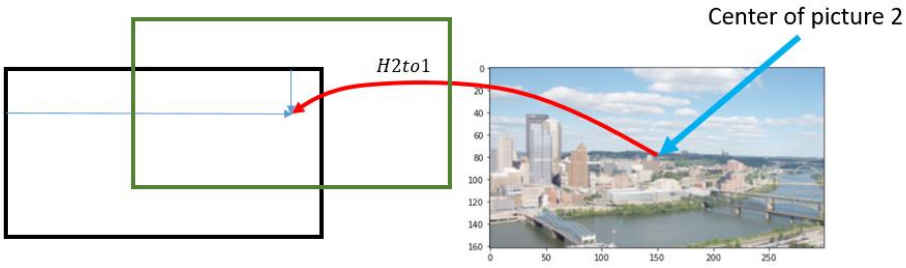
There is also something to say about the *out_size*:
If we take the template of the transformed picture 2 to be the size of picture 1, we won't be able to "hit" the entirety of picture 2. Even more, some pixels may translate to negative values!

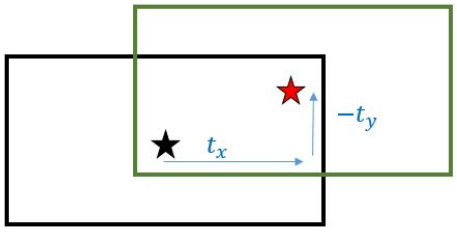


To solve this problem, we change the size of the template image, and introduce a translation to the pixel coordinates



One would ask… How to calculate the translation? Transform the center of picture 2 and see where it leads!





$$C = [x, y, 1]'$$

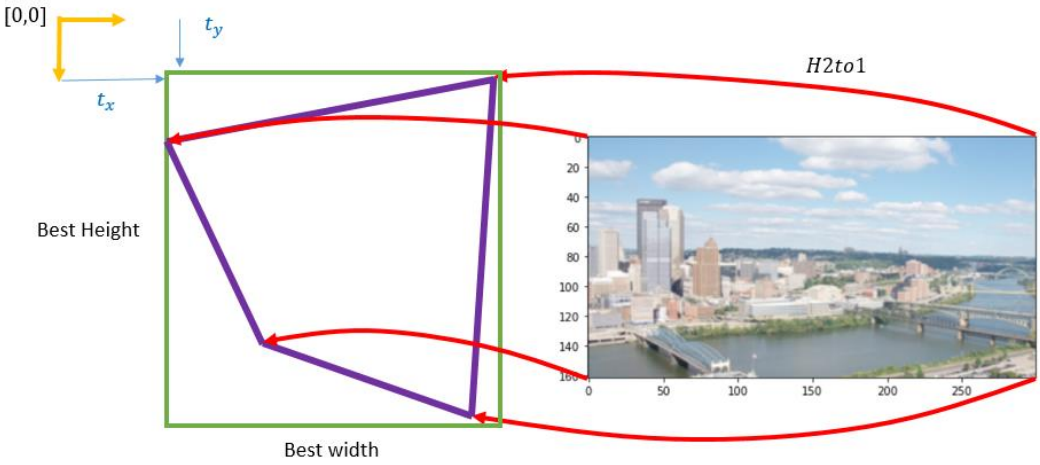
$$translation = H2to1 \cdot C_{picutre\ 2} - C_{template}$$

$$x_{template} = arange(template\ shape[1]) + translation[0]$$

$$y_{template} = arange(template\ shape[0]) + translation[1]$$

But this solution is "מצוץ מהאצבע" and requires us to guess a large enough *output_size*.

<u>Second thought:</u>
Decide on the optimal translation and size of the template image by applying the homography on the corners of picture 2.
Note: homography keeps lines as lines!

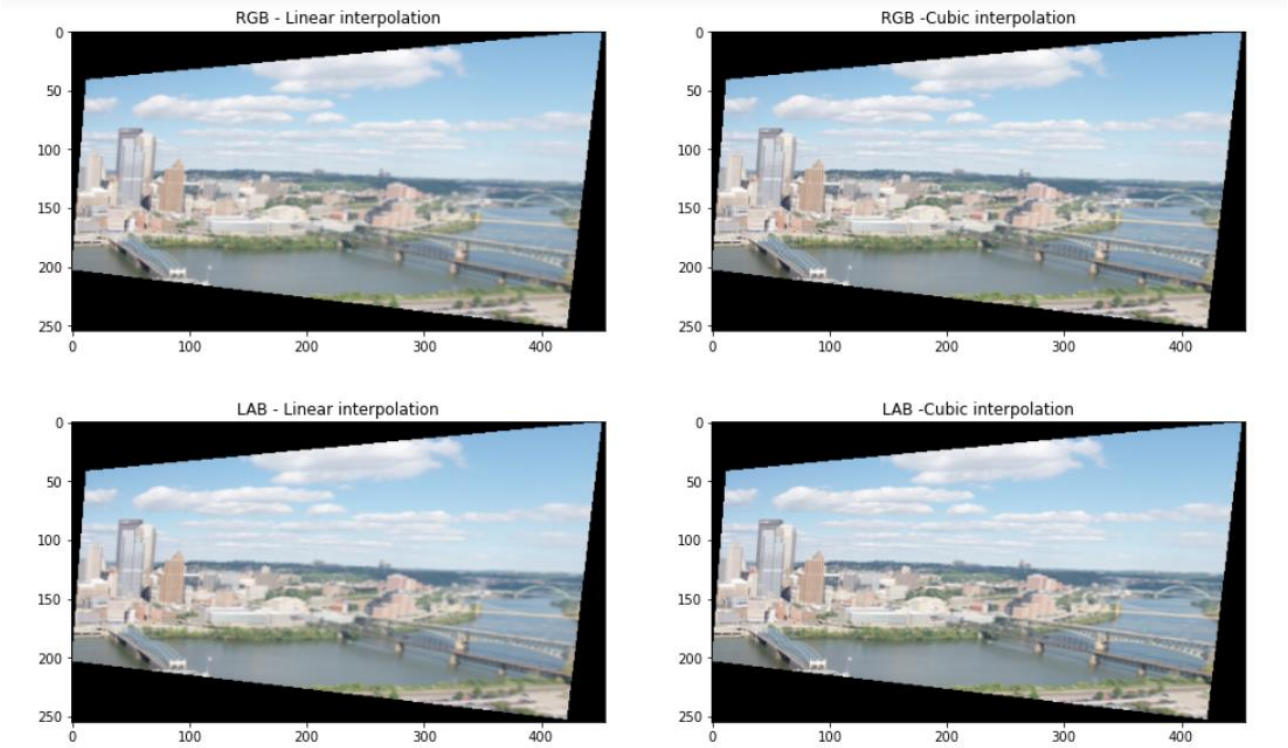


$$translation = \min(H2to1 \cdot P_{corners})$$

$$\text{Best Height} = ceil(\max(H2to1 \cdot P_{corners} \cdot \hat{y}) - \min(H2to1 \cdot P_{corners} \cdot \hat{y}))$$

$$\text{Best Width} = ceil(\max(H2to1 \cdot P_{corners} \cdot \hat{x}) - \min(H2to1 \cdot P_{corners} \cdot \hat{x}))$$

This worked really well – it allows to compute the smallest template image required, and also produces a boundary between pixels that will be and pixels that will remain black allowing us to accelerate the computation.

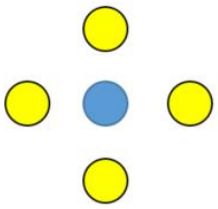
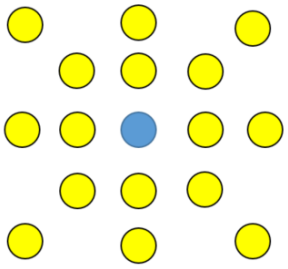


2.4.1 Interpolation method

The 4 pictures up to look identical to the naked eye. Let us zoom in a little and see the difference between the *RGB – Linear interpolation* and *RGB – cubic interpolation*:

RGB – Linear interpolation	*RGB – cubic interpolation*
	

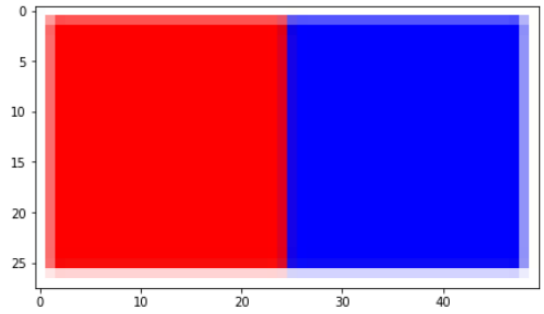
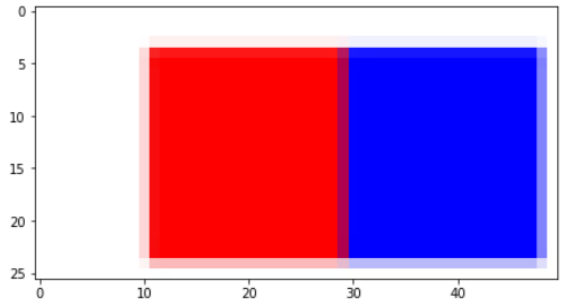
The image produced by cubic interpolation is definitely sharper which is of no surprise seeing how they are both computed:

interpolation formula and points (not pixels) with which it is computed	
Bilinear	Bicubic
$p(x,y) = a_0 + a_1x + a_2y + a_3xy$	$p(x,y) = [1\ x\ x^2\ x^3]\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}\begin{bmatrix} 1 \\ y \\ y^2 \\ y^3 \end{bmatrix}$
	

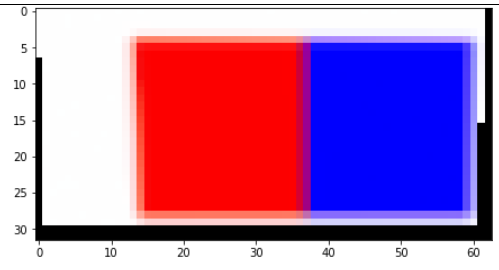
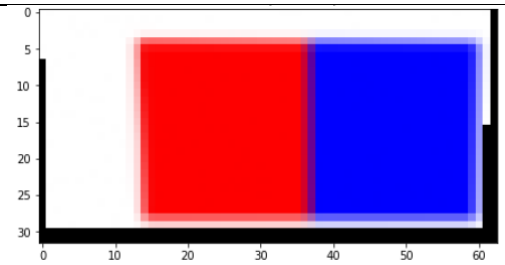
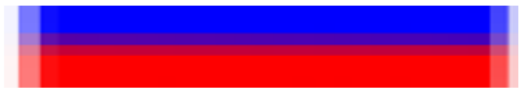
The difference in computation time between the interpolation methods is surprisingly negligible.

