# Implementing the GaLAM Outlier Detection Algorithm

Yu Dinh
*School of STEM*
*University of Washington Bothell*
Bothell, WA
dudinh@uw.edu

Neha Kotwal
*School of STEM*
*University of Washington Bothell*
Bothell, WA
nehajk@uw.edu

Anoop Prasad
*School of STEM*
*University of Washington Bothell*
Bothell, WA
anoopp@uw.edu

***Abstract*—We implement the GaLAM algorithm in C++ to evaluate and understand a modern two-stage outlier detection method for image feature matching. Traditional matching pipelines often suffer from a high number of incorrect correspondences due to limitations in descriptors and the complexity of scenes. GaLAM improves robustness by combining an optimized version of AdaLAM for local affine verification with a global geometric consistency step based on RANSAC fundamental matrix estimation. Our implementation focuses on reconstructing the core component of GaLAM and provides a functional and modular framework for studying two-stage outlier detection. Our results mirror those of the original paper, showing that the GaLAM method is highly accurate but requires high computation time. Furthermore, we find that the GaLAM method is fairly sensitive to parameter choice.**

## I. Introduction

In computer vision, one key challenge is accurately matching features across two or more images of the same scene. When two images are captured from slightly different viewpoints, with different lighting, with moving objects, or with a variety of other transformations, the feature descriptors (such as SIFT or ORB) often produce many incorrect matches (outliers).

Outlier detection methods like GaLAM aim to exclude these outliers for more accurate matching. Our project focused on implementing the GaLAM algorithm as described in the paper *GaLAM: Two-Stage Outlier Detection Algorithm* [2] in C++ and comparing it against outlier detection methods that already exist in OpenCV to determine its strengths and weaknesses. The first stage of the method involves making assumptions about the problem using the properties of the real world to first select "seed points" – correspondences with a high confidence – from the set of all correspondences by filtering out poor matches using bidirectional nearest neighbor matching and non-maximum suppression, identifying the neighborhood of each seed point using keypoint and image information, and finally determining the affine transformations. The second stage involves removing some of the more stubborn incorrect seed points and correspondences maintained by the first stage by determining their projection deviation using a fundamental matrix fit from the seed points.

Our goals for this project were as follows:

1) Implement both stages of the GaLAM algorithm in C++, since no code for the original Python implementation of the algorithm is publicly available.
2) Understand the effect of each of the parameters of the parameter-heavy GaLAM algorithm on its performance.
3) Compare the performance, including both accuracy and computational efficiency, of the GaLAM method with other outlier detection techniques available in OpenCV, such as the ratio test, GMS, and RANSAC.

## II. Methods and Accomplishments

In this section, we describe the implementation work completed for the GaLAM outlier detection algorithm and the experimental pipeline we built to evaluate it against other outlier filtering methods.

### A. GaLAM Implementation in C++

We implemented the GaLAM two-stage outlier detection algorithm as a standalone C++ class that is compatible with OpenCV feature detection and matching pipelines. The implementation follows the structure described in the original paper as closely as possible to enhance understanding and ensure a close adherence to the paper's method. It also exposes a configurable `InputParameters` struct that groups all of the algorithm's tunable hyperparameters (e.g., non-maximum suppression ratio, ratio-test threshold, epipolar threshold, RANSAC iteration counts, and error adjustment parameters). This design allows us to easily change GaLAM parameters during testing without modifying the core algorithm code and makes future use of the algorithm simple.

Stage 1 of our implementation focuses on the local affine verification stage described in the original paper. We first perform bidirectional nearest-neighbor matching with a ratio test to obtain a set of reliable candidate correspondences; though this ratio test is not specified in the paper, we found that it improved accuracy and decided to include it. For each correspondence, we compute a confidence score based on the ratio-test value and apply non-maximum suppression in the image domain to select spatially well-distributed seed points. Around each seed, we then construct a local neighborhood

by enforcing spatial, scale, and rotational consistency constraints between each seed point correspondence pair and their neighboring correspondences, filtering out those points which do not meet the constraints. After neighborhoods are formed, we normalize the coordinates of the keypoints using their seed points as a basis and fit affine transformations using RANSAC (unlike the paper, which uses PROSAC, since the latter algorithm is not provided by OpenCV and we lacked time to implement it). Matches whose residuals exceed a threshold based on a combination of internally-derived values and user-provided hyperparameters are removed as outliers, leaving a set of stage 1 inliers that are consistent with a local affine transformation.

