
**TRIBHUVAN UNIVERSITY**

A Project Report on

"Blendify"

A Panorama Generator

Submitted by

Alisha Adhikari (26301/077)

Ishan Poudel (26316/077)

A project proposal submitted in partial fulfillment of the requirements for the degree of Bachelor of Science (B.Sc.) in Computer Science and Information Technology awarded by IOST, Tribhuvan University

National College of Computer Studies

Paknajol, Nepal

Dec,2024

# ACKNOWLEDGEMENT

We extend our heartfelt gratitude to everyone who supported us in completing this report. We are especially grateful to our project supervisor, Mr. Navaraj Negi, whose valuable suggestions and encouragement helped us immensely, particularly in coordinating and writing this report. We also wish to acknowledge the essential support of the staff at NCCS College, who provided us with access to necessary equipment and resources to complete this project. We feel fortunate to have received continuous guidance from our seniors and every teaching staff member of the B.Sc. CSIT Department, whose assistance enabled us to successfully conclude our project. Additionally, we express our thanks to all non-teaching staff in the CSIT Department for their timely support. We greatly appreciate the guidance given by other supervisors and the panel during our project presentation, which improved our presentation skills through their feedback and advice. Our appreciation extends to each of our colleagues for their encouragement and support throughout the project. This accomplishment would not have been possible without the kindness and help of many individuals, and we offer our sincere gratitude to all of them.

With respect,

Alisha Adhikari (26301/077)

Ishan Poudel (26316/077)

# ABSTRACT

The “Blendify” is a comprehensive effort to seamlessly integrate multiple images into panoramic views, executed in Python. While the project draws inspiration from established techniques like the Scale-Invariant Feature Transform (SIFT) for feature detection and Random Sample Consensus (RANSAC) for robust homography estimation, it explicitly utilizes the Levenberg-Marquardt optimization method. The primary objective is to achieve precise alignment, or homography, between consecutive images and extend this capability to create coherent panoramas from larger sets. The algorithm incorporates gain compensation and blending methods to enhance visual coherence and achieve a natural appearance in stitched panoramas. The implementation addresses various scenarios successfully, presenting a versatile and automated solution for panoramic image stitching. Acknowledging potential challenges, particularly in cases of excessive distortion or blur in high-frequency details, the project contributes significantly to the broader fields of computer vision and image processing.

**KEYWORDS:** *Panoramic Image Stitching, SIFT, RANSAC, Gain Compensation*

# TABLE OF CONTENT

# LIST OF ABBEREVIATIONS

| | |
|---|---|
| AKAZE | Accelerated-KAZE |
| API | Application Programming Interface |
| BF | Brute Force |
| CNN | Convolutional Neural Networks |
| CSS | Cascading Style Sheets |
| Env | Environment |
| FCM | Fuzzy C-Means |
| FOV | Field Of View |
| HTTP | Hypertext Transfer Protocol |
| JS | JavaScript |
| LM | Levenberg-Marquardt |
| NN | Nearest Neighbor |
| ORB | Oriented FAST and Rotated Brief |
| RANSAC | Random Sample Consensus |
| SI | Satellite Imagery |
| SIFT | Scale Invariant Feature Transform |
| SURF | Speeded-Up Robust Features |
| UAV | Unmanned Aerial Vehicle |
| UML | Unified Modeling Language |

# LIST OF FIGURES

# LIST OF TABLES

# CHAPTER 1 INRODUCTION

## 1.1. Introduction

The Blendify app in Python is a convenient tool that transforms ordinary photos into captivating panoramic views with ease. It requires no elaborate setup; users simply drop in their pictures, and the app works its magic. By incorporating smart ideas from various sources, Blendify ensures that anyone can effortlessly create impressive panoramic images without specialized skills.

In today's fast-paced world, where simplicity and speed are valued, the app focuses on quick image stitching and panorama generation. With a user-friendly interface, users can input their pictures, adjust settings as desired, and quickly obtain stunning panoramic images. The primary objective is to streamline the image stitching process, maintaining both speed and high-quality results, catering to both professionals and novices alike.

As the app delves into its workings in further detail, it demonstrates its clever integration of images, utilization of various algorithms like RANSAC and SIFT, and techniques to ensure exceptional output quality. Users can expect effortless transformation of their regular photos into breathtaking panoramas, with the app's straightforward image stitching capabilities. Additionally, the app conceals the background processes from users, preserving simplicity while delivering impressive results.

## 1.2. Problem Statement

Creating panoramic images can be tricky for many people due to existing tools requiring manual setup or the use of complicated models. These tools often demand technical knowledge, making it challenging for users who just want a straightforward way to transform their regular photos into stunning panoramas. Additionally, the need for precise setup or reliance on complex models limits the accessibility of panoramic image creation. The existing system requires training before using that software and are also not available for free of cost for the users. The lack of simplicity and an intuitive interface in current solutions further complicates the image stitching process, creating barriers for users looking for a hassle-free experience.

This project seeks to address these challenges by developing a Python based image stitching application that eliminates the need for manual setup and complex models. It aims to create

a user-friendly tool that caters to a diverse audience, providing a seamless experience for turning multiple images into impressive panoramas. This app focus on simplifying the image stitching process, ensuring speed, cost and efficiency there by overcoming the current barriers to entry in the domain of panoramic photography.

## 1.3. Objective

The project aims to meet the following objectives:

1. To develop a streamlined image stitching algorithm using, SIFT, RANSAC and Levenberg-Marquardt optimization method to efficiently combine multiple images into captivating panoramas, providing a wider field of view.
2. To implement intuitive features and functionalities within the application to enable users to effortlessly upload their images and generate panoramic views without the need for manual intervention or complex software.
3. To Implement adaptive image processing techniques to ensure a seamless transition between images, addressing challenges like exposure variations and vignetting effects, resulting in natural-looking and visually appealing panoramas.

## 1.4. Scope and Limitations

The scope of the image stitching project is multifaceted, covering algorithmic advancements and user interface design. Algorithmically, the project aims to extensively explore and optimize the SIFT, RANSAC and Levenberg-Marquardt optimization algorithms to improve the efficiency and accuracy of image stitching. This involves researching and incorporating advanced computer vision techniques for robust feature detection, matching, and homography estimation. On the user interface front, the project envisions the creation of an intuitive and user-friendly interface that is accessible to individuals with varying levels of expertise. The interface will facilitate effortless interaction, allowing users to seamlessly input their images and customize parameters to generate impressive panoramic views. By combining algorithmic sophistication with a user-centric design approach, the project aims to provide a comprehensive solution for panoramic image creation.

