3D Scene Reconstruction from a Video Using Structure from Motion

ARI2129: Principles of Computer Vision for AI
Group Project 2025

AI Report

Daniela Curmi, 0058805L,
Liam Azzopardi, 0044605L,
Matthew Pirotta, 0297505L,
Matthew Privitera, 0393704L,
Matthew Vella, 0099105L.

Generative AI Usage

Introduction

Throughout the project, Gemini 2.5 Flash, Claude 3.7 and ChatGPT-4o were the main AI models that were used. Claude was mainly used for the initial code planning phase because of its robustness in code design and logic structure. GPT-4o was then used for the implementation phase for its efficient, clean code generation. Gemini was used when explanations were needed for complex concepts and outputs because of its high usage limits.

Ethical Considerations

The use of generative AI in this project raises several important considerations. Since large language models are trained on internet text, they may contain biased perspectives, outdated information or underrepresented viewpoints in computer vision research. This bias can take many forms. The AI might favor certain algorithmic approaches over others based on their popularity in online discussions or history rather than their technical rigor. This may potentially overlook newer or less commonly discussed methods.

While this project involved publicly available computer vision algorithms, the use of generative AI means that the prompts and code segments shared with the system are stored by the providers. Although the technical content of this project does not contain any sensitive information, it is still important to be mindful about what information is shared with AI systems.

While AI provided explanations of existing concepts and helped debug implementation issues, the challenge lies in distinguishing between AI-assisted learning and AI-generated content. This project maintained academic integrity by using AI primarily as an educational and debugging tool rather than as a code generator. This is done to make sure that any analysis or conclusions are from group member's understanding and reasoning.

Methodology

The generative AI models were accessed through their respective web interfaces.
Generated code outputs were reviewed and modified as needed before being implemented into the source code.

Feature Detection - Daniela Curmi:

1 Method to generate a montage of feature detection frames - ChatGPT's GPT-4o

<u>*Prompt:*</u>

"Can you provide a method that uses all the frames produced by the feature detection algorithm and returns a video of a montage of frames for each feature detection algorithm, respectively?"

<u>*Response:*</u>

```
def create_feature_detection_montage(frames, output_dir='feature_detection_montage_videos', fps=10):   .....
# Release all video writers
for writer in video_writers.values():
    writer.release()
```

<u>*Analysis:*</u>

This response helped produce 5 videos of feature detection montages to construct a better qualitative analysis of how well each algorithm keeps track of features and the new/discarded features per frame.

2 List of research papers used as a source of info. for the literature review - ChatGPT's GPT-4o

<u>*Prompt:*</u>

"Generate a list of research papers related to the 3D scene visualisation pipeline described in this document (assignment description file) so that I can construct a literature review that focuses on how this process is typically done, what methods are used and why, what algorithms have been used across the years of research in the field. These papers should be the main source of my knowledge when writing this literature review."

<u>*Response:*</u>

References

1. H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-Up Robust Features (SURF)," Computer Vision and Image Understanding, vol. 110, no. 3, pp. 346-359, 2008. …

10. M. Muja and D. G. Lowe, "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration," in International Conference on Computer Vision Theory and Applications, 2009, vol. 1, pp. 331-340.

<u>*Analysis:*</u>

When looking into past research for all components of this project, this list of references helped narrow down the information needed to provide a well-constructed literature review. It also provided more insight into what outputs we should expect moving forward with the actual implementation.

3 Qualitative Plots for analysis - ChatGPT's GPT-4o

<u>*Prompt:*</u>

"Could you provide a plot for the number of keypoints and time, per frame for each algorithm?"

<u>*Response:*</u>

```
def plot_keypoints_and_times(results, frame_indices):
        detector_names = list(results.keys()) …
        for name in detector_names: …
        for name in detector_names: …
        plt.tight_layout()
            plt.show()
```

Analysis:
This plotting code helped produce the diagrams used to qualitatively analyse both the number of keypoints produced per frame as well as the computational time it took for each algorithm.

