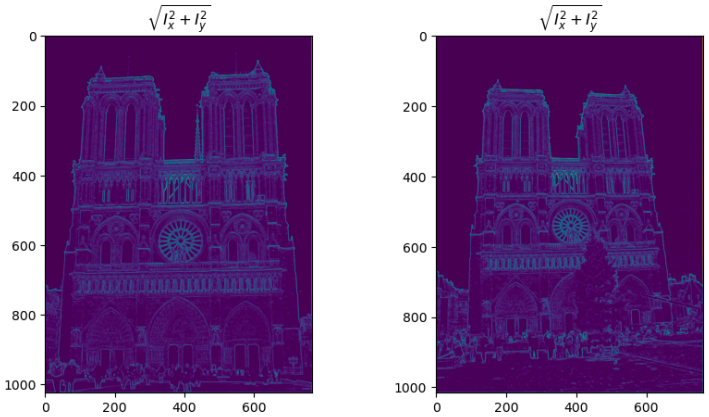
Warning: Do not delete slides.
This includes extra credit slides and any problems you do not complete.
All problems, including extra credit, must be assigned to a slide on Gradescope. Failure to follow this will result in a penalty.

CS 6476 Project 2

Venkata Aashutosh Aripirala
av84@gatech.edu
av84
904173725

Part 1: Harris corner detector

[insert visualization of \sqrt(I_x^2 + I_y^2) for Notre Dame image pair from proj2.ipynb here]



[Briefly explain what this value represents and why we need it for the Harris corner detector.]

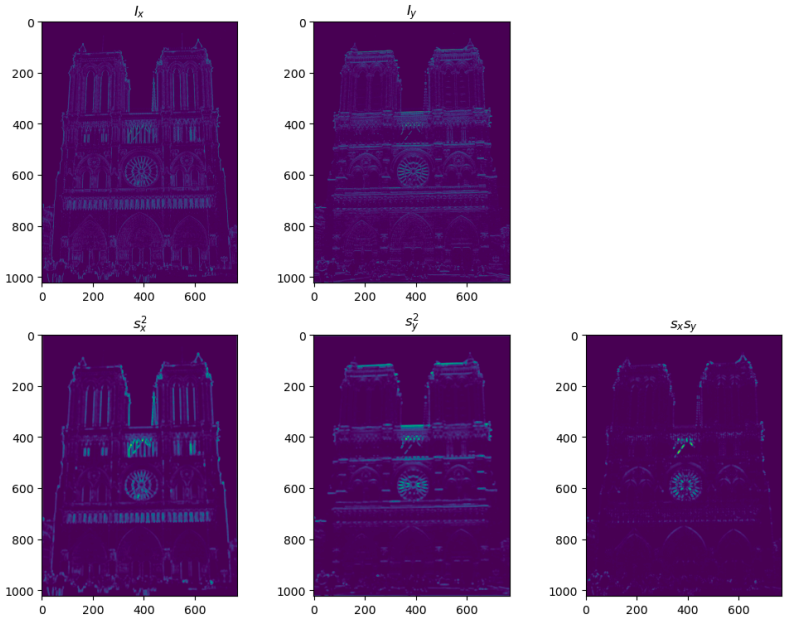
\sqrt(I_x^2 + I_y^2) shows the magnitude of the gradients at a pixel. It depicts the strength of local intensity change, and the magnitude of the gradient vector.

[Which areas have highest magnitude? Why?]

Corners have the highest magnitude of the \sqrt(I_x^2 + I_y^2). This is because corners have gradient changes in all directions, unlike edges, which have a flat gradient in one direction, therefore have the magnitude only in one direction, or a flat region, which has close to 0 gradients. In context of the image, the flat regions that have small magnitudes are regions like the sky etc. which don't have inherent texture, as we move across pixels in the neighborhood.

Part 1: Harris corner detector

[insert visualization of I_x, I_y, s_x^2, s_y^2, s_xs_y for Notre Dame image pair from proj2.ipynb here]



[Briefly explain what these values represent and why we need them for the Harris corner detector.]

Sx^2, sy^2 and sxsy are the second moment matrix values. Sx^2 represents the square of the horizontal gradient smoothed by a gaussian filter, sy^2 the square of the vertical gradient(smoothed by gaussian), and sxsy the product of the 2. They build the Harris matrix, who's eigen values determine the strength of cornerness of the pixel(both small – flat region, both large – corner, one small, one large – edge.

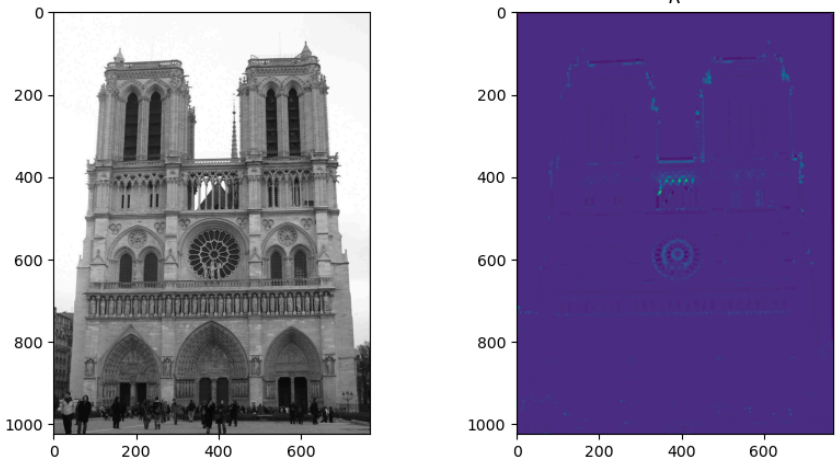
Part 1: Harris corner detector

[Why do we convolve the image with a Gaussian filter?]

We convolve the image with a Gaussian Filter because the Harris detector’s goal is to analyse regions(patches) of the image, not individual pixels. By the convolution, we’re essentially blending information of the neighborhood into the pixel(since it acts like a weighted average). We also want to improve robustness to noise. A single noisy pixel can produce a strong gradient, and smoothening it with a Gaussian ensures that the entries going into the Harris Matrix are stable, and represent the underlying structure of the image, rather than being skewed by random noise of intensities.