However, there are certain limitations to consider:

- Evaluation Challenge: A primary limitation lies in the absence of a standardized metric to evaluate the effectiveness of image stitching algorithms. Unlike classification or regression problems, where metrics provide clear assessments, image stitching relies on visual comparisons that may not universally capture algorithmic performance.
- User Dependency for Alignment: In scenarios involving multiple images, the algorithm may require approximate user input for proper alignment. This introduces a level of dependency on user interaction, potentially impacting the efficiency of the stitching process.
- Adaptation to Varying Image Characteristics: While the algorithm used adeptly handles the arrangement of images, adapting the appearance of each image to eliminate visible boundaries, challenges arise when dealing with variations in exposure, contrast, and effects like vignetting. Striking a perfect balance in such cases remains a complex task, presenting a limitation in achieving seamless transitions in all scenarios.
- Moving object in an Image: In the scenarios where object in image is moving then Blendify cannot generate panorama for such images.

## 1.5. Development Methodology

The development methodology used for the Blendify project is incremental delivery, which is a kind of iterative software development process. The project is divided into a series of incremental iterations. Each function or features are being added upon the previous one to gradually deliver a full-fledged functional system.

The incremental delivery approach is particularly suited for projects which involves a high level of complexity, or where the requirements are not clearly-defined or may change over the period of time. Breaking the project down into smaller iterations, allows for a more manageable and flexible development process.

Overall, the incremental delivery technique proved to be a successful strategy for creating the Blendify project, enabling a flexible and adaptive development process that finally resulted in the successful delivery of a useful and functional system.

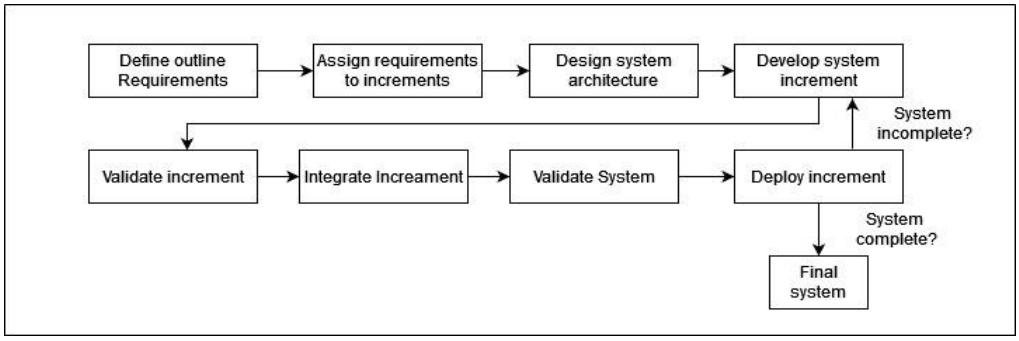



**Figure 1.1: Incremental Delivery**

## 1.6. Report Organization

This report is structured into six chapters, each serving a distinct purpose. The initial chapter encompasses the project's introduction, further subdivided into six sections. Within this segment, the problem statement, objectives, scope and limitations, and development methodology are elaborated. The second chapter delves into a comprehensive background study, exploring fundamental theories, general concepts, and a literature review that encompasses similar projects, theories, and findings by other researchers. Moving forward, the third chapter focuses on the analysis phase, covering system analysis, requirement analysis, and feasibility analysis. The fourth chapter is dedicated to outlining the overall system design. Subsequently, the fifth chapter details the system's implementation and testing, employing various tools. Finally, the last chapter encapsulates the conclusion and offers future recommendations pertaining to the current system.

# CHAPTER 2 BACKGROUND STUDY AND LITERATURE REVIEW

## 2.1. Background Study

Understanding the foundational principles behind image stitching and panorama creation using SIFT, RANSAC, and blending algorithms is essential for seamlessly combining multiple images into cohesive panoramas.

**SIFT**, or Scale-Invariant Feature Transform, is a powerful algorithm widely utilized for detecting and describing features in computer vision tasks. It identifies key point locations and generates descriptors that remain invariant to scale, rotation, and illumination changes. These descriptors facilitate the alignment of features across different images, crucial for accurate image stitching.

**RANSAC**, or Random Sample Consensus, is a robust estimation algorithm commonly employed for outlier rejection in geometric estimation problems. In image stitching, RANSAC plays a vital role in robustly estimating the transformation (e.g., homography) between image pairs, even in the presence of outliers or mismatches in key point correspondences. Through iterative sampling of key points and subsequent estimation of transformations, RANSAC identifies the optimal transformation model for effectively aligning images.

Blending algorithms are essential for seamlessly merging overlapping regions of adjacent images in panoramas. These algorithms ensure smooth transitions between images, preventing visible seams or artifacts. Techniques such as feathering, multi-band blending, and alpha blending are used to achieve natural-looking panoramas. The choice of blending algorithm depends on factors such as image content, lighting conditions, and desired output quality.

## 2.2. Literature Review

In the research conducted by Z.Qu [1], Jun Li, , the focus was on developing a user-friendly image stitching application. The application utilized practical algorithms such as SIFT, RANSAC, and a simple blending technique to seamlessly combine multiple images into a unified panorama. The SIFT algorithm played a pivotal role in feature identification and

alignment, while RANSAC contributed to the overall robustness of the stitching process. The simplicity of the app, as highlighted by the authors, catered to users prioritizing ease of use. This study forms a basis for understanding the practical aspects of image stitching, providing insights into algorithmic approaches for creating visually cohesive panoramic images. [1]

K.Joshi, [2] from the Electronics and Telecommunication department at Dr. D.Y. Patil Institute of Engineering, Management and Research in Pune, India, explores the field of image stitching in computer graphics and computer vision. Image stitching involves combining multiple images using various algorithms and techniques to create a panoramic view with a wide Field Of View (FOV) in a single, large, high-resolution frame through computer software. With the growing demand for panoramic views in diverse fields, such as medical image stitching, high-resolution photo mosaics, satellite photography, and telemedicine, the paper reviews recent approaches and advancements in image stitchi ng. Additionally, the paper addresses the challenges associated with image stitching, highlighting the complexities in this evolving area of research. [2]

Z.Zhang [3] research delves into image stitching technology, which overcomes camera angle limitations to achieve both high resolution and a broad field of view. The technology has wide applications in aerial photography, medicine, cultural relics protection, VR, and various industries. The study focuses on image registration, utilizing OpenCV library functions and examining feature point extraction algorithms. Notably, the approach employs oFAST + BRISK, with key point matching and filtering using the KNN and RANSAC algorithms. The final step involves splicing images through image projection. The method demonstrates robust rotation and scale invariance, showcasing adaptability across diverse environments. Zhang's research contributes valuable insights to image stitching advancements, particularly in addressing challenges related to feature extraction and key point matching. [3]

C.Yicheng [4] in the paper explores the implementation of a drone-based panorama stitching project. The study encompasses a meticulous investigation into the utilization of three key techniques: Scale-Invariant Feature Transform (SIFT), Fast Library for Approximate

Nearest Neighbors (FLANN), and Random Sample Consensus (RANSAC). The project is structured around a series of comprehensive steps, beginning with the capture of images using a drone equipped with a Raspberry Pi camera. This initial phase lays the groundwork for subsequent processing by acquiring high-quality aerial images necessary for panorama stitching.