Stage 2 performs global geometric consistency checking using the remaining seed points and their neighborhoods. We sample subsets of seed points using a RANSAC-like procedure, estimate a fundamental matrix using the 8-point algorithm, and evaluate each candidate model over all seeds via an epipolar distance measure. Models with sufficient support are retained, and seed points that consistently agree with strong models are treated as globally consistent, with the points used as support maintained.

After both stages are run, a final ratio test might be run if the number of matches exceeds 200, as outlined in the paper, to remove any excessively bad matches.

The final output of GaLAM is a set of matches that have passed both local affine verification and global geometric consistency checking, corresponding closely to the "final inliers" defined in the original algorithm.

### *B. Benchmarking and Evaluation Pipeline*

To evaluate GaLAM in a controlled and reproducible way, we implemented a benchmarking harness modeled after the one used in the first testing section of the original paper, which we called the `MatchTest` class. It utilizes the Oxford Affine Dataset [1] with image sequences similar to those of the HPatches dataset used in the original paper. The dataset includes several sets of six images which each include a transformation as well as a ground-truth homography from the first image in each set to each of the other five. The pipeline supports multiple feature detectors (currently SIFT due to its suitability for GaLAM, with the ability to add ORB or AKAZE) and multiple outlier-removal methods, including:

- A nearest-neighbor + ratio test (NN+RT) baseline without additional outlier removal,
- A homography-based RANSAC filter using OpenCV's `findHomography`,
- The OpenCV implementation of the GMS matching algorithm, which incorporates outlier filtering, and
- Our C++ implementation of GaLAM.

For each of the eight scenes and five image pairs per scene in the dataset, the pipeline:

1) Detects keypoints and computes descriptors in both images using the chosen detector.
2) Performs KNN matching to obtain initial matches.
3) Applies a selected outlier-removal method (NN+RT, RANSAC, GMS, or GaLAM) to filter the matches.
4) Evaluates the filtered matches against the ground-truth homography using a projection error metric.

We defined a unified `Metrics` struct to record, for each method and image pair, the number of correspondences, the average reprojection error, the percentage of matches with small error (e.g., within a few pixels), and the runtime in milliseconds, similar to the testing methodology used in the original paper's first test. These metrics are logged to a CSV file for later analysis and are also printed in a formatted summary to the console. The logging format is designed to support computing averages over scene categories (e.g., viewpoint changes vs. illumination changes) in the final analysis.

### *C. Visualization and Reproducibility*

For selected scenes and image pairs, the pipeline saves:

- Images showing the initial nearest neighbor matches between two images.
- Images showing the filtered matches produced by each method (NN+RT, RANSAC, GMS and GaLAM).
- Intermediate GaLAM stage output including seed point selection and local affine verification results.
- Per-image CSV logs containing correspondence counts, reprojection errors, inlier percentages and runtime measurements.
- Summary statistics aggregated by scene category, including viewpoint, illumination, blur, zoom and rotation, JPEG compression.
- Console output with formatted tables comparing all methods in the dataset.

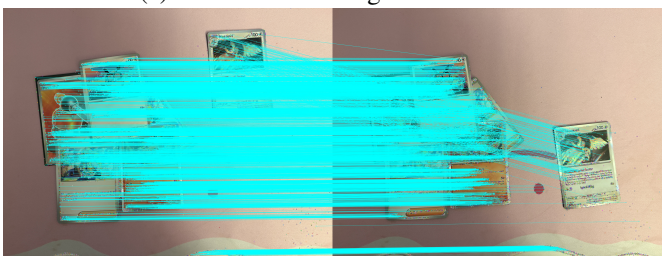
These visualizations are saved for one image pair per scene and automatically named by scene, detector, and method, which makes it easy to inspect how GaLAM behaves compared to other methods on the same inputs.

In addition to the quantitative tests and metrics outlined above, we generate visualizations to compare the methods qualitatively. Figure 1 illustrates the progressive filtering performed by GaLAM on a scene containing multiple Pokémon cards with similar visual features, one of which has been moved between the two images. The initial matching produces thousands of correspondences, many of which incorrectly match features between different cards or to background regions. The seed selection step finds a small set of reliable points. Local affine verification then builds neighborhoods around each of these seeds and removes matches that violate local geometric constraints, drastically reducing the number of correspondences. Finally, global geometric consistency with the fundamental matrix eliminates the remaining outliers that do not conform to the overall geometry of the scene, finishing the algorithm with most outliers filtered out successfully.