2.4.2 Color-space for interpolation

Unfortunately, even with a spyglass the most experience detective couldn't tell the difference in the *incline* images.
We created a test image, scaled and translated it:

Test image 1	*Test image 2*
	

After warping *Test Image 2* to *Test Image 1*:

LAB interpolation	RGB interpoaltion
	
middle pixels cropped, zoom and rotated	
	

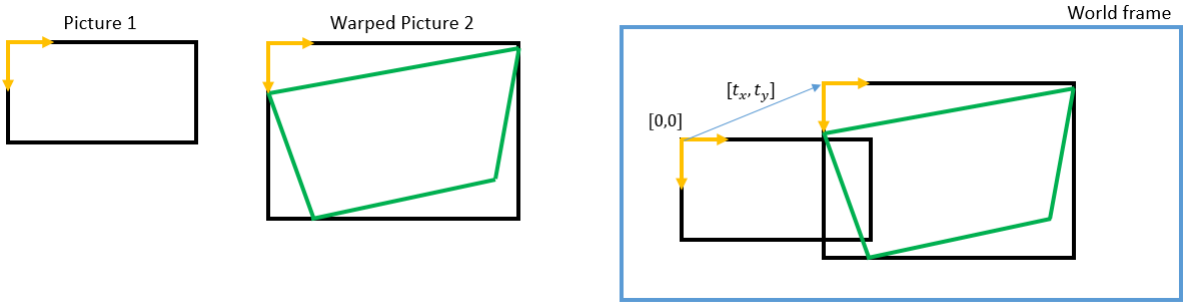
The RGB colors-pace holds Chroma and intensity in the same dimensions, hence moving from one Chroma to another also changes the intensity.
You can see how the shift between blue and red is darker in the RGB interpolation

Note:
in addition to an image and a homography matrix, our function *warpH* accepts interpolation method and color-space method, and returns the translation in addition to the warped image so that stitching could be made efficiently.

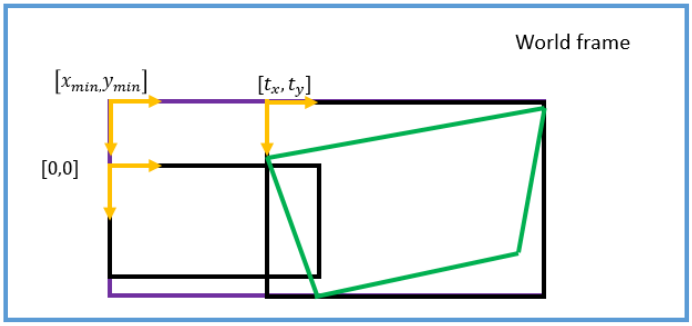
2.5 Q2.4 – Panorama Stitching

In this section we were to stitch the two images (picture1 and warped picture2) together. To do so, we need to account for the translation of the warped image, and the sizes of both images:

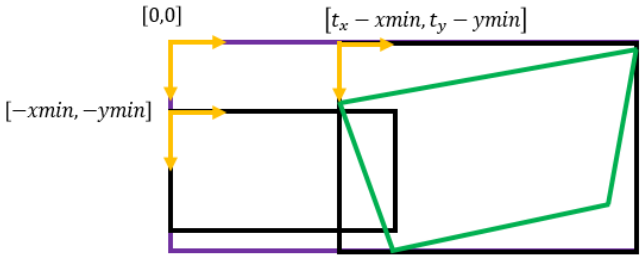
1. Apply the translation to the warped image pixels such that the data of both images is in context of picture 1 coordinates system.



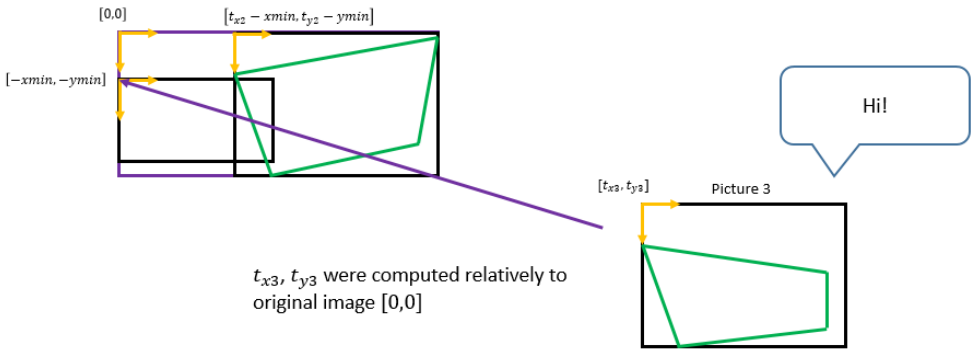
2. Find the minimal bounding box (width, height and location) required to hold the two pictures using *min*,*max* operators.

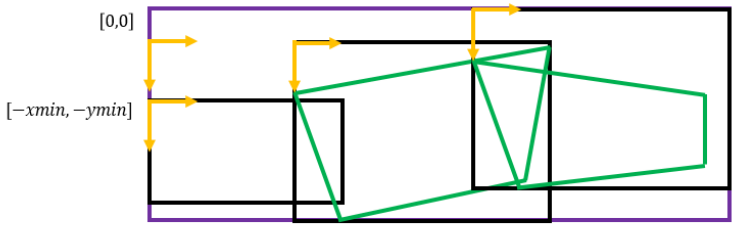


3. Switch coordinates system to that of the bounding box
4. Broadcast the pictures on to the bounding box using the respective translation for each.



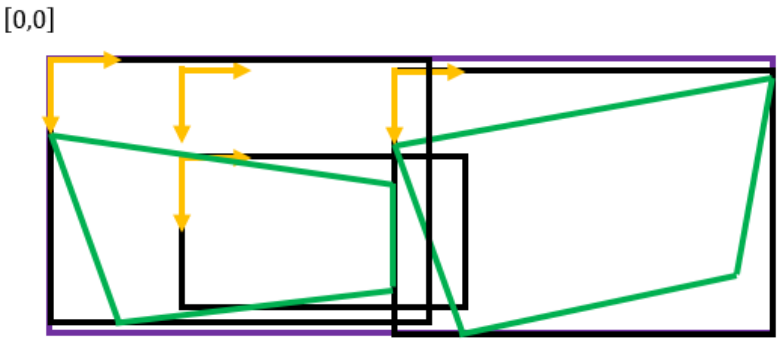
When stitching more than two images for a panorama, the translations to the new coordinate system need to be accounted for and summed up!





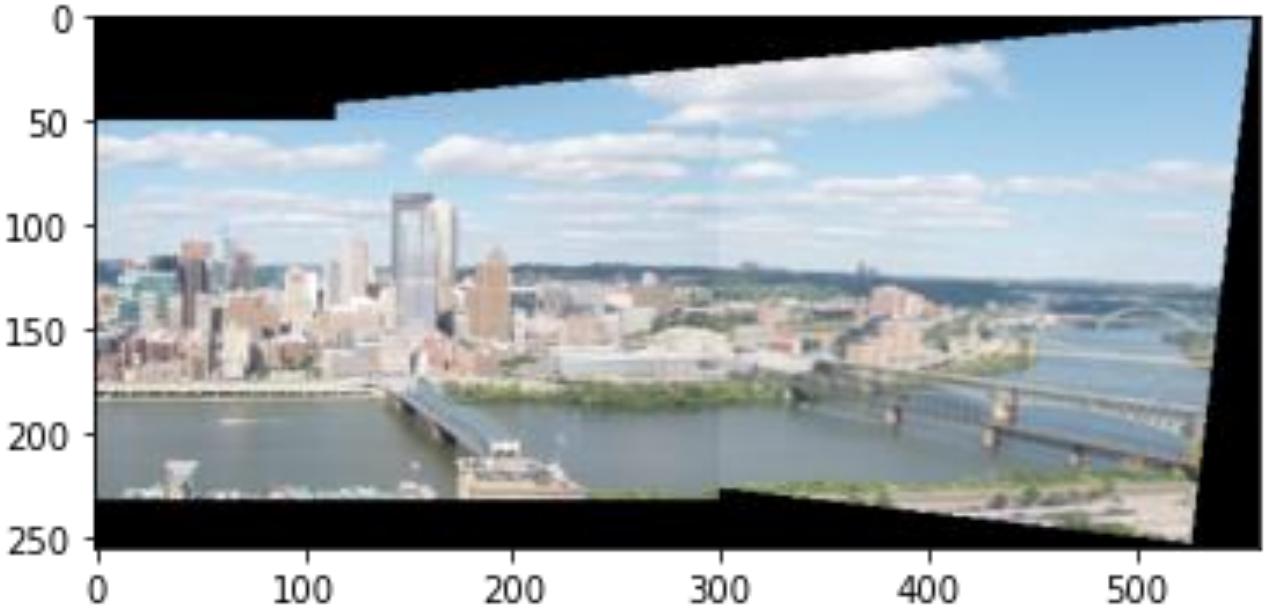
$$t_{x3} \coloneqq t_{x3} - xmin$$
$$t_{y3} \coloneqq t_{y3} - ymin$$

Depending on the stitch, the *origin* or *center* or *rectangular* image can change origin location relatively to the panorama. This shift should be accounted for as shown in step 5 on each new image that is stitched.



Our stitching functions accept two images and the translation parameter discussed in $Q2.3$, and also return a shift parameter $[-xmin, -ymin]$ to allow for continuous panorama stitching in addition to the panoramic image.