Following image capture, Y.Chang delves into the process of basic panorama stitching, encompassing key stages such as feature detection, descriptor matching, and transformation estimation. This phase involves a detailed examination of SIFT features and their role in detecting keypoints across multiple images captured by the drone.The study then explores the utilization of FLANN, a fast and efficient library for approximate nearest neighbor search, which is instrumental in matching keypoints extracted from the drone-captured images. By leveraging FLANN's capabilities, Yicheng Chang investigates methods to expedite the feature matching process, thereby enhancing the efficiency of panorama stitching algorithms.

Furthermore, the paper delves into the application of RANSAC, a robust estimation algorithm, for accurately estimating transformation parameters between overlapping images. Yicheng Chang examines the effectiveness of RANSAC in handling mismatches and outliers, ensuring the robustness and accuracy of the panorama stitching process.

By integrating these techniques, Y.Chang develops a comprehensive panorama stitching algorithm tailored to images captured by drones. This integrated approach leverages the strengths of SIFT, FLANN, and RANSAC to produce high-quality stitched panoramas, suitable for various applications including mapping, surveillance, and cinematography. Overall, the study conducted by Yicheng Chang offers valuable insights into the utilization of SIFT, FLANN, and RANSAC techniques in drone-based panorama stitching. Through meticulous experimentation and analysis, the paper contributes to advancing the field of drone-based imaging and surveillance, paving the way for enhanced techniques and methodologies in the future. [4]

Mohammed et al. [5] in their paper titled "Automatic Panoramic Medical Image Stitching Improvement Based on Feature-Based Approach, address the critical need for accurate and efficient medical image processing, particularly in fields like scoliosis and rib cage analysis.

The importance of collecting and combining medical images with overlapping areas is emphasized, as it aids clinicians in making informed decisions based on comprehensive evidence.

The primary challenge addressed in the paper revolves around the limitations of traditional X-ray machines, which often produce narrow-field images that lack a complete view of the target area. To overcome this limitation, the authors propose a novel method for creating panoramic images by combining multiple X-ray images. Their approach relies on a featurebased methodology, specifically utilizing Circle (Oriented-FAST and Rotated-BRIEF) descriptors for feature extraction. The feature extraction process is facilitated by the Oriented-FAST and Rotated-BRIEF techniques, which enable rapid and accurate description of image features. By adopting BRIEF technology, the authors aim to achieve efficient feature representation while minimizing processing time.

The effectiveness of the proposed method is evaluated based on two key performance metrics: processing time and image quality. The authors emphasize the importance of obtaining high-resolution panoramic images within a short processing time, thereby ensuring timely and accurate medical diagnosis. Experimental results demonstrate the superiority of the proposed approach, particularly in terms of image quality and processing time. The utilization of ORB descriptors yields promising outcomes, as evidenced by the experimental findings. [5]

Simon et al. [6] in the paper introduces a novel matching pipeline to address challenges associated with feature matching in computer vision. The study focuses on enhancing the robustness and accuracy of feature matches, which are crucial for various real -world applications in image processing and computer graphics.

The conventional approach to feature matching relies on greedily searching for corresponding features in an image pair based on their descriptor distance. However, this method often leads to incorrect correspondences and suboptimal matching accuracies, posing significant challenges in practical scenarios. These challenges include fundamental matrix degeneration, matching ambiguities caused by repeated patterns, and rejection of initially mismatched features without further reconsideration.

To overcome these challenges, Simon et al [6]. propose a novel matching pipeline that integrates several innovative techniques. Firstly, they introduce iterative rematching, allowing mismatched feature points a further chance to be considered in later processing steps. By searching for inliers exhibiting the same homographic transformation per iteration, this approach enhances matching recall and precision. Furthermore, the authors utilize Delaunay triangulation of the matching set to minimize the effects of repeated patterns and implement focused matching. This technique enables them to concentrate on local image areas defined by the triangular mesh, thereby improving matching quality and reducing matching ambiguities.

Jean K et al. [7] In their paper titled "Image Mosaicing Applied on UAVs Survey, delve into the application of image mosaicing techniques in the context of Unmanned Aerial Vehicles (UAVs). The paper highlights the significance of UAV technology in advancing robotics, particularly in image processing applications for surveillance, monitoring, photogrammetry, and mapping.

The primary focus of the study revolves around the generation of panoramic images through the stitching of multiple aerial photographs captured by UAVs. This process is crucial for various applications such as surveillance mapping, search and rescue operations, 3D scene reconstruction, and vegetation monitoring. The authors discuss the challenges associated with image alignment and ghosting, which are common issues encountered in image mosaicing, particularly in UAV-based scenarios.

The paper provides an overview of different mosaicing techniques, categorizing them into direct methods and feature-based methods. Direct methods involve pixel-based approaches, while feature-based methods rely on identifying and matching distinct features across images. Notably, feature-based methods are preferred for UAV applications due to their flexibility and ability to capture accurate and distinctive features.

The authors discuss the importance of feature extraction and registration in the mosaicing process, highlighting key algorithms and methodologies such as Harris Corner Detector, SIFT, FAST, ORB, SURF, and BRISK. These algorithms play a crucial role in detecting and matching features across images, facilitating accurate alignment and stitching. Furthermore, the study delves into the stages of image stitching, outlining the process of feature

registration, transformation estimation, and warping or stitching. Various techniques such as Random Sample Consensus (RANSAC) are employed for robust estimation of transformation parameters, ensuring accurate alignment of images despite potential mismatches or outliers.

Additionally, the paper explores mesh-based alignment techniques as a complement to traditional feature-based methods, particularly addressing challenges related to parallax and ghosting effects. Algorithms such as As Projective As Possible (APAP), shape-preserving half-projective warp (SPHP), and optimal similarity transformation (AANAP) are discussed in detail, showcasing their effectiveness in mitigating distortion and improving alignment accuracy [7].