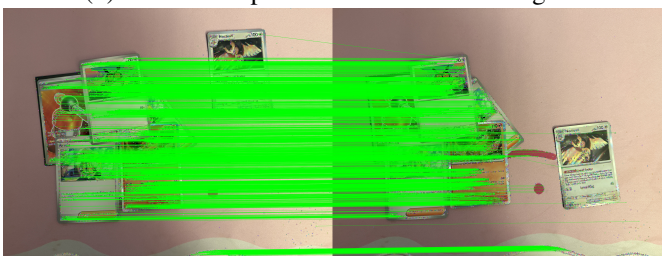
We also implemented a command-line interface that takes a dataset path and CSV output path as arguments and runs the full benchmark over all configured detectors and methods. This makes the experiments easy to rerun with different parameter

(a) Initial nearest-neighbor matches

(b) Seed correspondences obtained in stage 1



(c) Stage 1 correspondences after local affine verification



(d) Final GaLAM correspondences after stage 2

Fig. 1. **Visualization of correspondences through the GaLAM pipeline on a pair of Pokémon card images**.
(a) The raw SIFT nearest neighbor matches show thousands of correspondences, with many crossing incorrectly between different cards and the background as well as matches with the moved card.
(b) After non-maximum suppression based on confidence scores, GaLAM selects approximately 50 high-confidence matches as seed point correspondences (blue lines).
(c) Stage 1 finds correspondences in the local neighborhood of each seed point and applies the local affine verification step, retaining matches that are consistent with an affine transformation within each local region (cyan lines) while removing many gross outliers which are not.
(d) The final set of correspondences after Stage 2 (green lines) shows a sizable set of accurate matches concentrated on the cards that appear in both images, with cross-card mismatches and matches on the moved card largely eliminated.
Input parameters: (ratio = 100, rt_threshold = 0.8, epsilon = 1.0, lambda1 = 3.0, lambda2 = 3.0, lambda3 = 1.1, t_alpha = 10.0, t_sigma = 0.5, finalRTThreshold = 0.9, num_iterations = 128, minSampleSize = 8).

settings or datasets and ensures that our final results can be reproduced by other researchers using the same code base. For simpler tests and demonstrations, the command-line interface also supports running GaLAM on the SIFT matches found between two specified images, saving the results after each stage for quick evaluation and logging information to the console, as was done for the Pokémon images mentioned previously.

## III. Results

In this section, we present our experimental findings from our C++ implementation of the GaLAM outlier detection algorithm. The goal of these experiments was to verify that our implementation reproduces the behavior of the original algorithm to some extent, and to compare GaLAM against the other methods provided by OpenCV on the Oxford Affine Dataset, similar to the HPatches dataset used in the original paper.

### *A. Quantitative Comparison to Other Methods*

Using the evaluation pipeline described in Section II, we recorded, for each tested method, the number of matches retained, the average reprojection error compared to the ground-truth homography, the proportion of low-error matches, and the runtime.

Across the small set of evaluated image pairs, some observations we made are as follows:

- GaLAM retains fewer correspondences than other methods, but with significantly lower reprojection error.
- GaLAM achieves performance generally better than or comparable to RANSAC, and almost always better than NN+RT and GMS. In particular, it excels compared to other methods with image pairs that involve viewpoint, blur, or JPEG compression changes.
- GaLAM seems not to do as well on pairs which involve zoom and rotation changes.
- GaLAM incurs a higher runtime due to its two-stage procedure, especially the affine fitting and fundamental matrix sampling stages.
- Our implementation of GaLAM seems to have small variations between executions, likely due to the randomness introduced by the usage of RANSAC internally. However, these are generally small, and do not broadly affect the reliability of the method or implementation. It is unknown to us whether the same would be true of the original GaLAM method.

These initial findings support those of the original GaLAM paper: that two-stage filtering improves correspondence quality at the cost of requiring additional computation time [2].

Table I.I and I.II summarize the homography estimation performance of four correspondence verification methods under various transformations in the Oxford data set: NN+RT,

TABLE I.I
HOMOGRAPHY ESTIMATION ACCURACY FOR VIEWPOINT, LIGHTING, AND BLUR TRANSFORMATIONS USING SIFT FEATURE DETECTION.

| Method | View %H.E / AP | Light %H.E / AP | Blur %H.E / AP |
|---|---|---|---|
| NN+RT | 30.62 / 56.19 | 76.62 / 85.97 | 36.05 / 64.10 |
| RANSAC | 35.84 / 65.35 | 87.36 / **99.82** | 35.21 / 56.31 |
| GMS | 33.59 / 64.30 | 67.12 / 76.53 | 29.50 / 55.00 |
| GaLAM | **41.03 / 70.75** | **88.94** / 98.96 | **52.78 / 90.65** |