Part 1: Harris corner detector

[insert visualization of corner response map of Notre Dame image from proj2.ipynb here]



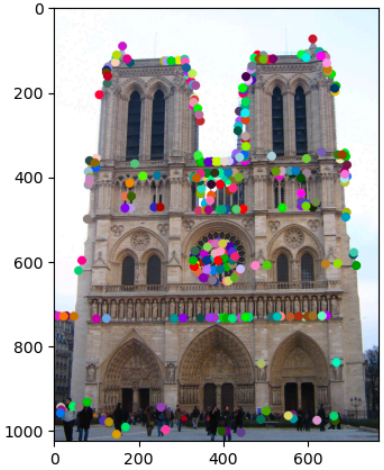
[Are gradient features invariant to both additive shifts (brightness) and multiplicative gain (contrast)? Why or why not? See Szeliski Figure 3.2]

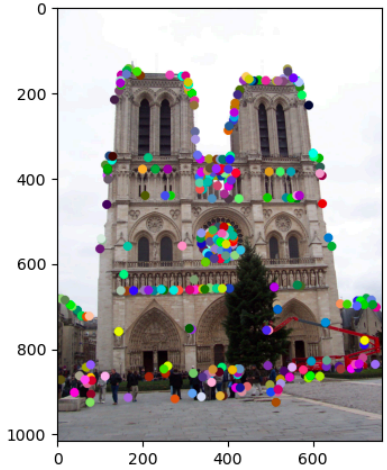
Gradient Features are invariant to additive shifts, since adding a constant c to I doesn't change dI/dx, since d(I+c)/dx = dI/dx + dc/dx = dI/dx + 0.

Multiplicative Gain multiplies each pixel by a constant factor "k". I => kI
d(kI)/dx = k * dI/dx. Here the gradient is also scaled by the multiplicative gain.

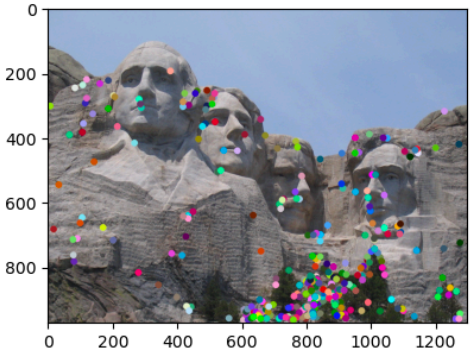
Part 1: Harris corner detector

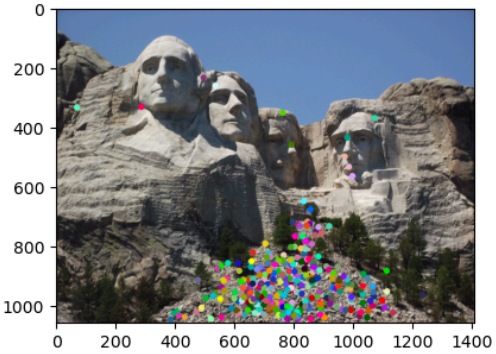
[insert visualization of Notre Dame interest points from proj2.ipynb here]





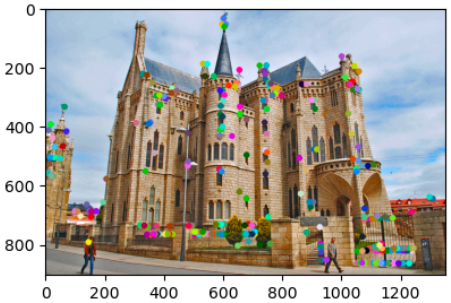
[insert visualization of Mt. Rushmore interest points from proj2.ipynb here]

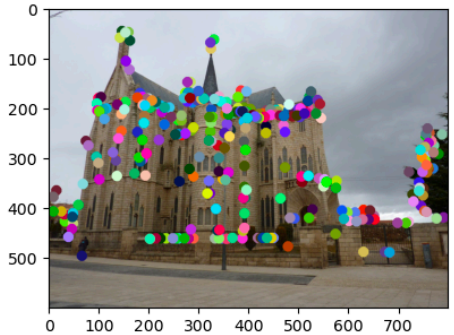




Part 1: Harris corner detector

[insert visualization of Gaudi interest points from proj2.ipynb here]





[What are the advantages and disadvantages of using maxpooling for non-maximum suppression (NMS)?]

Using maxpooling for NMS offers the primary advantage of efficiency, as it is highly optimizable and parallelizable, well suited for GPU operations. Its key disadvantage is imprecision, as it only finds local maxima in the pixel grid, and performance is subjective to kernel size. It also handles plateaus of identical values poorly by keeping all of them. This can lead to multiple detections of a single feature. It trades higher-precision and robustness for significant improvement in speed.

Part 1: Harris corner detector

[Explain the intuition behind the Harris corner score equation $R = \det(A) - \alpha \cdot \text{trace}(A)^2$]

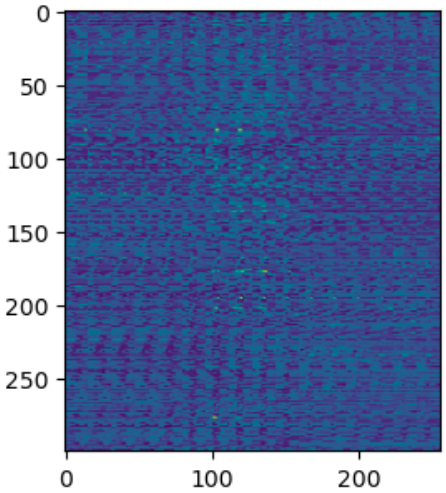
The determinant of a 2x2 matrix is the product of its eigen values, and the trace is the sum of eigen values. This Cornerness score allows for calculation of the strength of the corner by using these aspects of the eigen values. This R value will be large for corners only(where both eigen values are large), small for edges(a single small eigen value reduces the determinant value), and small for flat regions(product of both small eigen values minimizes the det(A) value further, leading to a close-to-zero / negative cornerness score.

[What is your intuition behind what makes the Harris corner detector effective?]

The intuition behind the Harris Corner detector is that the corners are information dense, as they have strong 2 dimensional structure. Unlike an edge, which is information-dense only in one direction(perpendicular to the edge) or flat regions(which aren't informative), a corner possesses strong intensity change in multiple directions, an idea used by the Harris Corner detector for finding features. This allows us to map corners as regions of importance, for tracking, stitching, and recognition.