Mohsin et al [8] In their paper titled "Feature points-based image registration between satellite imagery and aerial images of agricultural land", address the challenge of registering satellite imagery (SI) and images captured from unmanned aerial vehicles (UAVs) in agricultural applications. With the increasing utilization of remote sensing technologies in precision agriculture, efficient image registration methods are essential for tasks such as crop monitoring and change detection.

The authors highlight the significant temporal, textural, and intensity differences between SI and UAV images, primarily due to the dynamic nature of agricultural crops over time. Traditional feature points extraction methods, including Scale Invariant Feature Transform (SIFT), Oriented FAST and Rotated BRIEF (ORB), and Speeded-Up Robust Features (SURF), often struggle to handle such variations and exhibit suboptimal performance in SI-UAV image registration. To address this challenge, the paper proposes a novel method called NN-BF, which combines the strengths of nearest neighbor (NN) and brute force (BF) descriptor matching strategies for SI-UAV image registration. The NN-BF method involves identifying corresponding feature point descriptor matches between SI-UAV images of the training set with overlap error. These matches are then further refined using NN and BF strategies when matching with the descriptors of SI and UAV images from the test set, respectively. Finally, the resulting matches are processed with RANSAC to remove outliers and estimate a homography for image registration.

Experimental evaluations conducted on an agricultural land image dataset demonstrate the effectiveness of the NN-BF method compared to conventional feature matching strategies.

The results show improvements in precision scores for SIFT, SURF, and ORB feature points with NN-BF, with SIFT achieving particularly better performance. On average, SIFT achieves 6.1% and 18.9% higher precision scores than SURF and ORB with NN-BF, respectively. Additionally, SIFT exhibits lower root mean square error compared to SURF and ORB with NN-BF, further validating the efficacy of the proposed method [8].

Zhenzhou et al. [9] In their article titled "Underwater Terrain Image Stitching Based on Spatial Gradient Feature Block, address the challenges associated with stitching underwater terrain images obtained from side-scan sonar data. These images, typically processed into strip-shaped fragments, lack conspicuous features and exhibit high levels of noise, posing difficulties for traditional image stitching methods.

Image stitching technology is crucial for generating wide-view, high-resolution panoramic images by combining multiple images captured from different angles, times, and locations. Commonly used feature-based registration methods, such as Scale Invariant Feature Transform (SIFT), Speeded-Up Robust Features (SURF), and Accelerated-KAZE (A-KAZE), have been employed in various studies for image stitching. However, these methods often encounter limitations in terms of memory usage, computational complexity, and accuracy, particularly when dealing with underwater terrain images characterized by subtle features and high noise levels.

Previous research efforts have explored different approaches to overcome the challenges of stitching side-scan sonar images. Some studies have focused on enhancing feature extraction accuracy and effectiveness through preprocessing techniques and the utilization of algorithms like SURF. Additionally, methods involving elastic mosaic algorithms and block registration have been proposed to achieve feature-level conformal mosaic of images.

To improve the efficiency and quality of underwater terrain image stitching, the authors present an improved algorithm based on spatial gradient feature blocks. The proposed method leverages the spatial gradient fuzzy C-Means (FCM) algorithm to partition the underwater terrain image into feature blocks, considering both spatial and gradient information for accurate segmentation. Subsequently, the A-KAZE algorithm is employed to match feature block information between reference and target images, followed by optimization using Random Sample Consensus (RANSAC). Finally, image fusion is

performed using global homography and an optimal seam-line method to enhance image overlay fusion accuracy and eliminate artifacts such as ghosting and shape warping [9].

# CHAPTER 3 SYSTEM ANALYSIS

## 3.1. System Analysis

Systems analysis is a process of studying a system or organization in order to understand its components, how they interact and how they can be improved. It is a holistic approach that looks at the system as a whole and identifies the relationships between its parts. The goal of system analysis is to identify problems and inefficiencies in the current system and to propose solutions for improvement.

### 3.1.1. Requirement Analysis

Requirement analysis is a critical phase in software development that involves gathering, analyzing, and documenting the needs and constraints of the project. In building our app, Blendify, we took a close look at what it needs to do and what could limit it. This step is called requirement analysis. We wanted to figure out the important things the app must have and consider any challenges we might face in creating it. This way, we made sure we understood everything the app should do and any issues we might encounter along the way.

#### 3.1.1.1 Functional Requirements

Functional requirements aim to provide the overview of how the system works. Here, the functionalities such as services, tasks and functions required for the system is shown. The functional requirements are: -

1. User Selects Photos for Image Stitching: The user starts by selecting multiple photos they want to stitch together. This is represented by the include relationship, indicating that this step is necessary before moving on to the next steps.
2. System Performs Stitching: Once the user has selected the photos, the system takes over and performs the image stitching process. This involves aligning and combining the images to create a single, panoramic image.
3. System Performs Blend Images: After stitching the images together, the system blends them to create a seamless transition between the images. This is represented

by the extend relationship, indicating that this step is optional and may not always be necessary.

4. System Checks for Errors: The system checks for any errors in the selected files or modes before proceeding with the stitching and blending process. This includes checking for the correct file format and ensuring that the selected images can be stitched together.
5. Display Blended Images: Once the stitching and blending process is complete, the system displays the final panoramic image to the user.