Enough with the blabber I hear you say… SHOW ME THE RESUTLS:



2.6 Q2.5 – Autonomous panorama stitching using SIFT

In this section we were to find points for the homography automatically using SIFT feature detector. As SIFT is copy-righted and removed from cv2, we used *pysift* which is a python implementation of the technique found on https://github.com/rmislam/PythonSIFT.

We tested the following methods, all with norm2 as a distance measurement:

1. Brute Force - crosscheck matching, choosing N best matches
2. Brute Force – KNN(k=2) matching using Lowe's ratio
3. FLANN – KNN(k=2) matching using Lowe's ratio

Brute Force:
The Brute force method would test each feature descriptor against all others.

FLANN - Fast Library for Approximate Nearest Neighbors:
An alternative to the *Brute F*orce method. FLANN contains a collection of algorithms optimized for fast nearest neighbor search in large datasets and for high dimensional features.
We initialized it inserting the amount of k-d trees to be constructed (5), and the number of times the trees in the index should be recursively traversed (50).
Those values were decided as they showed up in multiple examples for different pictures.

KNN – K nearest neighbors:
Finds the K features with are closest to feature f_m in distance

Cross-Check:
If the feature f_m is closest in distance to feature f_n, and in turn feature f_n is closest in distance to feature f_m: Consider the two features a match!

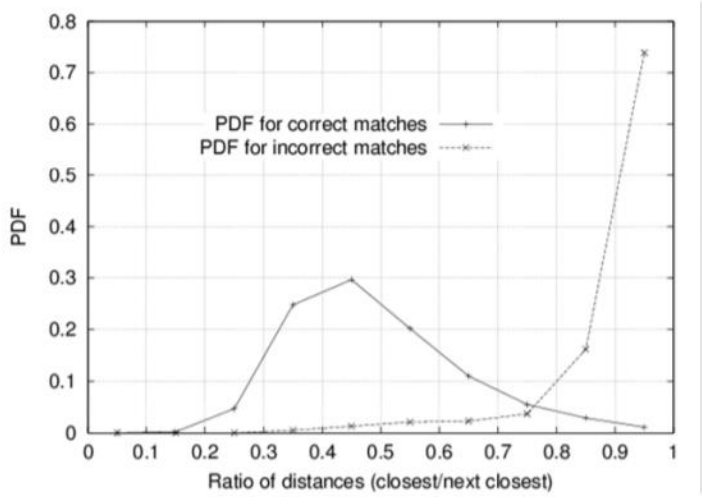
Lowe's ratio:
Lowe's ratio test if matches are ambiguous and should be removed.
It does so by testing the descriptor's distance two its two best matches.

The assumption is that the match with the smallest distance is the correct match, and the match with the bigger distance is noise.
If the distances are too close, the correct match can't be distinguished from the noise, hence it is ambiguous and should be cancelled.

In his article Lowe points to a ratio of 0.7 or lower between the best two match distances as a threshold for "good" match.

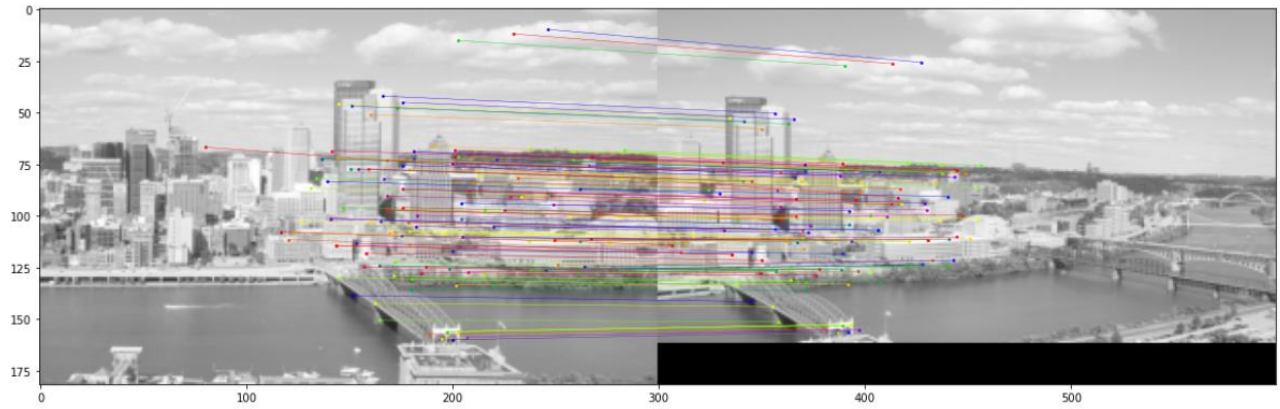
```
# Lowe's ratio test
good = []
for m, n in matches:
    if m.distance < 0.7 * n.distance:
        good.append(m)
```

A really good explanation of Lowe's ratio can be found here:
https://stackoverflow.com/questions/51197091/how-does-the-lowes-ratio-test-work
Lowe's article – Distinctive Image Features from Scale-Invariant Keypoints
https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf
A convincing graph from the article:

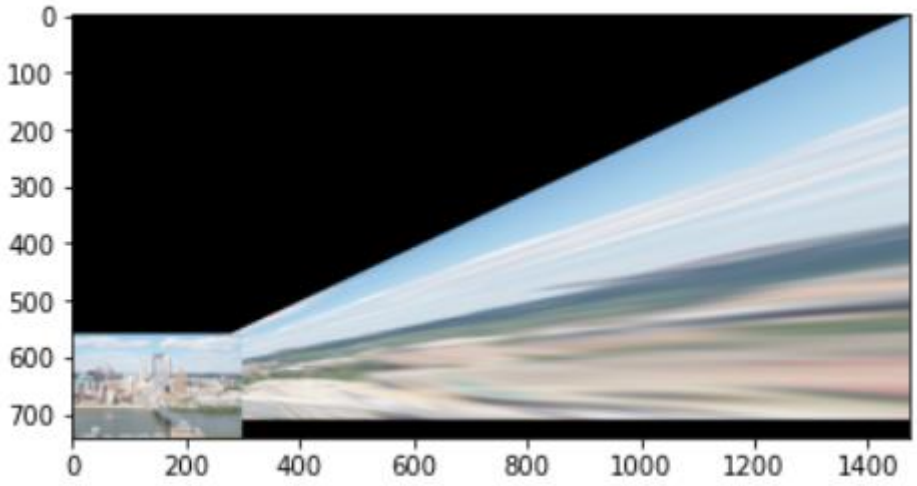


Results for the 3 different methods are shown and briefly discussed below.

2.6.1 Brute-Force KNN matching

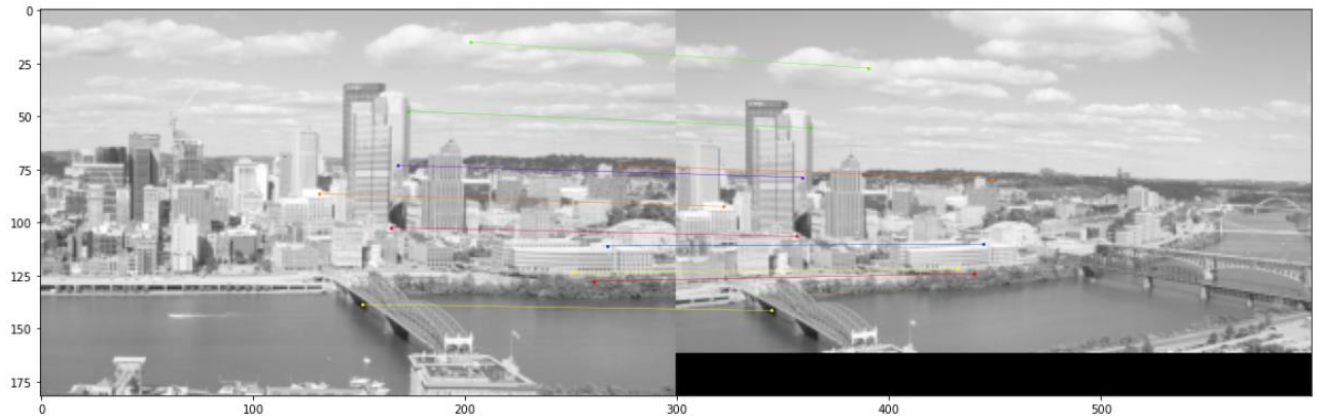


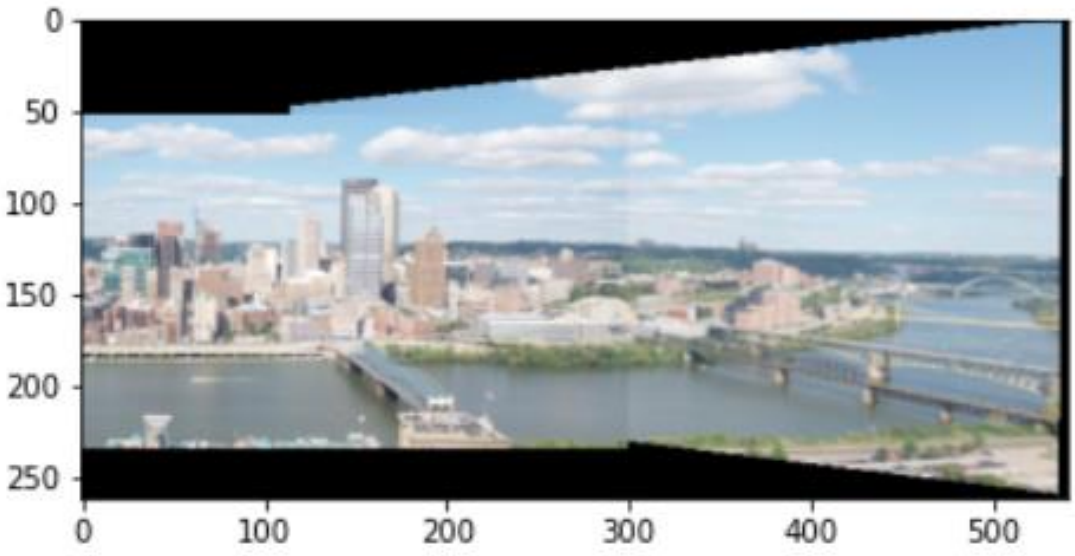
134 matches.