Feature Detection - Liam Azzopardi

1 Algorithm Parameter Tuning - ChatGPT 4o
Prompt:
“Recommended Parameter tuning for 8 second video for my Shi-Tomasi algorithm”

Response:
“Great question! Let’s break down how you can reduce the number of keypoints detected by SIFT using its parameters — while still keeping the most meaningful ones.
Ways to reduce keypoints using SIFT parameters….”

Analysis:
When it came to parameter tuning, I used this prompt to aid in finding parameters that yield optimal results in feature detection and the 3D representation. Although this prompt explicitly stated Shi-Tomasi, I used a number of prompts to cover parameter tuning of all algorithms and try to reason each parameter consistently to come to a better result.

2 Understanding Feature Detection - ChatGPT 4o

Prompt: “kp, des = orb.compute(img, kp) what does this do”

Response:

“This line is part of **OpenCV's ORB workflow**, and it means:

- You already have a list of keypoints (`kp`), probably detected using `orb.detect(...)` or some other detector.
- Now you're using `orb.compute(...)` to **calculate descriptors** for those keypoints…”

Analysis:
This prompt was used to be able to understand a critical aspect of the implemented code and to understand how a part of feature detection works. By exploring what orb.compute(img, kp) does, it helped clarify any distinction between keypoint detection and descriptor computation within the ORB pipeline. Understanding this difference was important for correctly applying feature matching techniques later in the project, as descriptors provide the numerical representations (rotation and scale) needed for matching keypoints across frames.

Feature Matching - Matthew Pirotta

1 Clearing confusion on how Lowe's ratio works - ChatGPT 4.o

Prompt:
"But how does Lowe's ratio know that it's rejecting matching features on a flat sky? Compared to a good feature point."

Response:
Lowe's ratio doesn't know anything about 'sky' or 'rock'... It operates purely on descriptor distances. In flat regions, descriptors tend to be weak or ambiguous. Several descriptors from different areas (e.g., blue sky) may look nearly identical.

When matching, the top two matches will have similar distances → ratio test fails → match is rejected. That's why Lowe's ratio effectively filters low-texture or repetitive areas — not because it 'knows' them semantically, but because the descriptor ambiguity reveals itself in the ratio test.

Analysis:
After researching how Lowe's ratio works, I was unsure of how few particular parts work. After prompting the generative AI model and reading its explanation, I got a better understanding of how the model fundamentally works. This helped me to more confidently analyse the results of Lowe's ratio on the frames.

2 Cleaning code structure - ChatGPT 4.o

Prompt:
I want to separate the functionality of detecting the keypoints and descriptors, so that I can pass them to match_features, as well as plotting them.

Response:
You can update your detect_features function to return a dictionary of (key points, descriptors), and have match_features consume that.

```
def detect_features(frames, detector_func):
    detector_results = {}
    keypoint_counts = []
    computation_times = []
    for idx, frame in enumerate(frames):
            …
    return detector_results, keypoint_counts, computation_times
```

Analysis:
Within each detection algorithm's processing loop, the feature detection function was initially being called multiple times, once for visualisation, again for matching, and again for benchmarking. This led to redundant computations and unnecessary code repetition. ChatGPT's recommendation to decouple feature detection from downstream tasks and store the results in a shared data structure allowed for more efficient reuse.

Fundamental Matrix and Pose Estimation - Matthew Privitera

1 Which matrix to compute for stage 5 - Gemini 2.5 Flash

Prompt:

“What are the main differences between a fundamental matrix and essential matrix in the context of structure from motion. Which one would be the most useful if the camera is not explicitly calibrated?”

Response:

“In the context of Structure from Motion (SfM), the fundamental matrix and essential matrix both describe the epipolar geometry between two camera views, but they differ significantly in their requirements and what information they encode: … If the camera is not explicitly calibrated, the Fundamental Matrix (F) is the most useful and often the only one that can be reliably computed directly from image correspondences.”