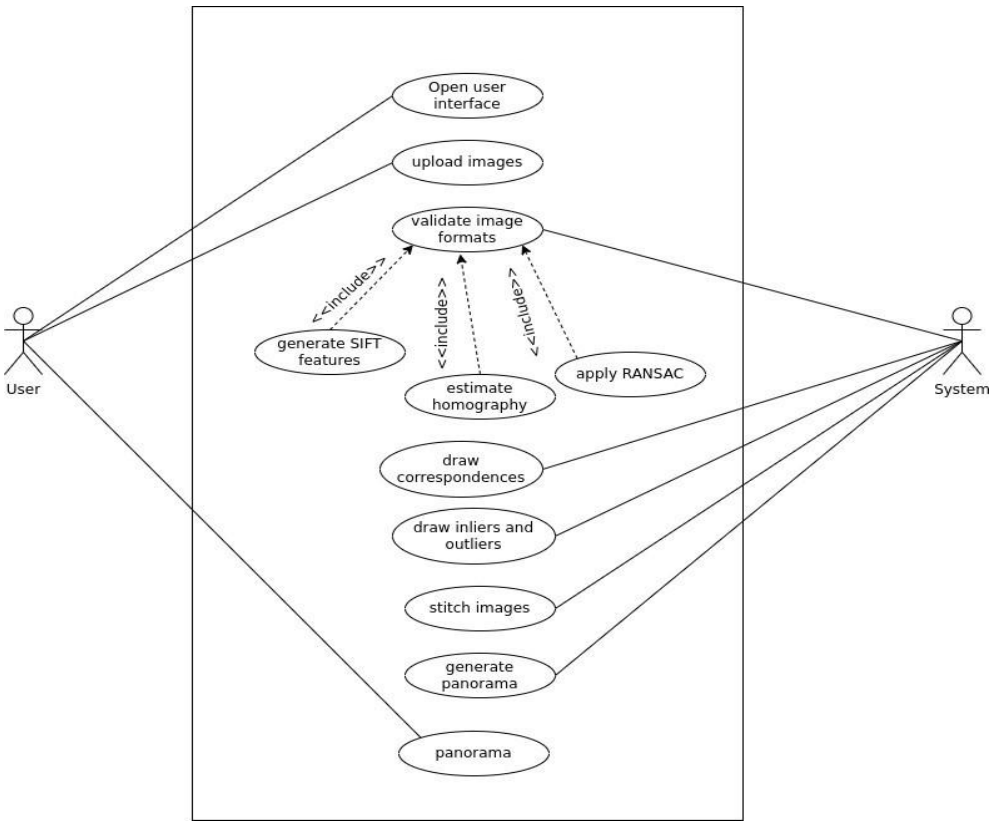



**Figure 3.1: Use Case for Blendify**

Figure 3.1 includes the system boundary, which represents the scope of the application and what is included within it. The actors in this diagram are the user and the system, with the user initiating the process by selecting photos and the system performing the stitching, blending, and error checking. The use case

diagram is a high-level representation of the functional requirements for your application, and it can be used to guide the development process and ensure that all necessary features are included.

#### 3.1.1.2 Non-Functional Requirements

The points below focus on the non-functional requirement of the system:

- Performance: The system should be highly performant and responsive, with minimal lag or delay when loading and displaying images.
- Usability: The system should be easy to use and navigate, with a clear and intuitive interface that appeals to a wide range of users.
- Scalability: The system should be scalable and able to handle large volumes of traffic and data, as the user base grows over time.
- Availability: The system should be highly available and able to handle high volumes of traffic without downtime or interruption, to ensure smooth user experience.

### 3.1.2. Feasibility Study

#### 3.1.2.1. Technical Feasibility

The technical feasibility of the project is high, as the required hardware and software resources are widely available and accessible. Requirements of our system can be categorized as:

**Hardware Requirements:**

- A computer with a minimum of 4 GB of RAM.
- Adequate storage space to store images and system files.
- Must be connected with reliable internet.

**Software Requirements:**

- Operating System: Windows 10, Linux, or MacOS
- Web Browser: Chrome, Firefox, or Safari

- Python 3.6 or above: This is required to run the Flask web framework and the various Python libraries used in the project, such as OpenCV, SciPy, NumPy and matplotlib.
- Flask web framework: This is required to build the web application and API for the system
- ReactJS: This is required to build the frontend of the web application and to create a responsive user interface.
- Git and GitHub: These are required for version control and collaboration among the development team.

In addition to the above requirements, some optional software may be used for development such as virtual environments like Anaconda or Virtual env to manage dependencies.

#### 3.1.2.2 Operational Feasibility

The operational feasibility of the panoramic image stitching project is considered high, thanks to its emphasis on user-friendly functionalities and the implementation of robust stitching algorithms. Users can easily navigate the application, which automates the process of seamlessly integrating multiple images into panoramic views. The success of the project relies on the efficiency of the stitching algorithms, ensuring accurate alignment between consecutive images. Regular maintenance and updates will be crucial to uphold the application's performance, aligning with evolving technological standards and user expectations.

#### 3.1.2.3 Economic Feasibility

The panoramic image stitching project demonstrates economic viability, with a justifiable initial investment for development and implementation. The user-friendly application holds significant market potential, offering revenue streams through app purchases, premium features, and potential advertising. Ongoing operational costs are manageable, primarily associated with maintenance and updates to stay competitive. Current market trends favor user-friendly photography tools, and the project anticipates a positive return on investment, making it economically feasible.

#### 3.1.2.4 Schedule Feasibility

The time given for the completion of this project was a whole semester. So, the project had enough time for completion. Since, the project has some machine learning mechanisms the project took some more time than done usually. Hence, the project has been developed according to the following time schedule to make our application schedule feasible:

**Table 3.1: Schedule Table**

| Task | Start Date | End Date | Duration |
|---|---|---|---|
| Requirement Gathering | 10/9/2024 | 18/9/2024 | 8 |
| Design & Development | 18/9/2024 | 10/10/2024 | 30 |
| Testing | 11/10/2024 | 22/10/2024 | 12 |
| Implementation | 22/10/2024 | 5/11/2024 | 14 |
| Documentation | 13/9/2024 | 10/11/2024 | 60 |