Part 2: Normalized patch feature descriptor

[insert visualization of normalized patch descriptor from proj2.ipynb here]

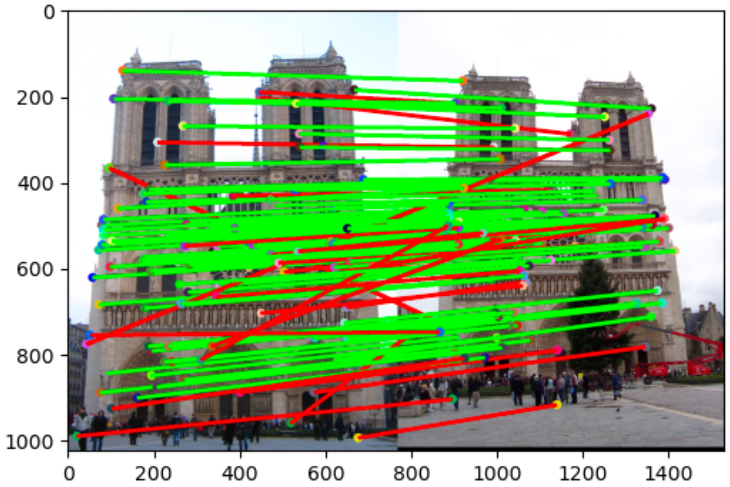


[Why aren't normalized patches a very good descriptor?]

While easy to compute, normalized patch descriptors are sensitive to geometric and illumination changes. Its very fragile to small shifts / rotations. This causes an issue with matching patches which are slightly rotated, as the Euclidean distance between the vector is increased in magnitude almost across all dimensions. Also, normalizing the patch can handle global shifts across the whole image, but fails in cases like occurrence of shadows, highlights in a part of the image. The descriptor also encodes raw pixel intensities, unlike SIFT which captures local gradients, which is much more resilient to exact brightness and color that form the feature (say a shape).

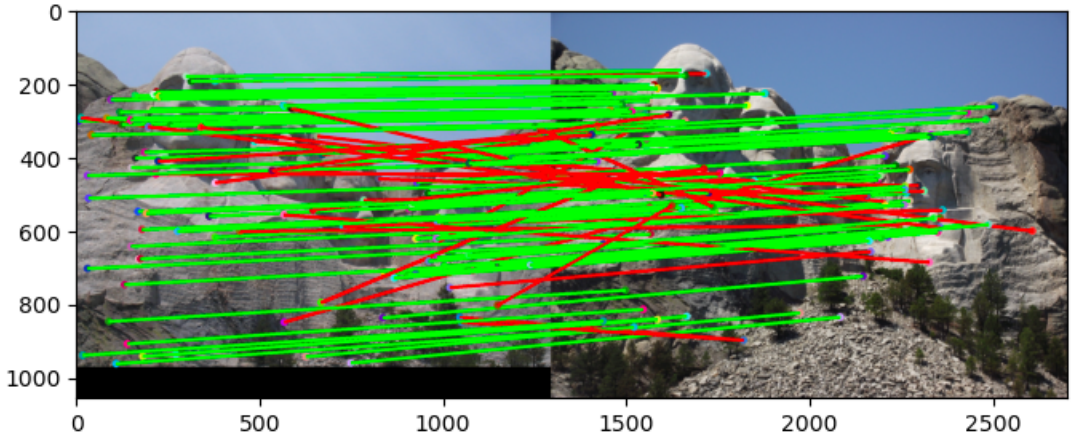
Part 2: Feature matching

[insert visualization of matches (with green/red lines for correct/incorrect correspondences) for Notre Dame image pair from proj2.ipynb here]



matches (out of 100): [110]
Accuracy: [0.754545]

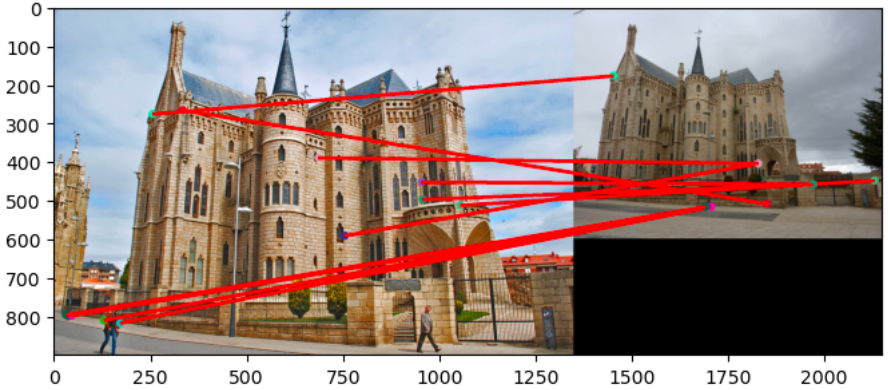
[insert visualization of matches for Mt. Rushmore image pair from proj2.ipynb here]



matches: [104]
Accuracy: [0.740385]

Part 2: Feature matching

[insert visualization of matches for Gaudi image pair from proj2.ipynb here]



matches: [11]
Accuracy: [0.00]

[How does your implementation of *match_features_ratio_test* filter and structure feature matches? What is the significance of the confidence score?]

While the first 2 images (Notre Dam and Mount Rushmore) have decent accuracies, the Gaudi Image fails in the Harris Corner Detector method. What we can infer from the confidence score (accuracy) is that the Harris Corner Detector isn't scale-invariant. The first 2 images work decently because there isn't much of a difference in the view-point and scale in them. The low accuracy (0.00) in the Gaudi image corresponds to what is a significant difference in camera zoom, size of features, etc.

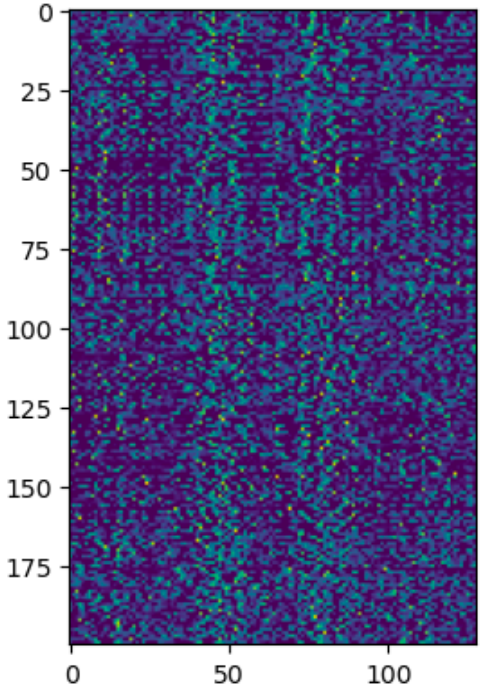
Part 2: Feature matching

[Why do feature matches that easily pass the nearest neighbor distance ratio (NNDR) test tend to be more accurate?]