Analysis:

The provided explanation clarified the use cases of both the fundamental and essential matrices within the SfM pipeline. This initial knowledge paved the way for further exploration into the implications of camera calibration and how to implement it.

2 The chirality problem and how to resolve it - Claude 3.7

Prompt:

“Can you explain the chirality problem and how it can be addressed using the recoverPose() function?”

Response:

“The chirality problem is a fundamental ambiguity that arises when decomposing the essential matrix to recover camera pose. The term "chirality" comes from chemistry/biology, referring to "handedness" - the same concept applies here to 3D coordinate system orientation…

The solution is to triangulate 3D points using each of the four pose hypotheses and count how many points have positive depth (appear in front of both cameras)...

OpenCV's `recoverPose()` function automatically performs the chirality test:”

```
inliers, R, t, mask = cv2.recoverPose(E, pts1, pts2, K)
```

Analysis:

Knowing about the chirality problem and how the recoverPose() function works in relation to the previous stage gave me a general idea of how to implement it using Python, develop the code for this section and a theoretical baseline for the write-up.

3D Point Triangulation and Scene Visualisation - Matthew Vella

1 Metrics for Point Cloud Accuracy - GPT-4o
<u>*Prompt:*</u>
"What are the standard quantitative metrics used to assess the accuracy of a point cloud? Explain what it measures and what typically constitutes a 'good' versus a 'bad' result"
<u>*Response:*</u>
Here are some standard quantitative approaches commonly used to evaluate the geometric quality and accuracy of a point cloud:

1. ***Reprojection Error***
    - ***Purpose:*** *Measures the discrepancy between the original image points and the projections of the reconstructed 3D points back onto the images. It reflects the internal consistency between the images and the point cloud geometry in photogrammetric or SfM processes.*
    - ***Interpretation:***
        - *Good: Low reprojection errors (often below 1 pixel), indicating accurate camera calibration and feature alignment.*
        - *Bad: High reprojection errors, which may signal poor calibration, bad feature matches, or reconstruction errors.*
2. ***Scale Ratio Analysis …***
3. ***Point-to-Point Distance Error (Cloud-to-Cloud Distance) …***

<u>*Analysis:*</u>
This was very helpful for defining my evaluation criteria. The explanation of reprojection error and what constitutes a "good" vs. "bad" result was used directly to frame the analysis in my report.

2 Building Projection Matrices for Triangulation - GPT-4o
<u>*Prompt:*</u>
cv2.triangulatePoints requires 3x4 projection matrices. How do I form a projection matrix P given my 3x3 intrinsic matrix K, a 3x3 rotation matrix R, and a 3x1 translation vector t? I have corresponding points from two views and need to get their 3D coordinates.
<u>*Response:*</u>
To use `cv2.triangulatePoints`, you need two projection matrices (P_1) and (P_2), each of size (3 \times 4).
Given:
- The intrinsic matrix (K) (3x3),
- The rotation matrix (R) (3x3),
- The translation vector (t) (3x1), …
Form projection matrix:

```
P = K @ np.hstack((R, t.reshape(3,1))) …
```

After forming P1 and P2, triangulate:

```
points_4d_hom = cv2.triangulatePoints(P1, P2, pts1, pts2) …
```

Convert from homogeneous to 3D:

```
points_3d = points_4d_hom[:3, :] / points_4d_hom[3, :]
```

Analysis:
This response solved a key coding issue by providing the exact NumPy syntax for the projection matrix. The included step on how to convert the 4D output also prevented a likely error in my code.

Improvements, Errors, and Contributions

During video acquisition, AI provided valuable suggestions for suitable video content and scene selection criteria. AI recommended and helped implement imageio.v3 as the primary method for video loading rather than OpenCV's VideoCapture.

For feature detection and description, AI assisted with parameter tuning of detection algorithms. AI also enhanced qualitative analysis by creating montage videos. These videos allowed systematic evaluation of algorithm performance in tracking features across frames and identifying newly detected or discarded features.