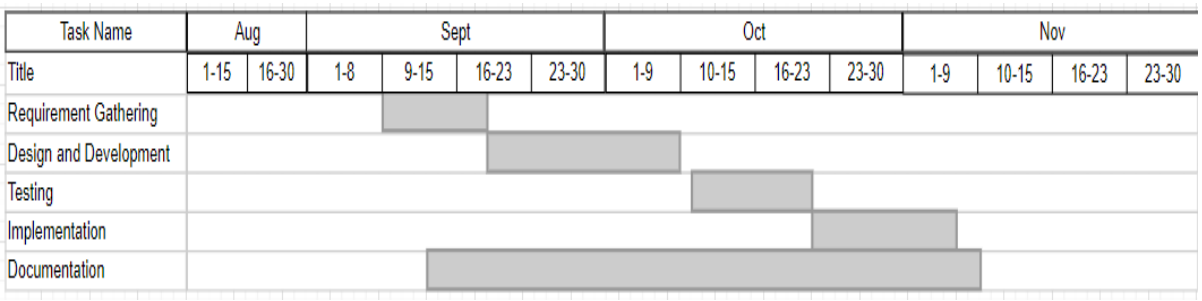



**Figure 3.2: Gantt Chart**

### 3.1.3. Analysis

#### 3.1.3.1 Object Modeling using class diagram

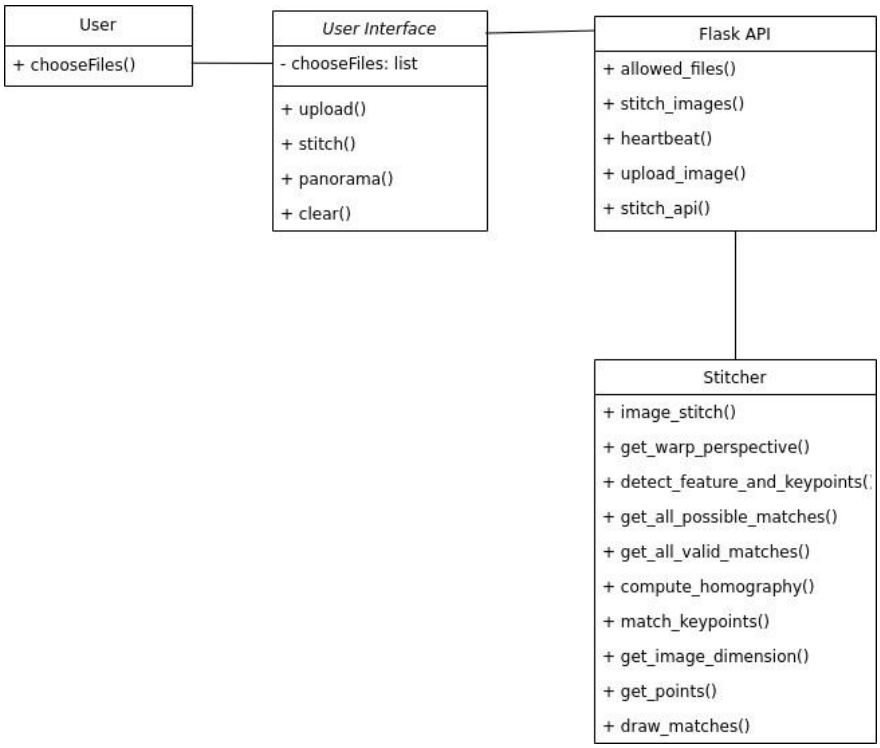



**Figure 3.3 Class Diagram of Blendify**

Figure 3.3 represents a collection of classes, functions, and methods that are used to create a panoramic image by stitching together multiple images. The user interacts with the system through the User Interface, which contains a choose_files () function that allows the user to select multiple images for stitching. The User Interface also has an upload () function for uploading the selected images.

The Flask API [1] is the backend server that handles HTTP requests. It contains several functions, including allowed_files(), which checks if the uploaded files are images, and upload_image(), which saves the uploaded images to a specified directory. The stitch_api() function takes the stitched image and sends it back to the user as an HTTP response.

The Stitcher class contains the primary functions for stitching images together. The image_stitch() function takes a list of images and stitches them together to create a

panoramic image. The get_warp_perspective() function calculates the perspective transform needed to map one image onto another. The detect_feature and keypoints() function detects features and keypoints in the images using the SIFT algorithm. The match_keypoints() function matches the keypoints between images to find possible correspondences. The get_all_possible_matches() function finds all possible matches between the keypoints. The get_all_valid_matches() function filters out the invalid matches based on distance and direction. The compute_homography() function computes the homography matrix that maps one image onto another.

The Stitcher class also contains several helper functions, including get_image_dimension(), which gets the width and height of an image, get_points(), which gets the coordinates of the matched keypoints, and draw_matches(), which draws the matched keypoints on the images.

### 3.1.3.2 Dynamic Modeling using sequence diagram

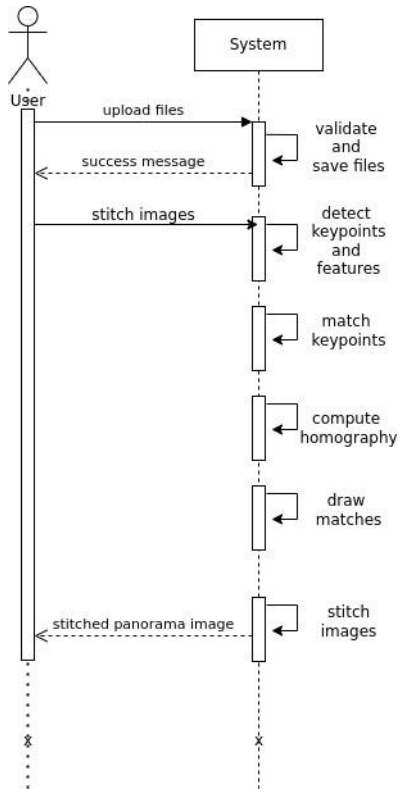



**Figure 3.4 Sequence Diagram of Blendify**

Figure 3.4 shows that the user initiates the process by selecting photos for stitching. The system then checks for errors in the file path or blending mode. If there is an error, the system displays the error message to the user and the process ends. If there is no error, the system performs image stitching and displays the stitched image to the user.

The diagram also shows that the system is responsible for performing image stitching and displaying the stitched image, while the user is responsible for selecting photos and receiving error messages. The system also checks for errors and displays them to the user.

### 3.1.3.3 Process Modeling using Activity Diagram

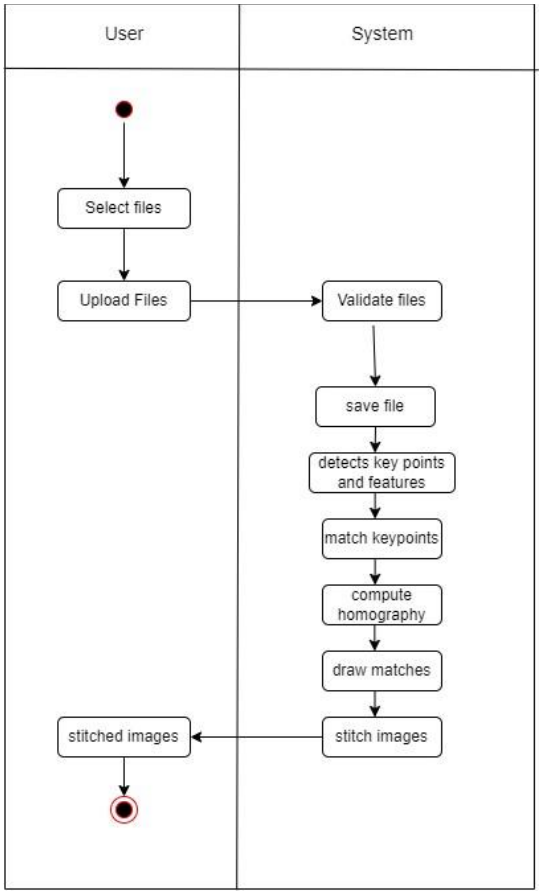



**Figure 3.5 Activity Diagram of Blendify**

Figure 3.5 shows the activity diagram of Blendify. This activity diagram represents the process flow of the image stitching app. The user starts the process by selecting images for stitching. The system then checks the file format of the selected images. If the file format is correct, the system performs image stitching and displays the stitched image to the user.
If the file format is not correct, the system displays an error message to the user.
Figure 3.5 shows that the user is responsible for selecting images, while the system is responsible for checking the file format, performing image stitching, and displaying the stitched image. The diagram also shows that the system checks the file format before performing image stitching, ensuring that only images with the correct file format are stitched together.