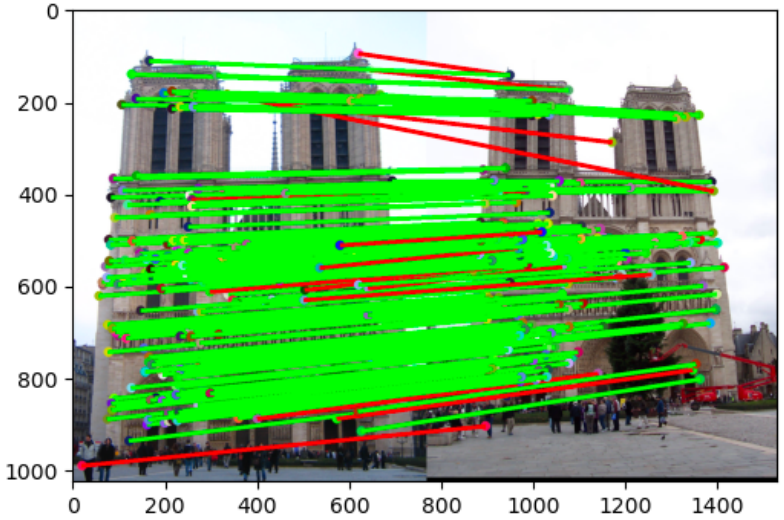
Feature Matches that easily pass the NNDR test tend to be more accurate because the test is a good measure of the feature's uniqueness and distinctiveness. If the ratio is very low, it implies the next closest match is much further away, allowing us to finalise a match without much contention from the next neighbors. By acting as a confidence filter, NNDR discards ambiguous matches, while keeping only the most reliable correspondences, which drastically increases the overall accuracy of the final set of matches.

Part 3: SIFT feature descriptor

[insert visualization of SIFT feature descriptor from proj2.ipynb here]



[insert visualization of matches (with green/red lines for correct/incorrect correspondences) for Notre Dame image pair from proj2.ipynb here]

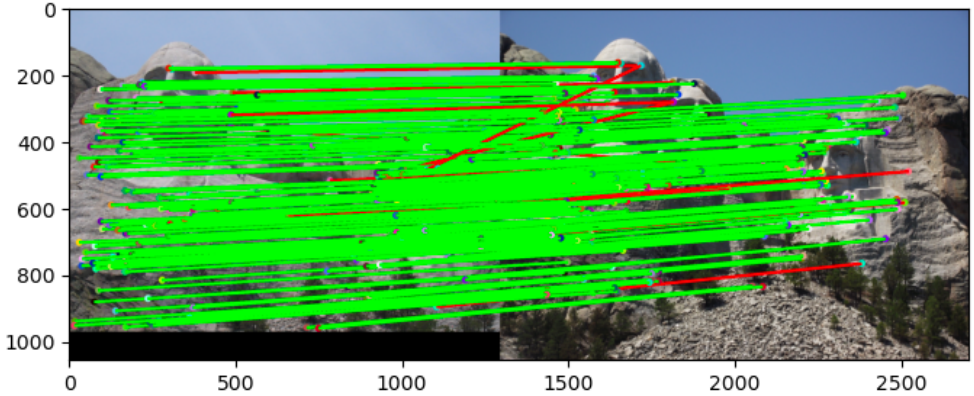


matches (out of 100): [191]

Accuracy: [0.926702]

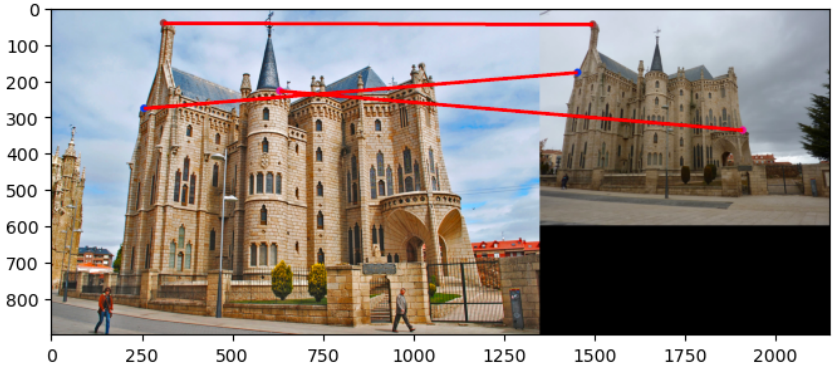
Part 3: SIFT feature descriptor

[insert visualization of matches for Mt. Rushmore image pair from proj2.ipynb here]



matches: [182]
Accuracy: [0.934066]

[insert visualization of matches for Gaudi image pair from proj2.ipynb here]



matches: [3]
Accuracy: [0.00]

Part 3: SIFT feature descriptor

[Explain how gradient histograms are constructed in the SIFT descriptor. What role do they play in the overall process?]

In a SIFT descriptor, gradient histograms are constructed taking 4x4 grids of 16x16 patches. Within each cell of the 4x4 grid an 8-bin orientation histogram is created. Each pixel in this weighted histogram contributes its gradient magnitude amount of weight to an orientation bin. This allows for the descriptor to be robust to absolute brightness etc. but instead describe the underlying shape, like edges, corners and textures within the patch. These 16 histograms are concatenated to form a 128-dim descriptor feature vector which encodes the local shape information.

[Why are SIFT features better descriptors than the normalized patches?]
SIFT features are illumination invariant, and they're primarily based on Image Gradients. They depend on orientation of edges and textures, and therefore are stable under significant lighting variations which adversely affect normalized patches. The SIFT descriptor is also more robust to small shifts, rotations, and view-point changes. A slight deformation would still lead to a similar descriptor whereas a normalized patch is extremely brittle, and a single pixel shift can lead to massive changes in the feature vector.

[What is the advantage of Square-Root SIFT?] (1.5)
Square-Root SIFT, which involves taking element-wise square root of the L2-normalized SIFT vector, allows for closer approximation of the distance between the 2 vectors to the Hellinger Kernel, a metric which is known to be robust in comparison of histograms. It is better at handling busty features, where a few histograms have high values. The vector is also constructed at no extra computational cost, making it a better choice.

Part 3: SIFT feature descriptor

[Why does our SIFT implementation perform worse on the given Gaudi image pair than the Notre Dame image and Mt. Rushmore pairs?]

Our SIFT implementation performs MUCH worse on the Gaudi image due to SIGNIFICANT view-point change, leading to projective distortion. With such a massive view-point change, we've essentially hit the threshold of the limits of SIFT's robustness. The Notre Dame image has only a minor translational shift, while the Mount Rushmore image has very minor scaling and view-point changes. With the second image of Gaudi being a highly oblique side-on view of the monument, the local shapes are warped in a non-uniform way. While the SIFT descriptor is robust to uniform scaling and rotation, its not invariant to shearing, non-uniform stretching and aggressive perspective change. The local gradients are fundamentally altered, leading to significant changes in the feature descriptor, resulting in a large Euclidean distance.