TABLE I.III
SUMMARY OF CORRESPONDENCE STATISTICS USING SIFT FEATURE DETECTION.

| Method | Avg Corr | Avg Error | Inlier% | Runtime (ms) |
|---|---|---|---|---|
| NN+RT | 1121.0 | 67.26 | 72.49 | **0.00** |
| RANSAC | 1167.3 | 70.08 | 75.33 | 19.23 |
| GMS | **1272.5** | 34.59 | 70.84 | 29.07 |
| GaLAM | 567.9 | **11.70** | **88.65** | 1516.37 |

RANSAC, GMS, and GaLAM. For each distortion category (viewpoint, lighting, blur, zoom and rotation, and JPEG compression), we report the percentages of the homography estimation accuracy (%H.E) and the average precision (AP) of the verified inliers.

TABLE I.II
HOMOGRAPHY ESTIMATION ACCURACY FOR ZOOM+ROTATION AND COMPRESSION TRANSFORMATIONS USING SIFT FEATURE DETECTION.

| Method | Z+R %H.E / AP | Comp %H.E / AP |
|---|---|---|
| NN+RT | 44.99 / 82.87 | 75.14 / 87.67 |
| RANSAC | 49.18 / 89.74 | 68.83 / 80.00 |
| GMS | 48.98 / 85.81 | 66.23 / 79.98 |
| GaLAM | **50.85 / 94.86** | **81.98 / 98.88** |

Across all distortion categories and with all feature detection algorithms, GaLAM achieves the highest homography estimation accuracy and a high inlier average precision. In particular, GaLAM shows substantial gains in some of the more challenging categories, such as viewpoint, blur, and JPEG compression, where descriptor quality degrades. Looking at the SIFT results, GaLAM achieves 52.72% homography accuracy under blur, outperforming NN+RT (36.05%), RANSAC (35.21%), and GMS (29.50%) by a large margin. Similar improvements are seen under JPEG compression, where GaLAM reaches 81.98% accuracy, compared to 75.14% for NN+RT, 68.83% for RANSAC, and 66.23% for GMS. Although all methods seem to have middling performance with viewpoint changes, GaLAM still does better than the others, with a homography accuracy of 41.03% compared to 35.84% for RANSAC, 33.59% for GMS, and 30.62% for NN+RT.

These results highlight the strengths and limitations of each method. NN+RT often provides weaker correspondences that are sensitive to noise and changes, but also manages to outperform RANSAC and GMS on occasion. GMS improves robustness by leveraging grid-based motion statistics, but fails to handle many of the distortions present in the image pairs. RANSAC remains a strong geometric estimator but assumes a single global transformation, making it less effective when local deviations are present. GaLAM, while not perfect, shows its robustness in handling a variety of transformations effectively, thanks to its two-stage approach helping it handle challenging image pairs.

Table I.III provides a global summary of correspondence verification performance across all image pairs, as run with SIFT as the feature detection algorithm. NN+RT, RANSAC, and GMS retain a large number of correspondences, with GMS managing to retain the most. However, each exhibits significantly higher projection error, with RANSAC having the worst average error of 70.08, NN+RT following it with an average error of 67.26, and GMS having a decent average error of 34.59, despite rarely performing better than them in other metrics. A different pattern is observed with the inlier rate, where RANSAC outperforms NN+RT and GMS. The runtimes follow the pattern one might expect, where the methods take longer to run based on their complexity, meaning that NN+RT is the fastest.

Comparing GaLAM to the other three methods, we observe that GaLAM produces the fewest correspondences (567.9 on average), yet yields the lowest average error of 11.70 and the highest inlier rate of 88.65%, confirming that its two-stage approach leads to substantially more accurate correspondence sets. However, this comes at the cost of resources and, therefore, the runtime: GaLAM is orders of magnitude slower (1414 ms) due to its two-stage approach, particularly the fitting of the fundamental matrix in Stage 2, is computationally intensive.

These results and those on the following page also demonstrate that the high performance of GaLAM varies little when using ORB or AKAZE instead of SIFT.