This is a classic outcome of an outlier affecting the homography. An outlier can even be spotted in the naked eye - the most left red dot in the left image.

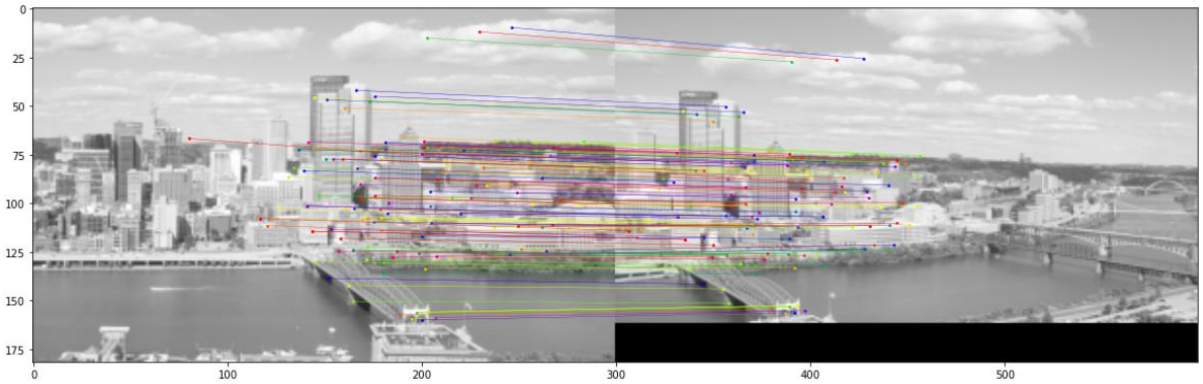
2.6.2 Brute Force – Cross check top 10 matches



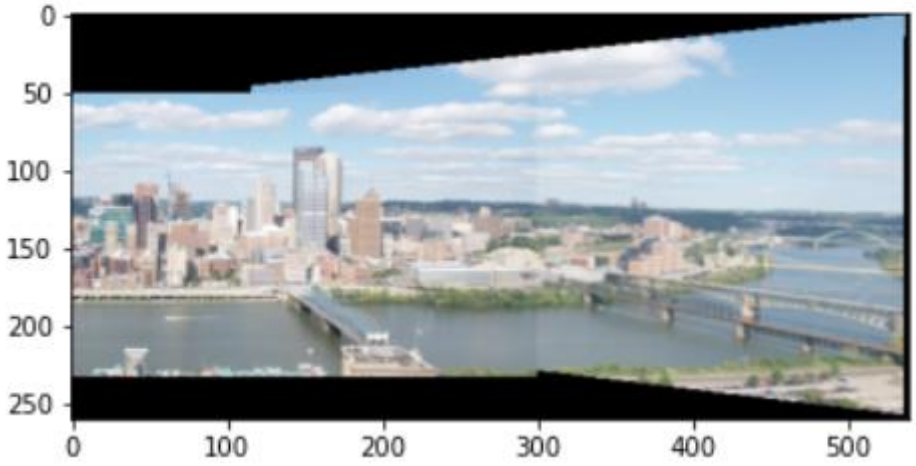


No outliers, no problems.
The risk in taking this method is that all the best matches will be concentrated in a small area, or even worse, on a same line in the source picture.

2.6.3 FLANN with KNN matching



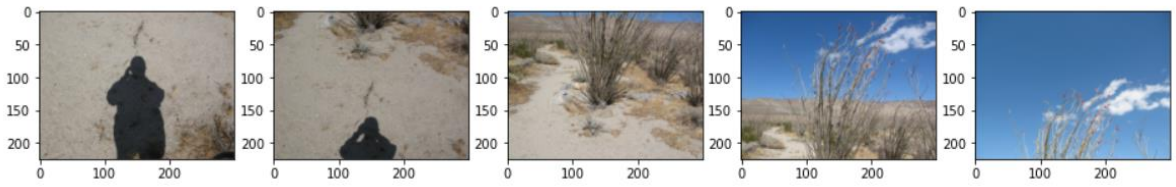
134 matches.

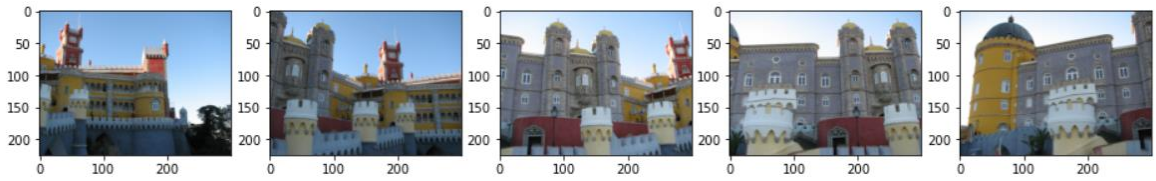


This outcome was surprising as the *Brute Force KNN* method failed.
We labeled it as more luck than brains having FLANN avoid matches by not discovering them.

2.7 Q2.7 – Comparing SIFT to the Manual Image Selection

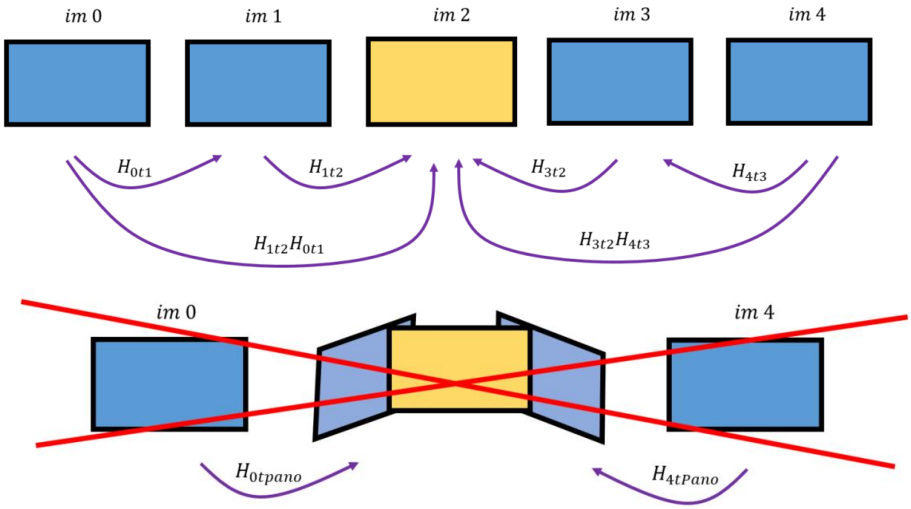
We were to compare the SIFT and Manual methods on the *Beach* and *Pena National Palace* picture we were provided with after resizing them.



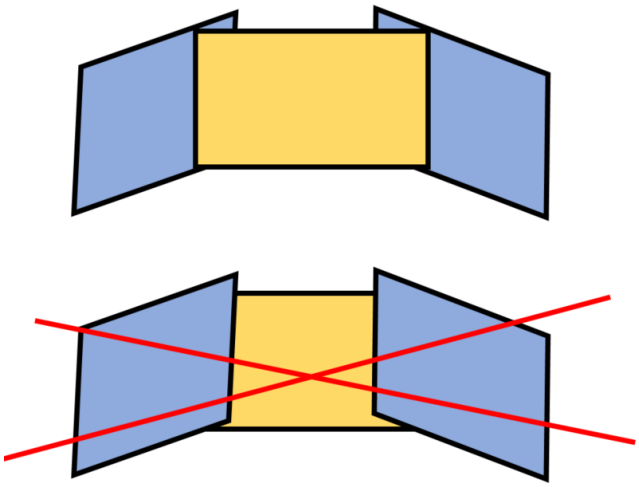


To stitch more than two pictures into a panorama we took the following steps:

1. We assumed the most interesting features lie in the middle of the panorama hence we transformed all pictures to the middle picture.
2. Wishing to avoid feature-matching on large images we decided to apply SIFT only on neighboring original images, and transforming outer images to the middle image coordinate system by multiplying homographies.
   For a detailed explanation of the stitching process for more than 2 images see $Q2.4$



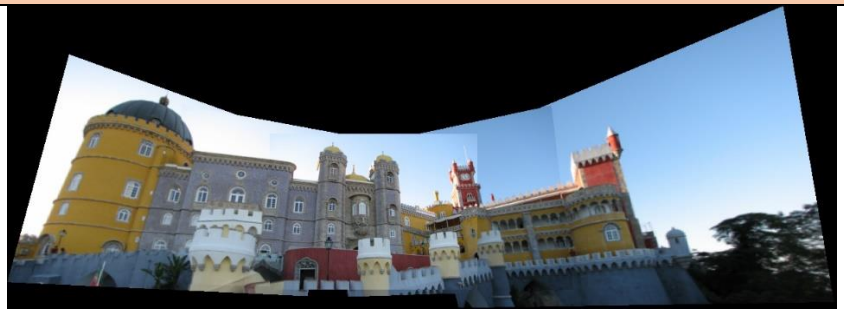
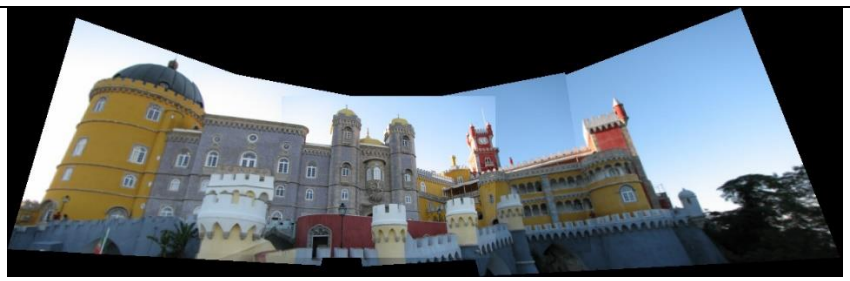
3. As the middle image is assumed to be most significant, we made sure to ‘paste it on top’.