Part 3: SIFT feature descriptor

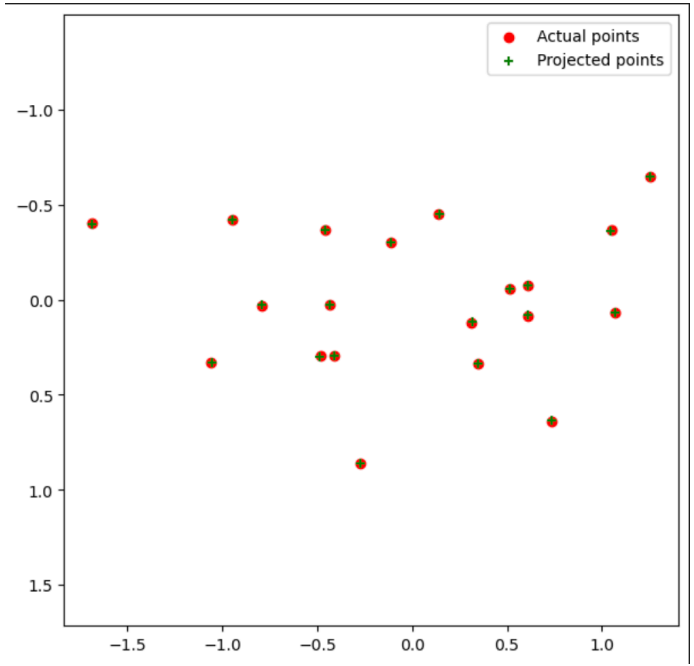
[Why aren't our version of SIFT features rotation- or scale-invariant? What would you have to do to make them so?]

In our get_feat_vec function, the 16 x 16 patch is always aligned with the image's horizontal and vertical axis. If the object rotates between the axes, the gradients will also rotate. The lack of the same results in our model not being rotation-invariant. To improve this we should make a histogram of gradient orientations in a circular neighborhood around the keypoint pixel. We must identify the peak of the histogram, which represents the dominant direction of the gradient. We must then rotate the descriptor patch and its gradients to align with this direction.

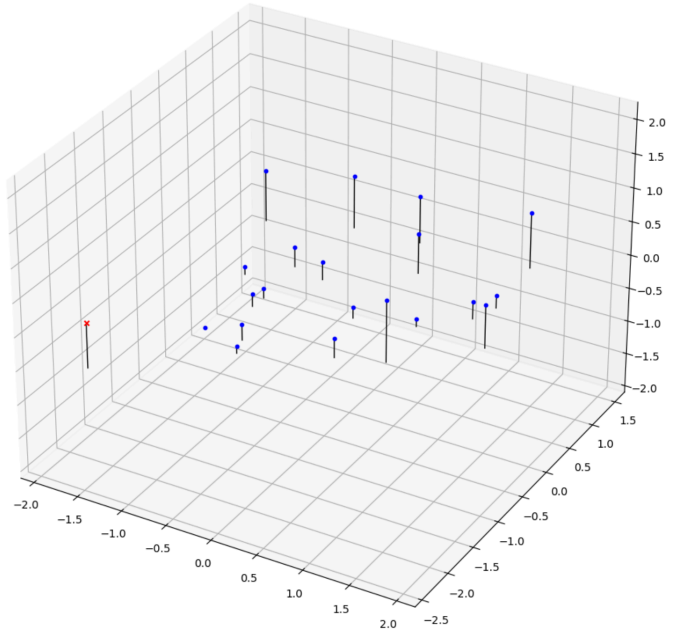
In the pipeline we also use a single-scale Harris Corner detector, and extract a fixed-size 16x16 patch. This makes so that our model isn't scale-invariant. To make it scale-invariant, we should implement a scale-space pyramid, experimenting with different scales and assigning a characteristic scale to each feature. We should identify keypoints that're stable local extrema, using the Difference of Gaussians approach. Finally, we should scale the size of the window relative to the characteristic scale we found.

Part 4 (Extra Credit): : Projection matrix

[insert visualization of projected 3D points and actual 2D points for the first CCB image we provided here]

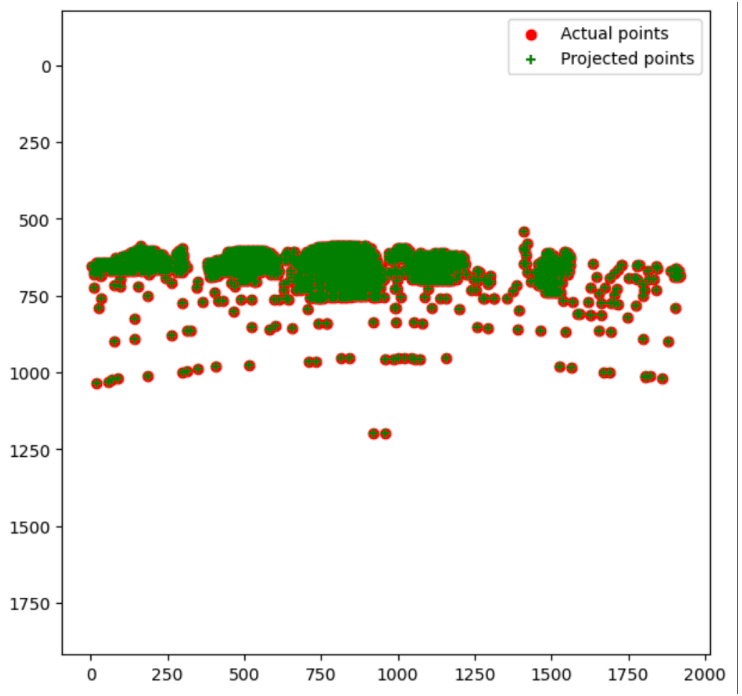


[insert visualization of camera center for the first CCB image here]

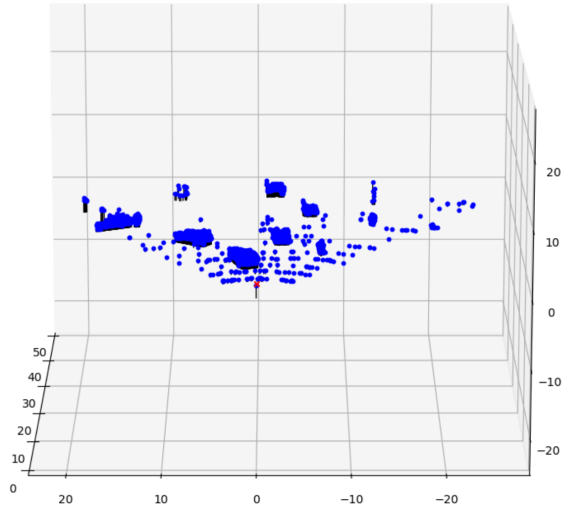


Part 4 (Extra Credit): : Projection matrix

[insert visualization of projected 3D points and actual 2D points for the first Argoverse image we provided here]



[insert visualization of camera center for the first Argoverse image here]



Part 4 (Extra Credit): : Projection matrix

[What two quantities does the camera matrix relate?]