TABLE II.I
HOMOGRAPHY ESTIMATION ACCURACY FOR VIEWPOINT, LIGHTING, AND BLUR TRANSFORMATIONS USING ORB FEATURE DETECTION.

| Method | View %H.E / AP | Light %H.E / AP | Blur %H.E / AP |
|---|---|---|---|
| NN+RT | 30.62 / 56.19 | 76.62 / 85.97 | 36.05 / 64.10 |
| RANSAC | 35.84 / 65.35 | 87.36 / **99.82** | 35.21 / 56.31 |
| GMS | 33.59 / 64.30 | 67.12 / 76.53 | 29.50 / 55.00 |
| GaLAM | **40.57 / 70.66** | **88.92** / 98.96 | **52.39 / 90.75** |

### *B. Qualitative Behavior in Oxford Affine Dataset*

We also conducted qualitative analysis on the final matches for some image pairs from the Oxford Affine Dataset. For each of these pairs, We visualized the original nearest-neighbor matches from SIFT, the matches retained following the ratio-test (NN+RT), the matches retained by OpenCV's homography-based RANSAC filter, the matches obtained by the OpenCV extra modules' implementation of the GMS algorithm, and the matches retained after both stages of GaLAM.

NN+RT typically produced a large number of correspondences, but many were clear outliers, especially in complex scenarios. RANSAC did impressively well in the quantitative testing, often coming in with the second-best or even best results, and this holds up in qualitative examination as well, where the filtered matches seem to be accurate for the most part. GMS similarly holds up fairly well under a qualitative

TABLE II.II
HOMOGRAPHY ESTIMATION ACCURACY FOR ZOOM+ROTATION AND COMPRESSION TRANSFORMATIONS USING ORB FEATURE DETECTION.

| Method | Z+R %H.E / AP | Comp %H.E / AP |
|---|---|---|
| NN+RT | 44.99 / 82.87 | 75.14 / 87.67 |
| RANSAC | 49.18 / 89.74 | 68.83 / 80.00 |
| GMS | 48.98 / 85.81 | 66.23 / 79.98 |
| GaLAM | **50.82 / 94.80** | **81.89 / 98.86** |

TABLE II.III
SUMMARY OF CORRESPONDENCE STATISTICS USING ORB FEATURE DETECTION.

| Method | Avg Corr | Avg Error | Inlier% | Runtime (ms) |
|---|---|---|---|---|
| NN+RT | 1121.0 | 67.26 | 72.49 | **0.00** |
| RANSAC | 1167.3 | 70.08 | 75.33 | 19.23 |
| GMS | **1272.5** | 34.59 | 70.84 | 29.07 |
| GaLAM | 567.9 | **11.70** | **88.65** | 1414.22 |

TABLE III.I
HOMOGRAPHY ESTIMATION ACCURACY FOR VIEWPOINT, LIGHTING, AND BLUR TRANSFORMATIONS USING AKAZE FEATURE DETECTION.

| Method | View %H.E / AP | Light %H.E / AP | Blur %H.E / AP |
|---|---|---|---|
| NN+RT | 30.62 / 56.19 | 76.62 / 85.97 | 36.05 / 64.10 |
| RANSAC | 35.84 / 65.35 | 87.36 / **99.82** | 35.21 / 56.31 |
| GMS | 33.59 / 64.30 | 67.12 / 76.53 | 29.50 / 55.00 |
| GaLAM | **40.55 / 70.66** | **88.92** / 98.96 | **52.27 / 90.35** |

TABLE III.II
HOMOGRAPHY ESTIMATION ACCURACY FOR ZOOM+ROTATION AND COMPRESSION TRANSFORMATIONS USING AKAZE FEATURE DETECTION.

| Method | Z+R %H.E / AP | Comp %H.E / AP |
|---|---|---|
| NN+RT | 44.99 / 82.87 | 75.14 / 87.67 |
| RANSAC | 49.18 / 89.74 | 68.83 / 80.00 |
| GMS | 48.98 / 85.81 | 66.23 / 79.98 |
| GaLAM | **50.84 / 94.80** | **81.89 / 98.86** |

TABLE III.III
SUMMARY OF CORRESPONDENCE STATISTICS USING AKAZE FEATURE DETECTION.

| Method | Avg Corr | Avg Error | Inlier% | Runtime (ms) |
|---|---|---|---|---|
| NN+RT | 1121.0 | 67.26 | 72.49 | **0.00** |
| RANSAC | 1167.3 | 70.08 | 75.33 | 19.23 |
| GMS | **1272.5** | 34.59 | 70.84 | 29.07 |
| GaLAM | 567.9 | **11.70** | **88.65** | 1445.37 |

examination, with few noticeable outliers, like RANSAC. GaLAM clearly has few outliers on most tested image pairs, but on some, such as the graffiti images, it seems to remove a huge number of correspondences, meaning it most likely took out many inliers with the outliers due to its strict requirements.