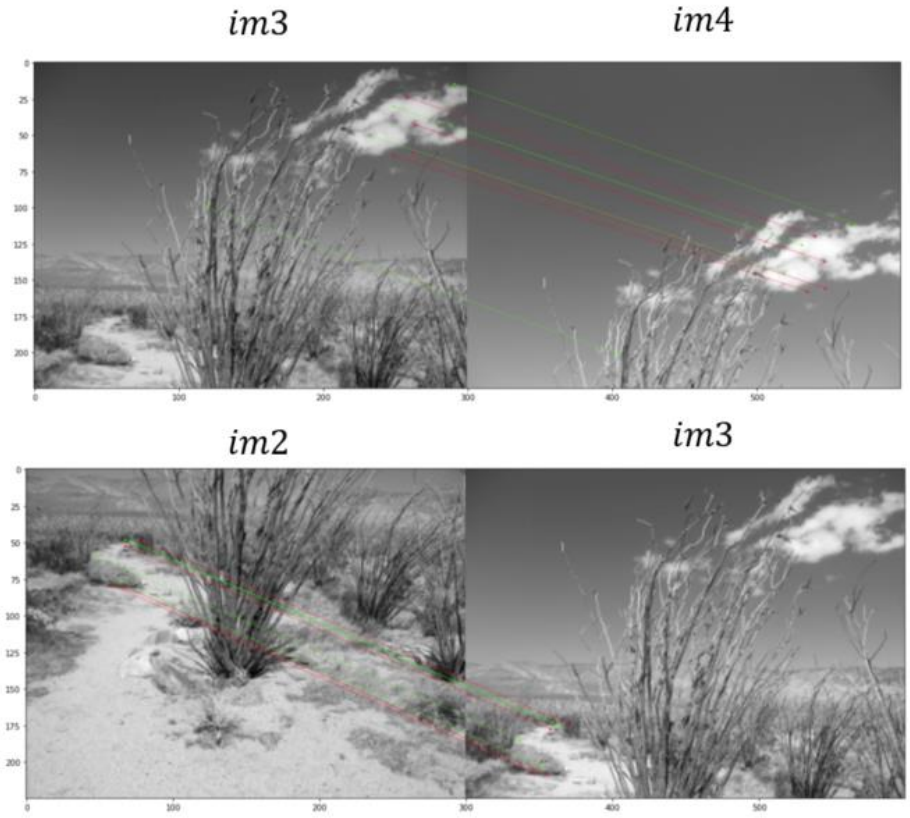
Having the *SIFT* generated homography matrices ‘good enough’ without using RANSAC was a challenge. Still, we made it work:

Beach Pictures

Manual 4 matches for H	SIFT: Brute Force, Cross-check , 9 best matches
	

Pena National Palace Pictures	
Manual 4 matches for H	
SIFT: Brute Force, Cross-check , 50 best matches	

It is rather clear that the *Beach* panorama computed with *SIFT* has had some problems stitching the top most image.
To view why we looked at the two feature-matching products of the relevant pictures $H_{4t2} = H_{3t2}H_{4t3}$. Small convex hulls in the source images relative to its size are defiantly the cause of the problem.



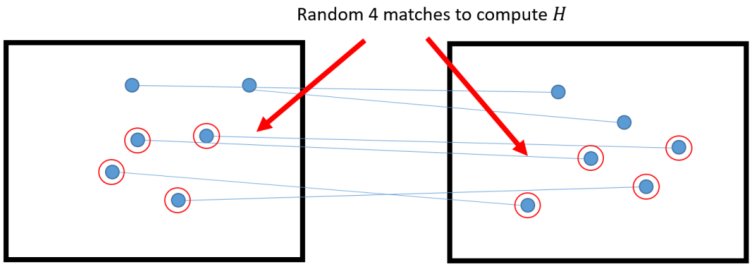
2.8 Q2.8 – RANSAC

RANSAC (Random Sample Consensus) is an iterative method to estimate parameters of a mathematical model, like Least Mean Squares.
It accepts two parameters: number of iterations and tolerance for determining inlier from outlier.
RANSAC is an algorithm that is very indifferent to outliers which is both a positive and a negative as it distances the engineer from the data.
An outlier does not always represent a measurement error – it might represent a real event which can cause real harm and must not be ignored.

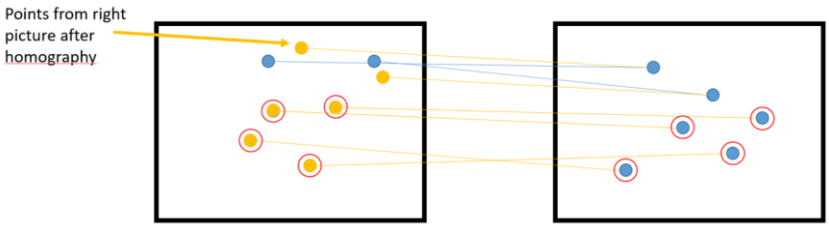
We will demonstrate we to found the homography parameters provided the SIFT matches through this technique:

1. Choose two parameters – iteration amount and tolerance.

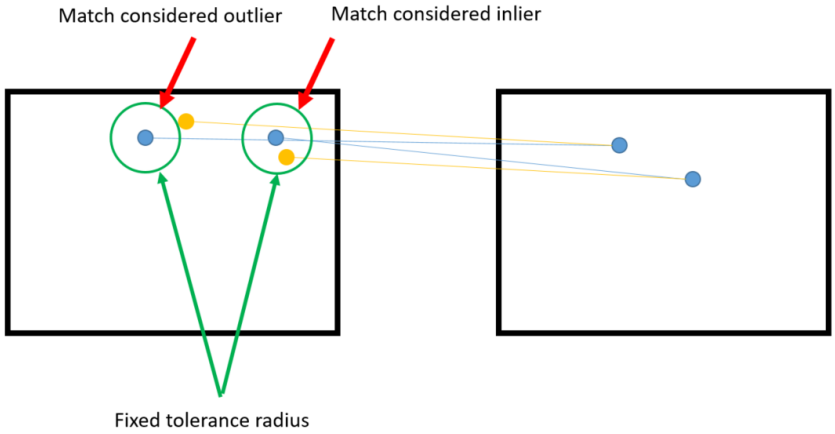
2. Choose 4 random matches and compute a homography



3. Apply the homography on all matches



4. Count the amount of inliers determined by the tolerance:
   We checked the Euclidian distance between the match and the homography projected point.

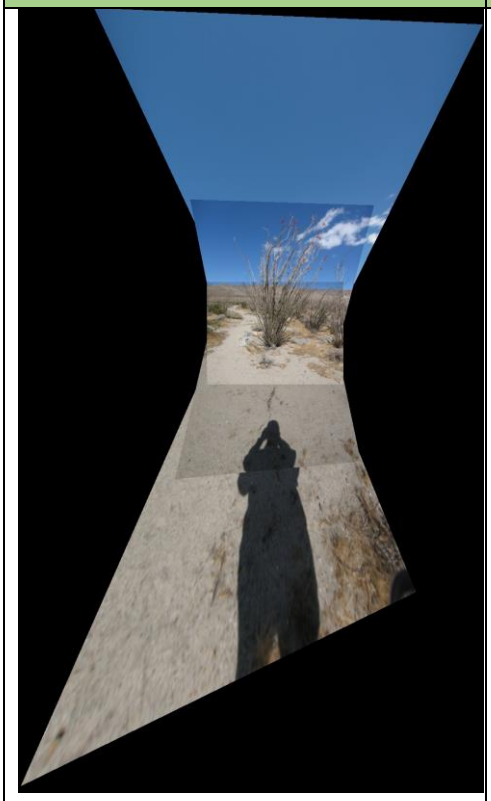


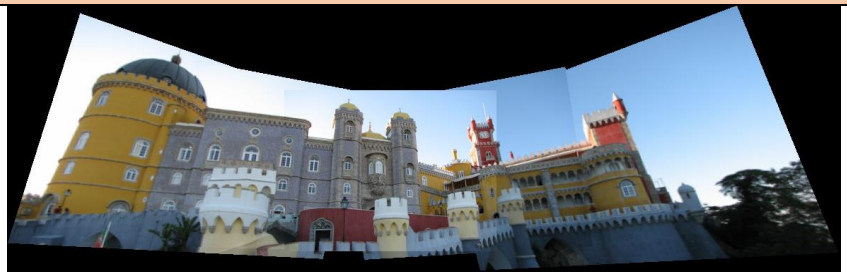
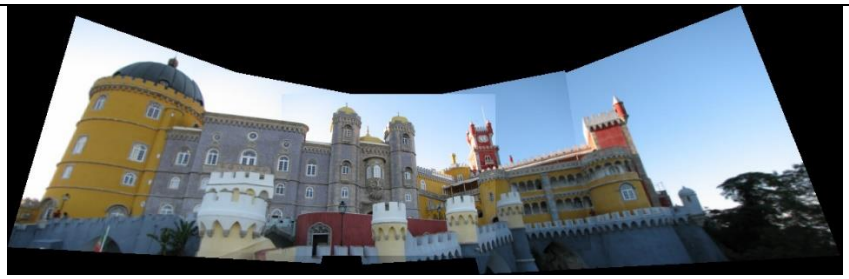
5. Go back to step 2 for another iteration unless iteration amount capped.
6. Chose homography that produces the most inliers.

Note: One could incorporate LMS with RANSAC by choosing more than 4 points on every iteration. Hypothetically this will allow us to reduce the amount of iterations and provide a better model.
Alas, this is a naïve idea with no real advantages when the data has not been properly inspected. We don't plan on making such inspections and thus went for the most robust method: pure RANSAC.

Below is a comparison between SIFT – LMS and SIFT – RANSAC.
We provide no comparison for the manual method as we chose 4 points for every manual homography with good results.