The 3D coordinates of the point in the world space, and the 2D coordinates of the same point in the pixel space.

[What quantities can the camera matrix be decomposed into?]

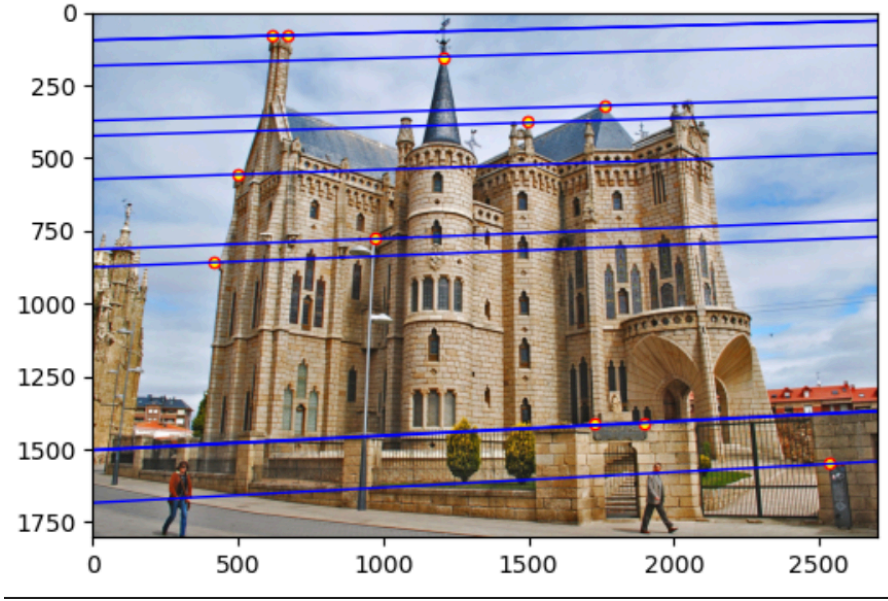
The camera matrix can be decomposed into the extrinsic and intrinsic matrices. The extrinsic matrix describes the camera's position and coordinates w.r.t the 3D world system, and is composed of the Rotational and Translational components. The intrinsic matrix contains the properties of the camera itself, and is the calibration matrix of the camera, comprising of values like the focal length, optical center etc.

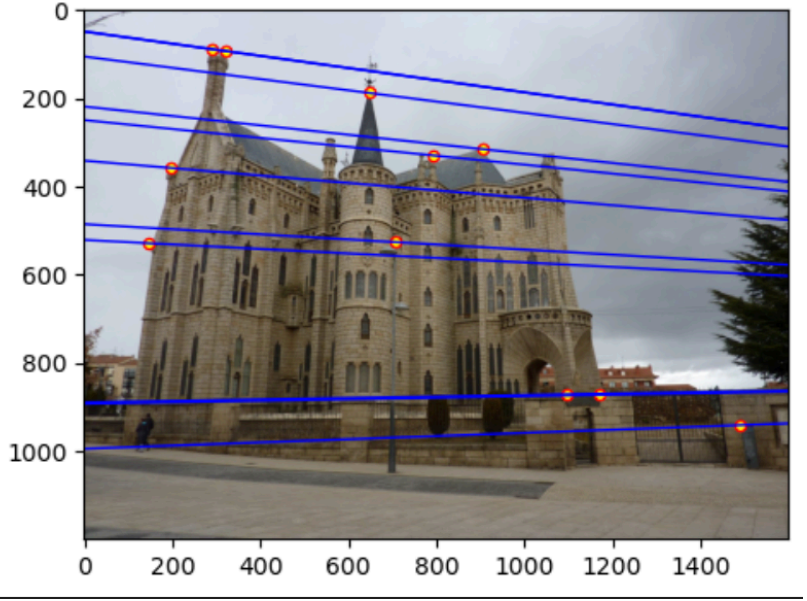
[List any 3 factors that affect the camera projection matrix.]

1. Focal Length, which changes the intrinsic matrix. Caused by changes in the lens, adjusting the zoom etc.
2. Camera Position (Translation) affects the extrinsic matrix, and particularly the translation component (t) of the same.
3. Camera Orientation (Rotation) affects the rotation component (R) of the extrinsic matrix.

Part 5 (Extra Credit): Fundamental matrix

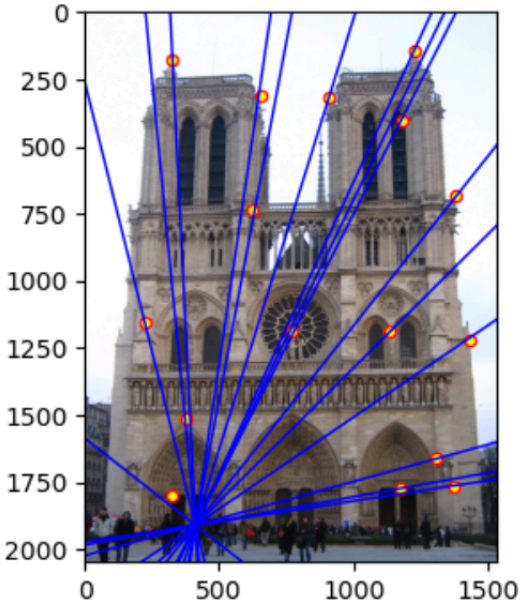
[insert visualization of epipolar lines on the Gaudi image pair]

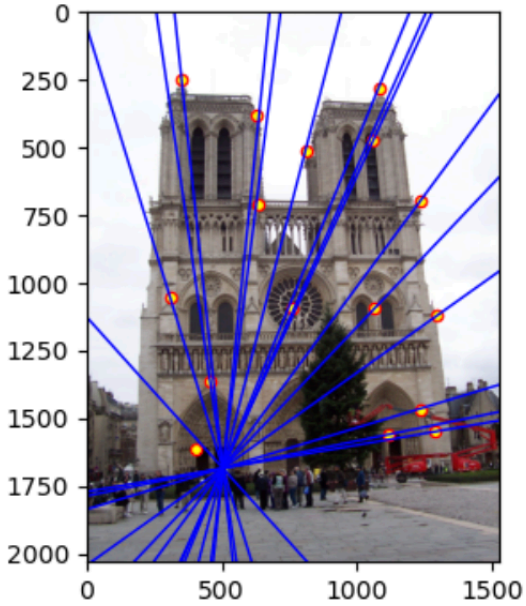




Part 5 (Extra Credit): Fundamental matrix

[insert visualization of epipolar lines on the Notre Dame image pair]





Part 5 (Extra Credit): Fundamental matrix

[Why is it that points in one image are projected by the fundamental matrix onto epipolar lines in the other image?]