In light of this, however, we can see that our implementation of GaLAM consistently produces fairly clean correspondence sets with behavior roughly in line with that of the paper, where GaLAM noticeably removed many inliers along with the outliers in testing but were more accurate overall.

### C. Parameter Sensitivity

Because GaLAM includes several tunable hyperparameters, we explored different values for each of them to determine the effects they have on the method's performance. The parameters we investigated were as follows:

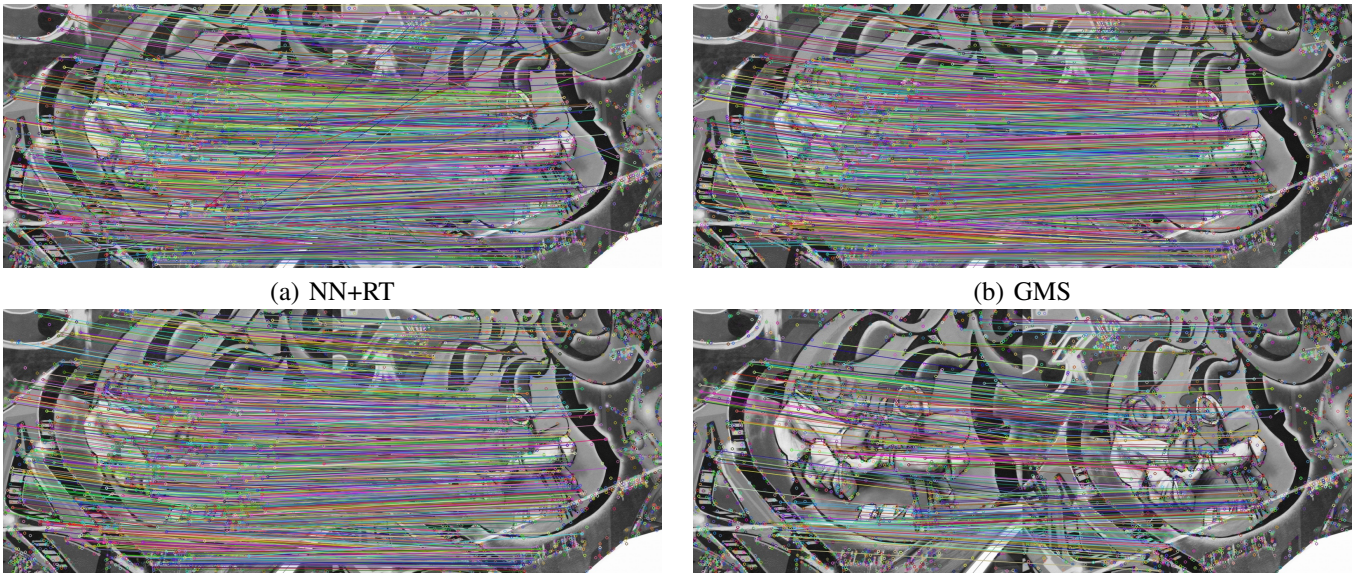
- The ratio used in non-maximum suppression in Stage 1 (`ratio`),
- The threshold for the ratio test used in bidirectional nearest-neighbor matching (`rt_threshold`),
- The fundamental matrix projection deviation threshold (`epsilon`),
- The scaling parameter for neighborhood sizes (`lambda1`),
- The neighborhood point error range adjustment parameter (`lambda2`),
- The parameter used for thresholding projection deviation from the fundamental matrix in Stage 2 (`lambda3`),
- The threshold for rotational consistency in Stage 1 (`tAlpha`),
- The threshold for scale consistency in Stage 1 (`tSigma`),
- The number of iterations used for RANSAC (`num_iterations`),
- The minimum sample size for fitting a fundamental matrix in Stage 2 (`minSampleSize`), and
- The threshold for the final ratio test applied at the end if there are too many correspondences (`finalRTThreshold`).