Beach Pictures (see $Q2.7$ for orginials)

SIFT: RANSAC Siftmethod – flann + KNN Iteations – 1000 Tolerance - 1	SIFT: LMS Siftmethod - Brute Force + Cross-check , 9 best
	

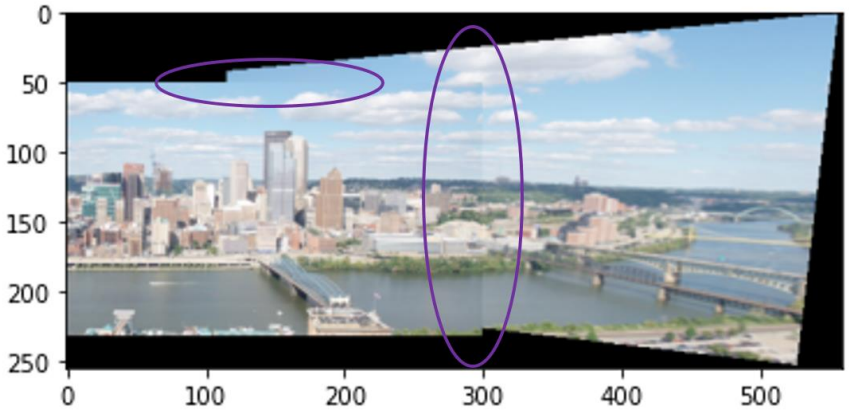
Pena National Palace Pictures (see $Q2.7$ for orginials)	
SIFT: RANSAC Siftmethod – flann + KNN Iteations – 1000 Tolerance - 1	
SIFT: LMS Siftmethod - Brute Force + Cross-check , 50 best	

Just by viewing the results in the tables the advantage isn't quite clear (results are really good).
Adding RANSAC makes the automization process robust. That is to say we weren't required to tune matching method!

About the parameter choice:
The RANASC algorithm is rather quick in comparison to the other algorithms in the program, and SIFT outputs really good matches - for those reasons we felt free to take a big iteration number (amount of matches is <100) and a small tolerance.

2.9 Q2.9 – Blending

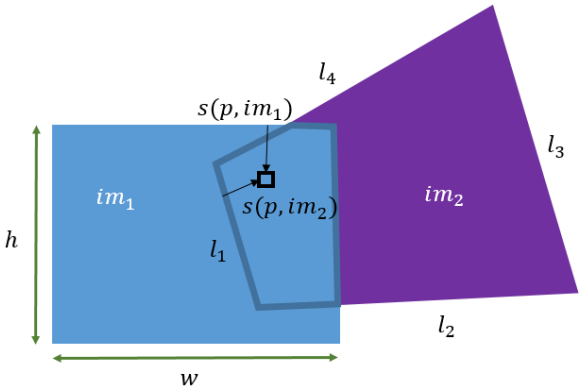
The purpose of Blending is to rid the panorama of edges created by difference in luminance



We tried two different methods to achieve proper blending and will present them both.
The better method (Poisson blending) was continued with.

2.9.1 Distance in contested area

We proposed a method where pixels in the contested area (where the two images overlap) will be evaluated by their distance from image edges.
The intuition is that the closer the pixel is to an image's edge, the less significant that image should be for said pixel.



$s(p, im) = minima\ length\ between\ pixel\ and\ edges$

$p = [i = rows, j = columns]$

$$s(p, im_1) = \min(p[0], p[1], h - p[0], w - p[1])$$

$$s(p, im_2) = \min\big(d(l_1, p), d(l_2, p), d(l_3, p), d(l_4, p)\big)$$

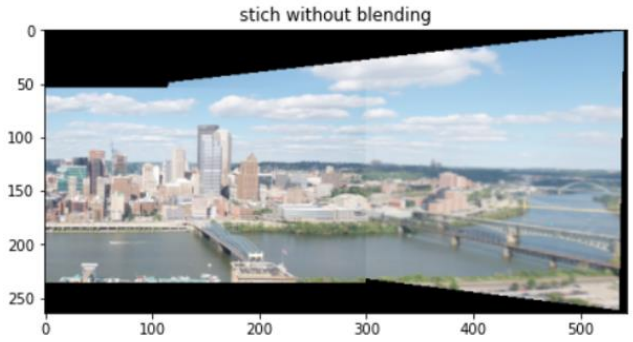
$$alpha_{im_1} = \frac{s(p, im_1)}{s(p, im_2) + s(p, im_1)}$$

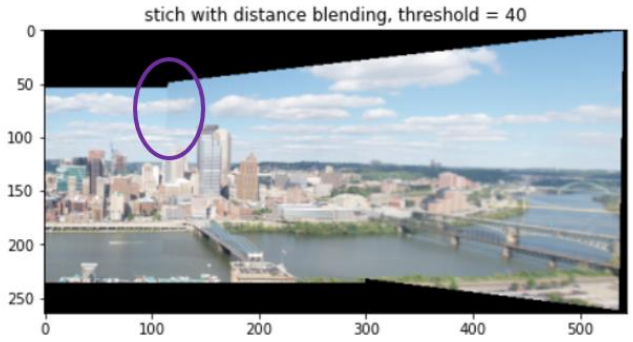
$$p \coloneqq alpha_{im_1} \cdot im_1(p) + \big(1 - alpha_{im_1}\big) im_2(p)$$

We added a threshold value to ensure we have control on how much area is considered 'contested':

$$if\ s(p, im_1) > threshold:$$
$$p[i, j] = im_1(p)$$

This resulted in the following:



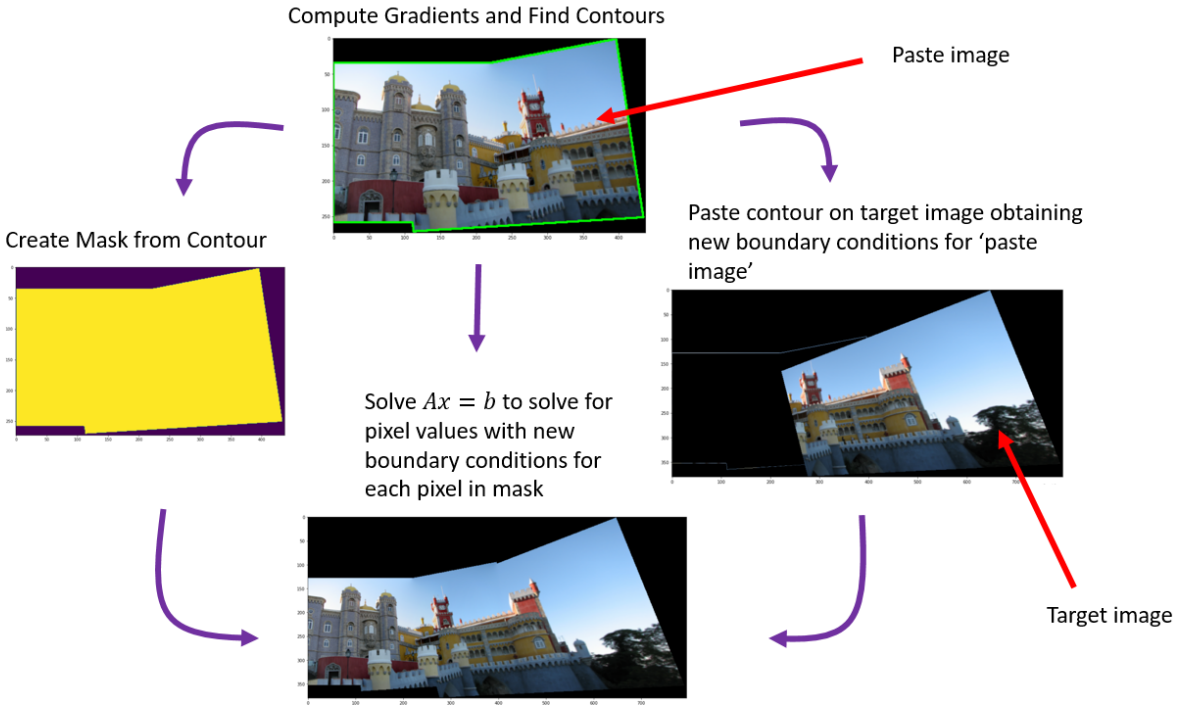


The blending worked rather well for the big stich line on the right, but created a stitch line on the left.
We weren't pleased and went to implement Poisson blending.

2.9.2 Poisson Blending

We first disclose the sources we used for Poisson blending. They do a better job in explaining the process then we could ever do.
http://cs.brown.edu/courses/cs129/results/proj2/taox/
https://erkaman.github.io/posts/poisson_blending.html
https://piazza.com/class_profile/get_resource/hz5ykuetdmr53k/i0zbj9rijcs7m7

The core idea of the method is that from boundary conditions and gradients one could completely restore an image.
Changing the boundary conditions but keeping the gradients will allow blending.
The beauty of this method is that it does this by solving a sparse linear system of equations!



A is a matrix of coefficients of size $N \times N$ where N is the amount of pixels in the mask image

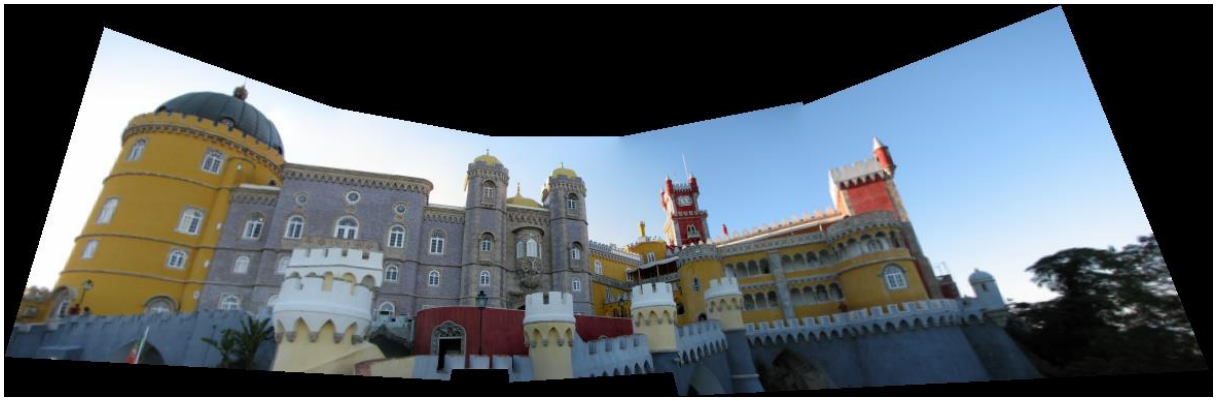
x is a vector containing the 'paste image' pixel values after blending

b is a vector containing the desired gradient information.