# CHAPTER 4 SYSTEM DESIGN

## 4.1 Design

System design is the process of representing architecture, interfaces, components that are included in the system. i.e., system design can be seen as the application of system theory to product development.

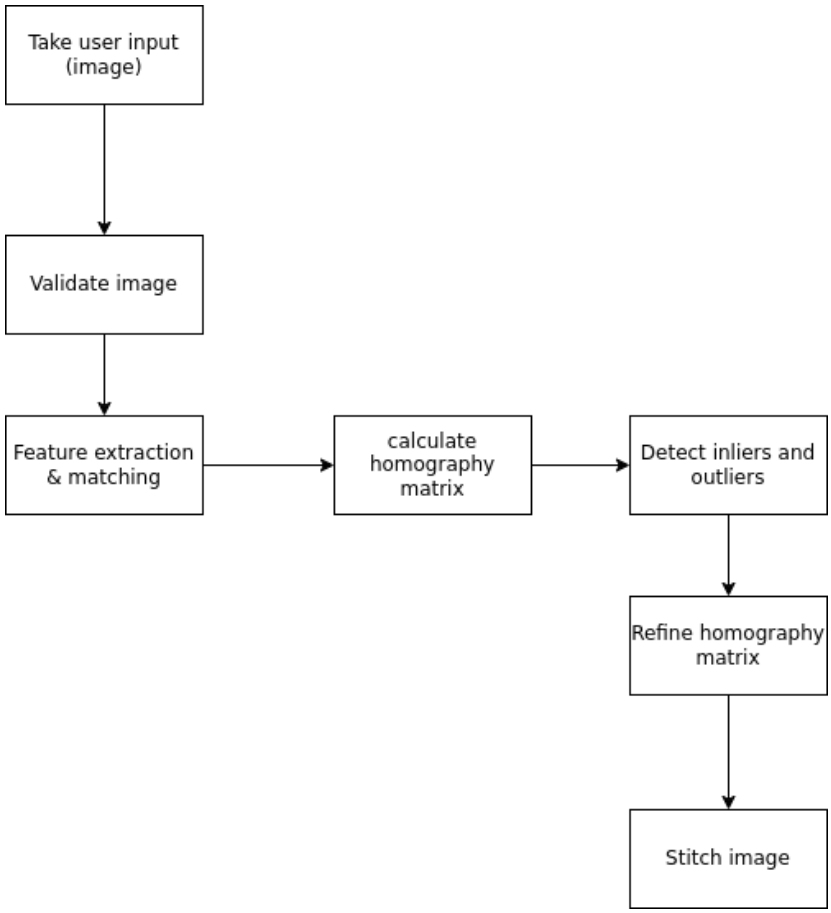



**Figure 4.1 High-Level System Design of Blendify**

- **Take user input(images):** Figure 4.1 demonstrates the process begins by accepting two or more input images but condition is that there must be some matching parts between the images i.e. there must exist some similarities between images.

- **Validate Image:** When image is uploaded then it is validated in order to check the format of image user cannot upload pdf or any other invalid format other than image format.
- **Feature Extraction:** The next step is to extract features from each input image. Features are distinctive points in an image that can be used to identify and match corresponding points between images. For feature extraction methods SIFT (Scale- Invariant Feature Transform) [2] is used. It helps to find common matching parts in images and plot the point in image Which facilitates in computation of homography matrix. It finds two pair which lies in same coordinate in both image or two point which are similar but lies in different coordinates.

  $\theta(x, y) = \tan{-1}\ [\ L(x, y+ 1) - L(x, y- 1)/L(x+1, y)-L(x-1,y)]$: ……. (i)

  This equation computes the orientation angle at each keypoint location $(x,\ y)$ based on the gradients of the image intensities in the x and y directions. This angle is an essential descriptor for keypoints.

  $D(x, y, \delta) = [G\ (x, y, k\delta) - G\ (x, y, \delta)]^{*}\ I\ (x, y)$: …… (ii)

  This equation calculates the difference of Gaussian (DoG) for each pixel location $(x,\ y)$ and scale level $\delta$. DoG is a key component in the SIFT algorithm for detecting scale-invariant keypoints.

  $L(x, y, \delta) = G(x, y, \delta)^{*}\ I\ (x, y)$: ……(iii)

  This equation computes the blurred image at scale level $\delta$ using a Gaussian kernel. Blurred images at different scales are used to detect keypoints across multiple scales.
- **Feature Matching:** Once features have been extracted from each image, the next step is to match corresponding features between the images. This is done by comparing the features extracted from one image to the features extracted from the other image and identifying matching points inlier and outlier are detected here and outlier are removed using RANSAC [3].

- **Calculate Homography matrix:** After matching corresponding features between the images, the next step is to calculate Homography matrix. It is calculated to take image and wrap to coordinate frame of another image. Homography matrix computes the geometrical relationship between the images i.e., transformation like rotation, scaling, translation in order to wrap the one image to another.

  The points $(x_i,y_i)$ for source and $(u_i,v_i)$ for destination the Homography is computed as:

$$\begin{bmatrix} ui \\ vi \\ 1 \end{bmatrix} = H * b \begin{bmatrix} xi \\ yi \\ 1 \end{bmatrix} \quad \text{(iv)}$$

  AH = b, where:

  *A* is a matrix constructed from the source points.

  *H* is the homography matrix.

  *b* is a vector constructed from the destination points.

- **Detect inlier and outliers:** Inlier are common features between image and lies in same coordinate whereas outlier are the common features which lies in different coordinates. RANSAC helps to calculate inlier and outlier then with the help of it Homography Matrix can be optimized for better result.

$$K = \frac{\log(1-p)}{\log(1-w^n)} \quad \text{(v)}$$

- **Refine Homography matrix:** When RANSAC computes the inlier and outlier then homography matrix is optimized for better result.
- **Stitch image:** Then finally on the basis of computed homography transformation is made to wrap the image in coordinate frame of another image this is what stitching is.