TABLE IV
PARAMETER SENSITIVITY ANALYSIS VALUES

| Parameter | | | |
|---|---|---|---|
| `ratio` | 50 | 100 | 200 |
| `rt_threshold` | 0.5 | 0.8 | 0.9 |
| `epsilon` | 0.5 | 1.0 | 3.0 |
| `lambda1` | 3 | 4 | 5 |
| `lambda2` | 1 | 2 | 3 |
| `lambda3` | 0.4 | 0.8 | 0.9 |
| `tAlpha` | 10 | 20 | 50 |
| `tSigma` | 0.3 | 0.5 | 1.0 |
| `num_iterations` | 64 | 128 | 256 |
| `minSampleSize` | 8 | 12 | 16 |
| `finalRTThreshold` | 0.4 | 0.9 | 0.95 |

We found that, although some of the parameters do not have a large effect on the performance of the method, others can create notable differences. The parameters which had very little effect when changed included `ratio`, `rt_threshold`, `epsilon`, `tSigma`, `num_iterations`, and `minSampleSize`, and `finalRTThreshold`; although some of these are predictable, such as `finalRTThreshold` due to its rare application and `num_iterations` and `minSampleSize` due to the general reliability of RANSAC and the 8-point algorithm for fundamental matrix estimation even without needing to increase iterations (and therefore runtime), the others were quite surprising, as we expected them to have noticeable differences.

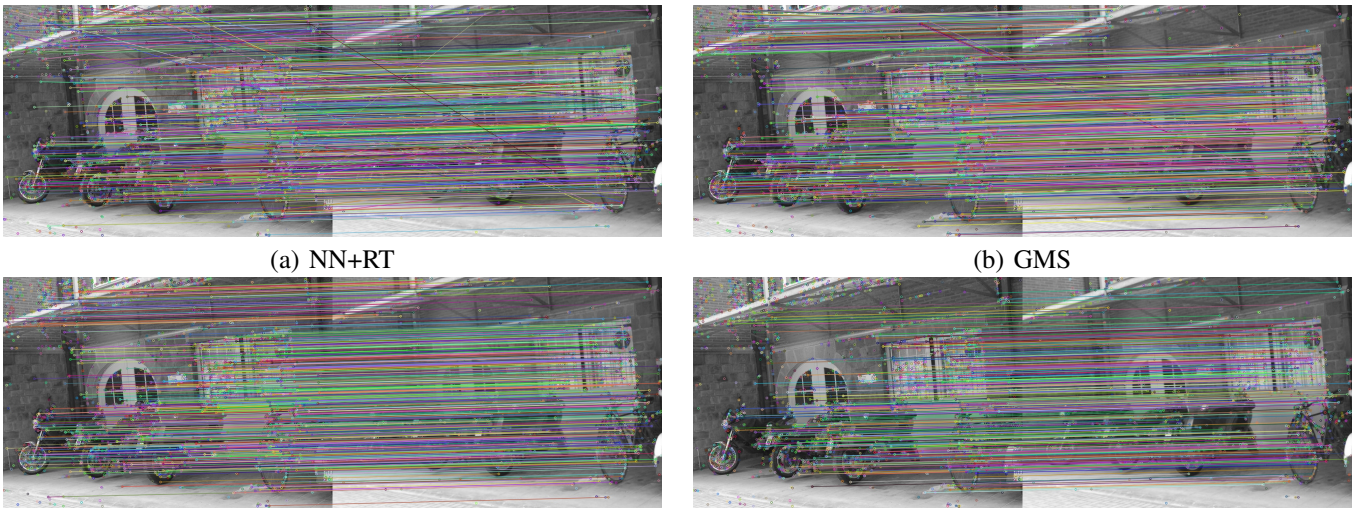
The parameters which we found to be high sensitivity were each of `lambda1`, `lambda2`, and `lambda3`, as well as `tAlpha`. Changing any of these variables led to notable changes in performance, typically improving certain metrics



(a) NN+RT (b) GMS

(c) RANSAC (d) GaLAM

Fig. 2. Comparison of correspondence filtering results on a *graf* image pair. Although GaLAM performed well in terms of accuracy, this is a prime example of GaLAM maintaining very few correspondences in some scenarios compared to other methods, which could have negative repercussions for some applications.



(a) NN+RT (b) GMS

(c) RANSAC (d) GaLAM

Fig. 3. Comparison of correspondence filtering results on a *bike* image pair. We see fairly good results from all methods, though NN+RT has some obvious outliers and GaLAM seems to have the fewest outliers.

significantly while decreasing others drastically. Although the $\lambda$ parameters having a significant effect is to be expected due to their being the "primary" parameters mentioned in the original paper, `tAlpha` being sensitive to change was a surprise, as it was seemingly unimportant enough to have no stated suggested value in the original paper.