Points in one image corresponds to a ray in the 3D space, from the camera's center through that pixel. When the 3D ray is viewed by the second camera, its projection onto the second image plane forms a 2D line. This is the epipolar line, representing all possible locations of that point from the second viewpoint. The fundamental matrix is the operator that does this projection, reducing the search space from a 2D area plane to a line.

[What happens to the epipoles and epipolar lines when you take two images where the camera centers are within the images? Why?]

When the camera centers are in the field of view of each other, the epipoles(projections of camera centers onto the other plane) become visible points within the images. The effect on the epipolar lines is that they now appear to radiate out from this single visible epipole, like the spokes of a wheel emerging from the center.

Part 5 (Extra Credit): Fundamental matrix

[What does it mean when your epipolar lines are all horizontal across the two images?]

This is a case where both the image planes are perfectly co-planar, and the baseline connecting the two camera centers is perfectly parallel to the horizontal axis of both the images.

[Why is the fundamental matrix defined up to a scale?]

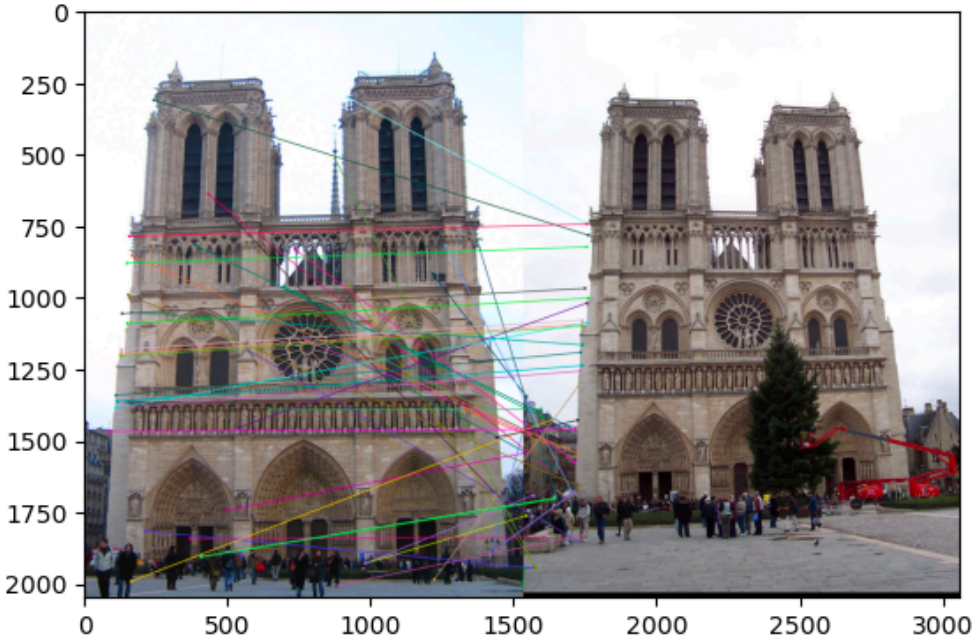
The fundamental matrix is defined up to a scale because there isn't necessarily a unique solution to it. P2T * F * P1 = 0 is the epipolar constraint, and this equation is independent of scaling F by a constant k. Since for any k this is true, this equation leads to an entire family of solutions, unless we constraint the value of the F matrix like making it unit-norm etc. while solving with SVD.

[Why is the fundamental matrix rank 2?]

The fundamental matrix is of rank 2 because of the geometric constraint that all epipolar lines must intersect at a point, the epipole. A matrix that maps all points in the first image plane to the family of lines that intersect at a point can't be full rank. This property means the determinant would be zero, leading to the Fundamental Matrix being a singular rank-2 matrix.

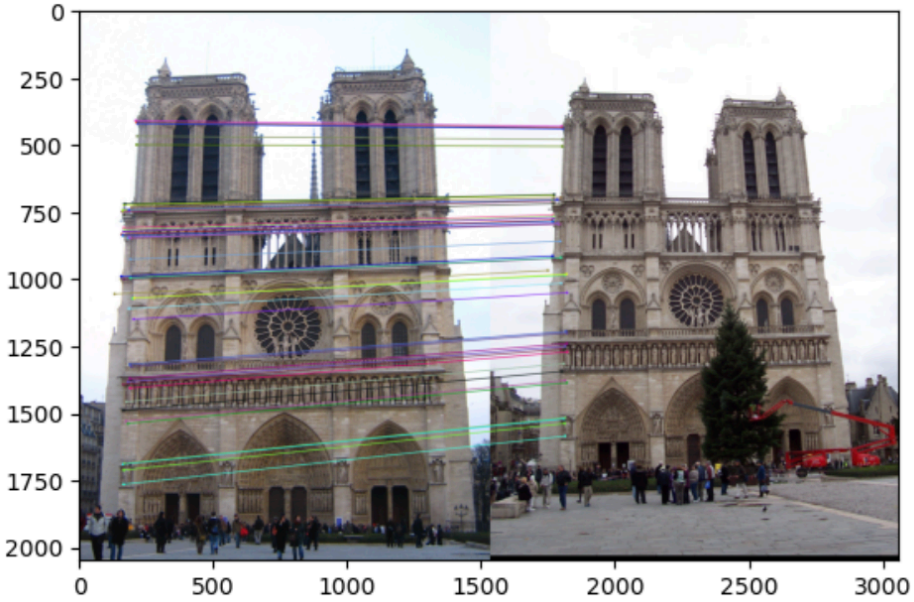
Part 6: RANSAC

[insert visualization of correspondences on Notre Dame without RANSAC]



Part 6: RANSAC

[insert visualization of correspondences on Notre Dame with RANSAC]



Part 6: RANSAC

[How many RANSAC iterations would we need to find the homography with 99.9% certainty for your Notre Dame SIFT results? Assume a hypothetical inlier ratio (correspondence accuracy) of 90% and that you are using the minimal sample size of 4 points]

Inlier_ratio = w = 0.9, number of points = n = 4
N = log(1 – p)/log(1 – w^n) = log(1 – 0.999) / log(1 – 0.9^4)
N = log(0.001) / log(0.3439) = 6.47…
The number of iterations we'd need = ceil(N) = 7

[One might imagine that if we have many correct matches, it would be better to use more than the minimum number of points to compute the homography in each sample. Investigate this by finding the number of RANSAC iterations you would need if you used a sample size of 8 points instead of 4 (keeping the same 99.9% certainty and 90% inlier ratio).