For pixels inside the masked region, the output image pixel x at (row, col) depends on its neighbors such that each row in the equation takes form:

$$4x(row, col) - x(row+1, col) - x(row-1, col) - x(row, col+1) - x(row, col-1) = desired\ pixel\ gradient$$

Results:

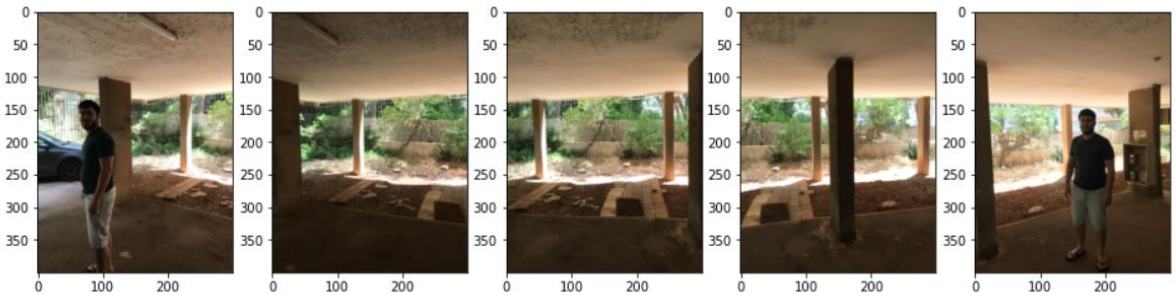


Beauties!!

2.10 Q2.10 - Be creative

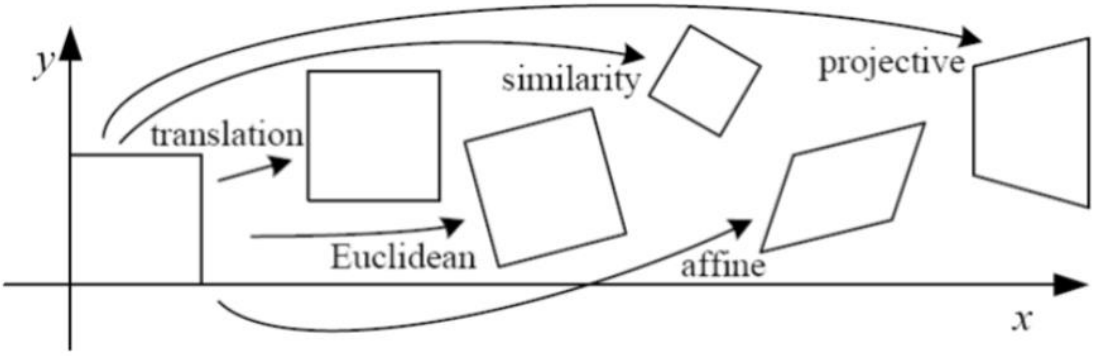
We thought it would be fun to have an image or where paradoxically one of us shows up twice. It also shows the strength in choice of multiplying homographies instead of stacking panoramas (see $Q2.7$).

We created two panoramas from the images:
One where the center picture is in-front, and one where it is in back (again, see $Q2.7$)



2.11 Q2.11 – Affine vs Projective

An Affine transformation has 2 less degrees of freedom than a homography which translate to the constraint of keeping parallel lines parallel.
In the following section we compute, test and compare an affine transformation to a homography transformation when dealing with panoramas.



2.11.1 Solving for affine projection

Given a set of homogenous points p_i in an image taken by camera C_1, and corresponding homogenous points q_i in an image taken by C_2, suppose there exists an affine transformation H such that:

$$p_i = Hq_i$$

Given N correspondence we derived a set of $2N$ independent linear equations in the form $Ah = b$

We denote: $H = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ 0 & 0 & 1 \end{bmatrix}$, $p_i = [x_i, y_i, 1]$, and $q_i = [u_i, v_i, 1]$

$$\begin{bmatrix} [x_1, y_1, 1] \\ \vdots \\ [x_N, y_N, 1] \end{bmatrix}^T = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} [u_1, v_1, 1] \\ \vdots \\ [u, v_N, 1] \end{bmatrix}^T$$

Solving for $i = 1$:

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix}$$

Opening up the terms:

$$\begin{cases} u_1 h_1 + v_1 h_2 + h_3 = x_1 \\ u_1 h_4 + v_1 h_5 + h_6 = y_1 \end{cases}$$

Each pair of matching points i will provide two equations. Hence, the lot of points will provide the $2N$ equations we were asked to find.

For $i = 1 \dots N$:

$$\begin{cases} u_i h_1 + v_i h_2 + h_3 = x_i \\ u_i h_4 + v_i h_5 + h_6 = y_i \end{cases}$$

Writing the $i'th$ pair of equations in matrix form:

$$\begin{bmatrix} u_i & v_i & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & u_i & v_i & 1 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

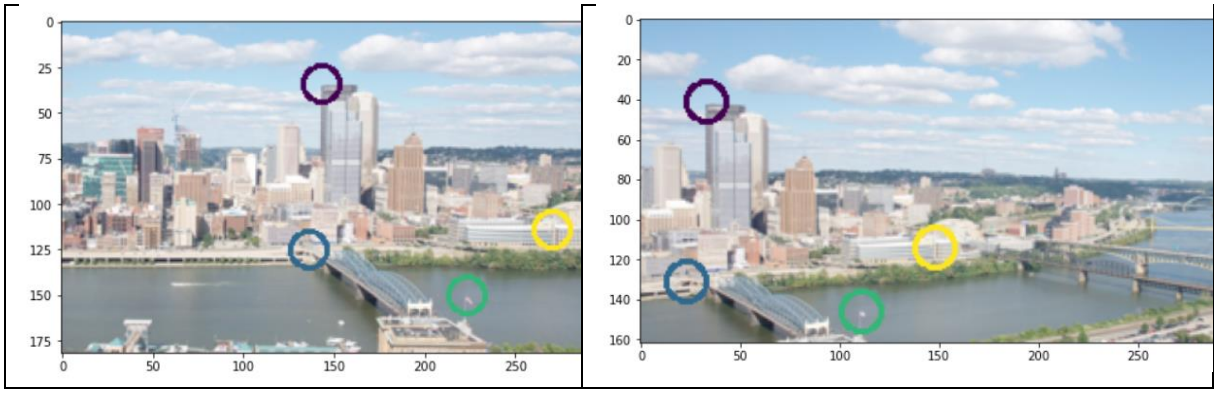
Solving for the vector h requires a minimum of 6 equations hence a minimum of 3 points.

We solved the LMS problem using pseudo inverse: *h=np.linalg.pinv(A)@b*.

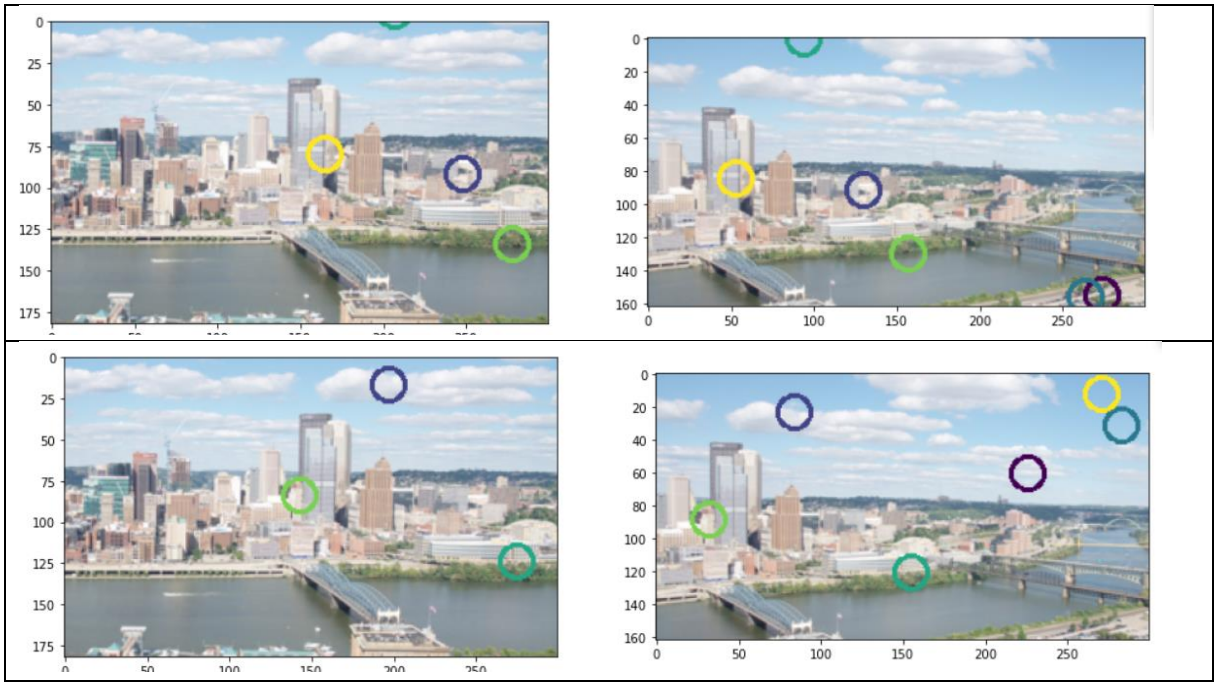
2.11.2 Testing the Affine transform

The affine transformation should hold well for planar objects. Objects that are far away from the camera can be considered as planar. Hence, we decided on testing our affine transformation against the *Incline* images

Manual Selected points:



Random points testing:



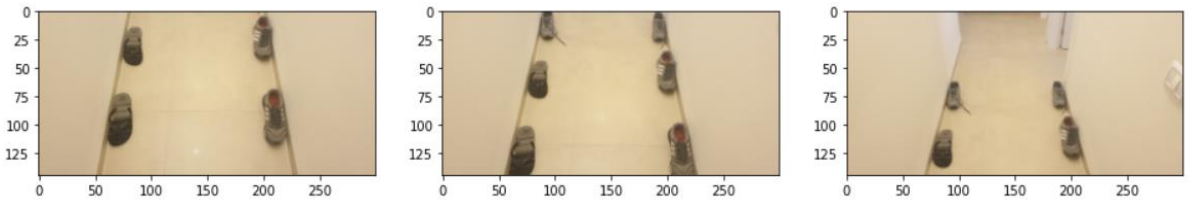
Seems to work rather well!