One parameter that deserves further discussion is `lambda3`. The original paper mentions that $0 < \lambda_3 < 1$ is a constraint for the parameter; however, we also tested setting the value to precisely 1, and even some higher values, which led to an extremely strict filtering in Stage 2. This sometimes led to appealing results, as can be observed in Figure 1. This result could speak to a disparity between our implementation and that of the authors, or something which the authors did not consider in their experimentation; either way, it might be worth future investigation.

Although the best parameters will vary based on application, the best performing combination we found for the Oxford Affine Dataset was:

- `ratio = 100`
- `rt_threshold = 0.8`
- `epsilon = 1.0`
- `lambda1 = 4.0`
- `lambda2 = 2.0`
- `lambda3 = 0.9`
- `tAlpha = 20.0`

- `tSigma = 0.5`
- `num_iterations = 128`
- `minSampleSize = 8`
- `finalRTThreshold = 0.9`

Notably, this combination is extremely similar to the values recommended by the authors of the original GaLAM paper, though differs slightly in some aspects, like `lambda3` being set to 0.9 instead of 0.8 as in the paper.

## IV. Conclusion

In this project, we implemented the complete GaLAM two-stage outlier detection algorithm in C++ using OpenCV, recreating the algorithm as described in the original paper with minor deviations. This includes the first stage of local affine verification – including bidirectional nearest-neighbor matching, seed point selection through non-maximum suppression on confidence scores assigned via ratio-test values, and local affine neighborhood verification with geometric and scale constraints to filter out points – and the second stage of global geometric consistency testing using fundamental matrix estimation on the remaining correspondences.

We developed a comprehensive evaluation pipeline that enables fair and consistent comparison of GaLAM against multiple matching and outlier filtering methods like NN+RT, RANSAC, and GMS using different feature detectors, including SIFT, ORB, and AKAZE. These tests are performed on image pairs using diverse transformations from the Oxford Affine Dataset, which include viewpoint, lighting, blur, zoom and rotation, and JPEG compression transformations; our testing implementation could also be adapted to other similar datasets with minor effort.

Our experimental results demonstrate that GaLAM's two-stage approach provides meaningful improvements over traditional single-stage methods, particularly on challenging viewpoint-change scenes where other methods struggle with high outlier ratios. The combination of local affine verification with global geometric consistency effectively filters false matches while preserving some true correspondences, though the method does have downsides regarding high computational requirements compared to other methods (leading to longer runtimes), removing many inliers alongside the outliers (which results in fewer correspondences), and being highly sensitive to some parameters (which can lead to seemingly inconsistent results if parameters are not chosen carefully). Future work may explore further parameter optimization and sensitivity analysis (especially to determine patterns in what parameters work well and when), additional comparisons to other methods, testing on other datasets, implementing GPU acceleration in C++ to match the original paper's Python implementation with GPU acceleration, or extending GaLAM to support dense correspondence methods.

## Team Member Contributions

All three team members – Yu Dinh, Neha Kotwal, and Anoop Prasad – worked collaboratively on most parts of the project. Together, we shared responsibilities for design, implementation, debugging, and refinement and jointly developed both the main method and the testing and evaluation pipeline. All members participated in understanding the paper, developing the system, conducting experiments, analyzing results, and creating the demonstration and final report. The project was completed through continuous teamwork, with each member contributing to every major component while also primarily focusing on developing and improving a specific component of the system.

**Du (Yu) Dinh**

- Designed and implemented Stage 2 of the GaLAM algorithm, involving global geometric consistency. Responsibilities included RANSAC sampling for fundamental matrix estimation, projection deviation computation, and filtering seed points based on global consistency metrics.

**Neha Kotwal**

- Developed and executed initial seed point selection along with the testing and evaluation pipeline. This included preparing datasets, running experiments across different parameters settings and baseline methods, and generating quantitative metrics and visualizations. Responsibilities focused on implementing the initial seed point selection code, MatchTest benchmarking framework for systematic comparison of GaLAM against the NN+RT, RANSAC, and GMS methods across the Oxford Affine Dataset.

**Anoop Prasad**

- Designed and implemented Stage 1 of the GaLAM algorithm, focusing on local affine verification. This included seed point selection, neighborhood construction, affine transformation estimation, and filtering based on local affine transformation adherence.

## References


[1] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 10, pp. 1615–1630, Oct. 2005, doi: 10.1109/TPAMI.2005.188.

[2] X. Lu, Z. Yan, and Z. Fan, "GaLAM: Two-stage outlier detection algorithm," *IEEE Access*, vol. 13, pp. 76135–76144, 2025, doi: 10.1109/ACCESS.2025.3561823.