AI helped generate matplotlib plots throughout the assignment for analyzing results and debugging purposes. However, AI occasionally forgot previously implemented functions and recreated duplicate code. This resulted in redundant implementations that required cleanup. AI assistance helped validate pose estimation results by implementing checks for rotation matrix orthogonality and translation magnitude constraints, which helped with validating pose estimates. However, the initial implementation of cv2.recoverPose() returned incorrect inlier masks.

For 3D point triangulation and scene visualisation, AI recommended and helped implement Open3D as the visualisation library. AI also suggested CloudCompare as an external analysis tool. When reconstruction quality was poor, AI helped diagnose whether issues stemmed from visualisation code errors or underlying problems in feature detection and pose estimation. AI also suggested and assisted with bundle adjustment implementation to enhance overall reconstruction accuracy.

When attempting to format IEEE references from research paper links, the AI consistently failed to generate properly-formatted references and required manual correction.

In conclusion, AI provided valuable assistance throughout the implementation and research process. However, its output still contained errors and inconsistencies that required careful review. AI-generated code and solutions still make mistakes that need verification and manual correction before use.

Individual Reflection

Daniela Curmi

Generative AI, specifically ChatGPT's GPT-4o model, was used to enhance the overall quality of this project. It helped improve the clarity, precision, and overall cohesion and coherence of the report by refining word choice and sentence structure. In the coding aspect, AI was used to develop standard plotting functions and generate feature detection montage videos, which helped visually and quantitatively evaluate the performance of the implemented algorithms. However, sometimes it made severe lapses in its judgment when providing coding solutions or interpreting complex requirements, occasionally generating incorrect or suboptimal approaches that required manual correction and further validation. Overall, AI served as an effective and efficient tool to support both code development and knowledge acquisition throughout this project.

Liam Azzopardi

Throughout this project, I expanded my understanding of how feature detection and parameter tuning contribute to the success of a SfM pipeline. By consistently seeking clarification on key concepts, such as the separation of keypoint detection and descriptor computation in ORB, or how to tune parameters of Shi-Tomasi and the other algorithms, I was able to reason about the impact of each parameter on the final 3D reconstruction and feature detection. This process helped me build both technical knowledge and the required thinking skills to evaluate and adjust methods to improve performance and reliability throughout this project. The use of generative AI didn't just aid in understanding certain concepts, but was very useful for efficient debugging feature detection code and also helped in understanding the reasoning behind certain algorithmic choices and understanding their limitations.

Matthew Pirotta

In my experience, generative AI has been a helpful tool for improving code structure and speeding up initial prototyping. It allowed me to test ideas quickly and better understand what's needed to build a functional system. However, I've found that relying too heavily on AI-generated code can lead to issues later, often requiring more time to debug and correct. Instead, I've learned it's more effective to write the core code myself and use AI for assistance with debugging, optimisation, or clarifying specific concepts.

Matthew Privitera

Overall, using generative AI improved my efficiency throughout this project. It accelerated my understanding of more complex computer vision concepts such as the difference between fundamental and essential matrices, the use of post estimation and how they interact with each other within the SfM pipeline. Most importantly, AI enabled rapid iteration because I could test different implementation approaches, debug issues and improve my code much faster without wasting any time on trial and error. It was very useful when learning new concepts as I could ask follow up questions and receive an immediate response. However, when I wanted an in-depth and technical explanation, I would seek other sources such as research papers or official documentation.

Matthew Vella

Generative AI served as a useful tool for both research and implementation tasks. It effectively explained concepts like triangulation theory and bundle adjustment, and generated boilerplate visualisation functions while helping debug specific errors. However, all AI-generated code required validation to ensure reliability and correct implementation. For debugging, AI was helpful with small, specific issues like interfacing with OpenCV and Open3D library functions, but struggled with complex, higher-level bugs that involved the entire pipeline, often arriving at incorrect conclusions. The efficiency gained from generative AI handling well-defined coding tasks allowed more time for analysing and interpreting reconstruction results.