By increasing the number of sample points, the number of iterations will be a larger value, since 0.9^8 < 0.9^4. On computing with n=8, the number of iterations N will be 13. This is because the more points we pick up, the higher the chance that one of them is an outlier, and therefore the more iterations we'd need to run our RANSAC model. Since our approach now will be to minimize error considering more points, this process has an effect of averaging out the random noise from individual points, therefore leading to a more robust model. Even though its statistically harder to pick a set of pure inliers, that leads to the cost of extra iterations, it will lead to a more robust model.

[If our dataset had a lower point correspondence accuracy (i.e., a lower inlier ratio), say 70%, what is the minimum number of RANSAC iterations needed to find the homography with 99.9% certainty? Remember to use the minimal sample size of 4 points in your calculation.]

N = log(1 – p)/log(1 – w^n)

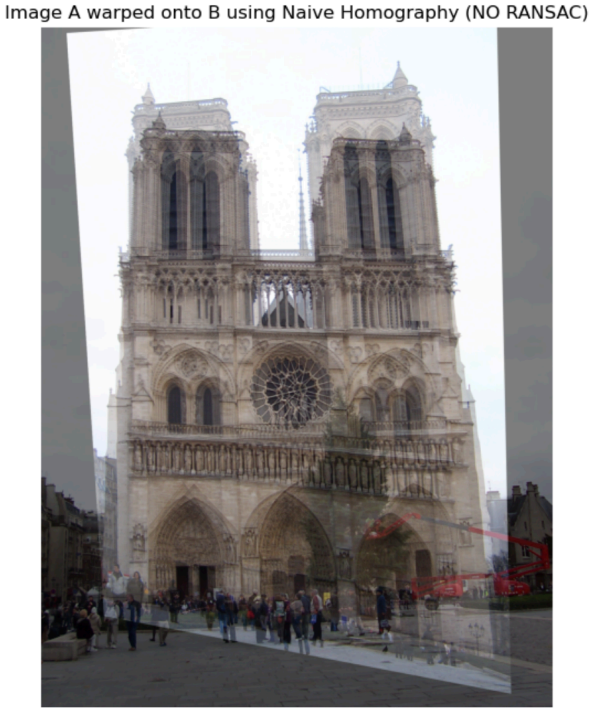
p = 0.999, n = 4, w = 0.7

N = log(0.001)/log(1 - 0.7^4) = log(0.001)/log(0.7599) = 25.16

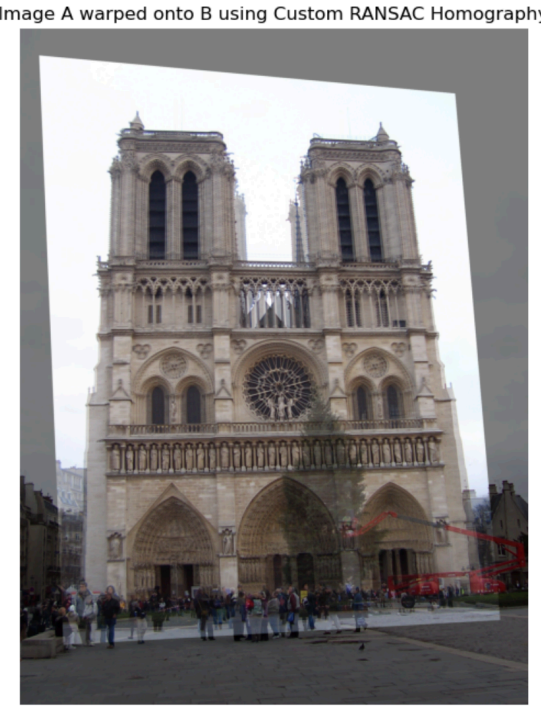
Our answer should be ceil(N) since we need to complete the last iteration, and therefore, we'd need 26 iterations, a much larger number than what we had required with a 90% inlier probability.

Part 6: Performance comparison

[insert visualization of the warped and blended image pair without RANSAC]



[insert visualization of the warped and blended image pair using RANSAC]



Part 6: Performance comparison

[Describe the different performance of the two methods.]
The naïve SIFT-based homography failed completely. The resulting warped image is severely distorted, sheared and incorrectly scaled, showing drastic misalignment with the target image. In contrast, homography with RANSAC fared much better, providing geometrically correct alignment of all features and the whole monument itself. The overlay in the warped image is precise as well.

[Why do these differences appear?]

The differences arise from how the 2 methods handle outliers. The naïve method takes all the SIFT matches, and tries to minimize the least squares error with a single Homography matrix. It assumes all the matches are correct, and therefore is fundamentally flawed, since it doesn't take into account, outliers which cause significant geometric errors. On the other hand, RANSAC selects a small set of points, and computes a candidate homography for these points, and later benchmarks this against all the other points to see how many matches are inliers. This terminates in the condition that we have found the largest set of inliers.

[Which one should be more robust in real applications? Why?]

The RANSAC method is definitely more robust in real applications. Feature matching in practical scenarios is very difficult to achieve with good accuracy. Naïve homography approaches cannot tolerate inevitable outliers, and therefore it is guaranteed to fail in majority of real-life applications. RANSAC systematically rejects outliers, and tries to find the true consensus in noisy data, and therefore it's the better choice for the task.

Conclusion

[What made the RANSAC-based homography estimation significantly more effective than the naive homography estimation? In your answer, discuss the role of outliers in the initial SIFT matches and explain how the two methods differ in handling them.]

The RANSAC-based homography estimation is significantly better because of its fundamentally different approach to handling outliers in the initial SIFT feature matches. While SIFT is a powerful feature matcher, it is not perfect and inevitably produces some outliers. A naïve homography estimation amplifies the outliers by using a MSE-type approximation for finding the best fit, thus going in with the assumption that these outliers are correct. This makes it extremely sensitive to those outliers, whose amplification leads to large geometric errors pulling the best-fit line far from the true solution and resulting in severe distortion we observed in slide 28.

The RANSAC method on the other hand is more robust to them. It works in multiple iterations by hypothesizing a model from a minimal set of points, and tries to find a consensus set of how many other points align with this model. By finding the model with the largest consensus set, we know we’re robust to outliers, since outliers are small in comparison to the total number of valid points. The final accurate homography is achieved by only using these inlier points, making the process robust to imperfections of the initial feature matching.