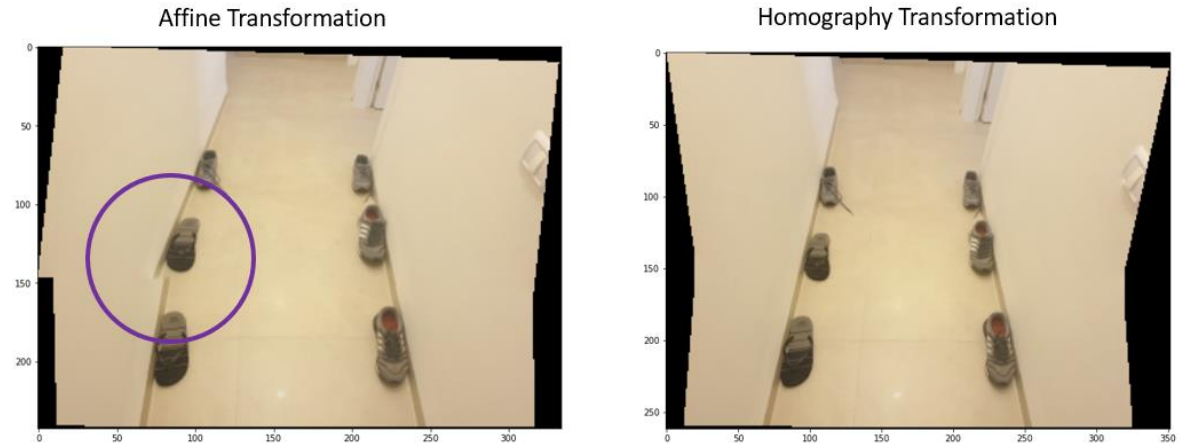
2.11.3 Affine vs Homography

Affine transformation will keep parallel lines parallel.

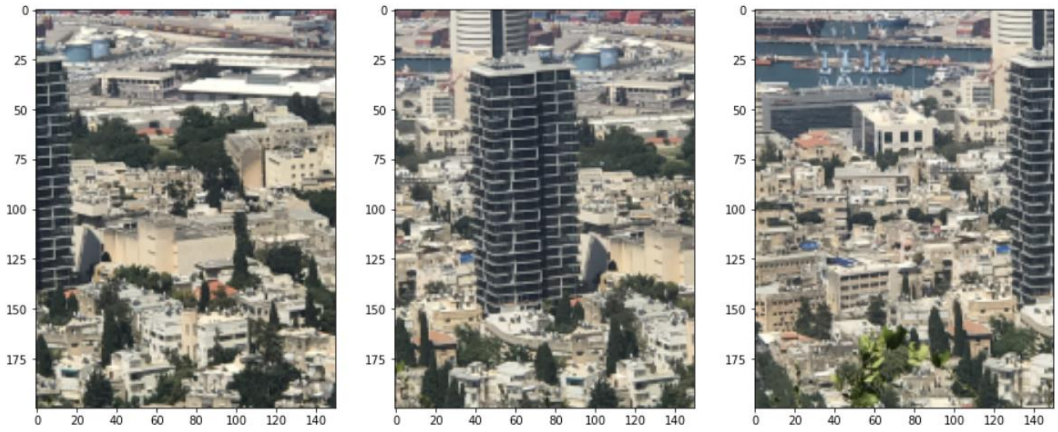
The nature of a road building towards the distance is that the lines start parallel when looking downwards, but end up converging at the horizon. An affine transformation won't be able to handle this situation.

We simulated the road experiment in a corridor. Shoes for feature markings.





An affine transformation will work well on scenery images as the objects are far off from the camera and could be considered planar. One could argue that for such a case it is just better to use an affine transformation.



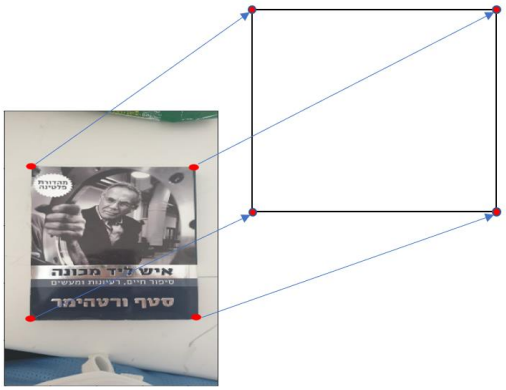
Affine Transformation

Homography Transformation

3 Creating your Augmented Reality application

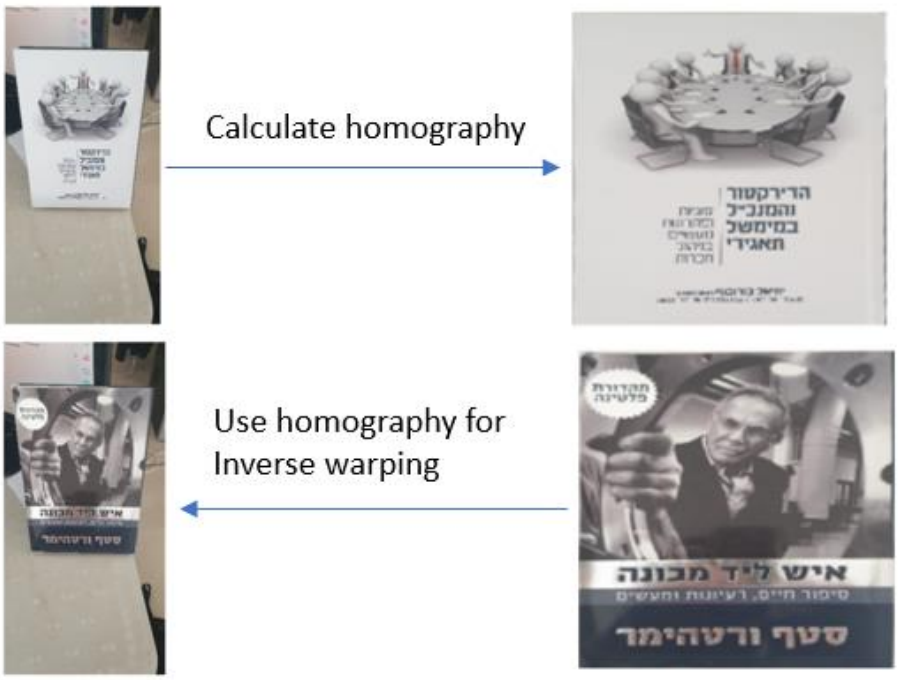
3.1 Create reference model

To create the reference model, we choose rectangular template (four pixels in rectangular shape) and warp the edges of the book as correspond points on the warped image. Points on the book were chosen manually.

Book image	transformation	Reference image
		

3.2 Implant an image inside another image

To implant new image on the book cover we make two correspond reference images. First, we calculate the correspond warp between the reference image and the plant scene. Next, we warp the second book reference image and warp him to the plant scene.



3.3 Video time

To warp the video onto the book cover, we first converted both videos to an image sequence.

One reference image was not good enough for all the videos (probably because of the flexible cover of the book and the color changes through the video), so to make the video precise every ten images we obtain a new reference.

A few illustration frames

The difference between 3.3 is first of all the method of matching from sift to ORB (orientation fast and rotated brief). The full explanation for the ORB matching method is given by the following link.

https://medium.com/data-breach/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf#:~:text=The%20feature%20is%20defined%20as,the%20orientation%20of%20the%20keypoints.

Although the SIFT matching algorithm perform better performance, the time of computation for each image was unacceptable, so we choose the ORB as reasonable exchange.

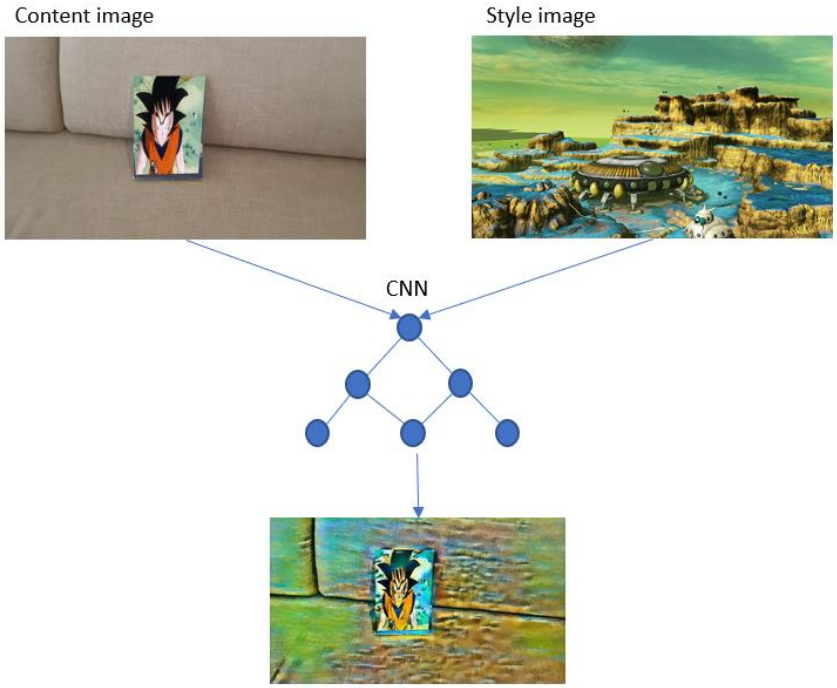
https://www.youtube.com/watch?v=6Re51HnoozA&feature=youtu.be

3.4 Creativity Section

We used a Neural-Style algorithm in this section.
In general the algorithm changes the “style” of the image (in our scope video) by minimizing the distances between the features of the “content image” and the style of the “style image”. To implement the CNN we use (with some modifications) the code in the following link.

https://pytorch.org/tutorials/advanced/neural_style_tutorial.html

Note:
The original implementation weights the style image rather heavily. We reduced the weight so that the DBZ video will be recognizable (style_weight=1,000,000→ $10{,}000$).



A few illustration frames

https://www.youtube.com/watch?v=XTCk3lW3Crg&feature=youtu.be