### 4.1.1. Refinement of Class Diagram

Figure 4.2 is the detailed formed of the class diagram with the addition of precise definition of each attribute and operations of each class.

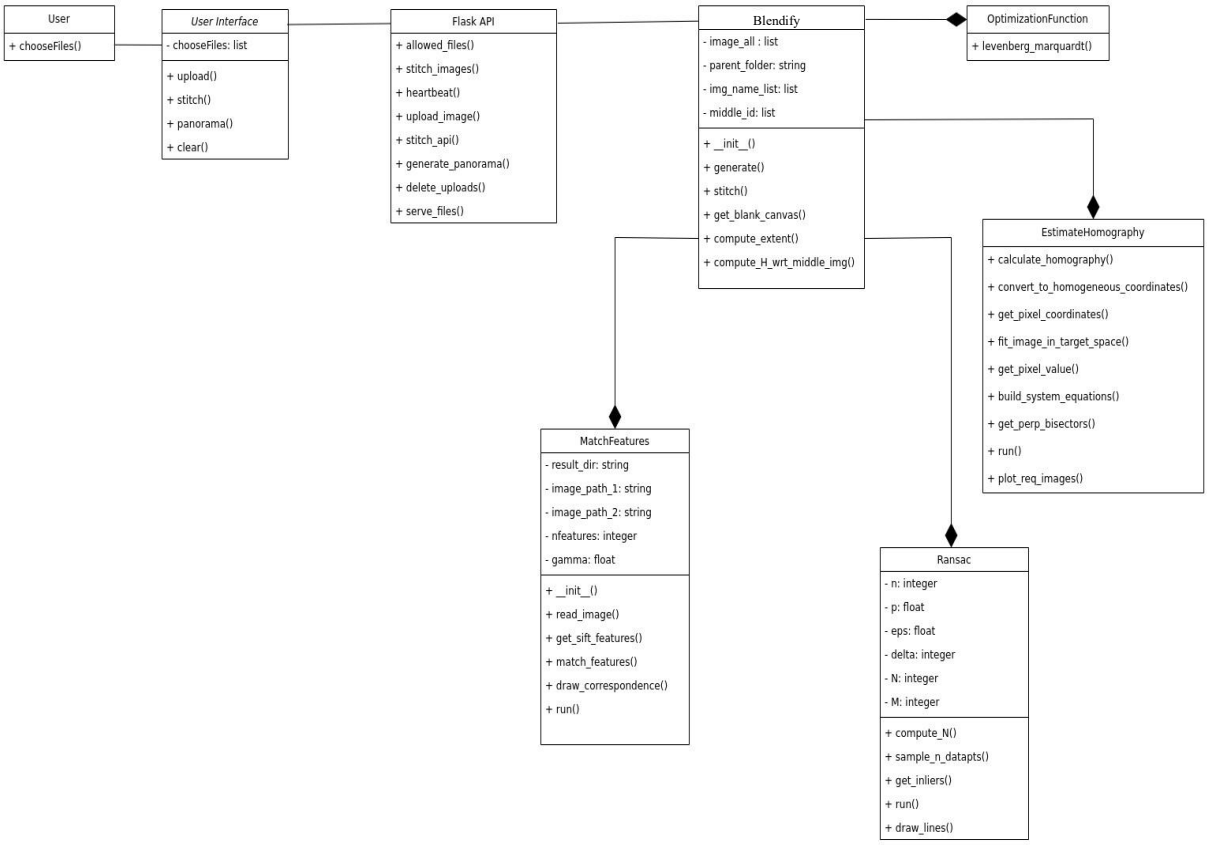



**Figure 4.2 Refined Class Diagram of Blendify**

Figure 4.2 outlines a system for creating panoramic images from multiple stitched images. The User class facilitates image selection, while the User Interface class manages image uploading, stitching, panorama generation, and clearing. The Flask API class handles HTTP requests, featuring functions for file checks, image stitching, server status, and image serving.

The Blendify class is central to image stitching, with functions for stitching, canvas creation, extent computation, and homography matrix calculation. The Match Features class focuses on feature matching, including SIFT feature detection and correspondence drawing. RANSAC manages robust homography matrix estimation, while the Optimization Function class handles matrix optimization using the Levenberg-Marquardt algorithm. The Estimate Homography class is responsible for homography matrix estimation with various supporting functions.

### 4.1.2. Refinement of Sequence Diagram

Figure 4.3 gives a thorough perspective of the interactions between the components in a system. It displays the order in which these interactions take place as well as the sequence of messages sent and received by various system components or objects.

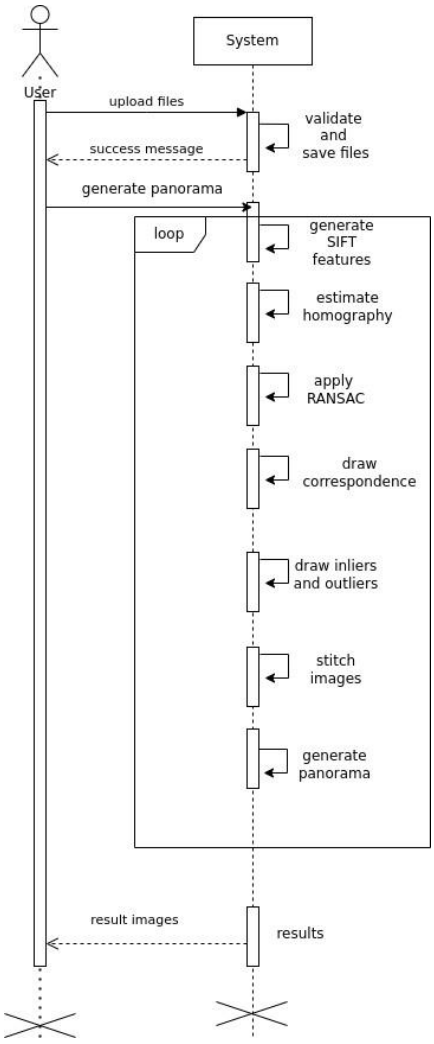



**Figure 4. 3 Refined Sequence Diagram of Blendify**

Figure 4.3 shows the interaction between the user and the system for creating a panoramic image by stitching together multiple images. The user starts by selecting files to upload, and the system validates and saves the files. The system

then generates SIFT features for each image and estimates the homography matrix using the matched features. The system applies RANSAC to the homography matrix to get the inliers and outliers, and then stitches the images together. Finally, the system generates the panorama and displays the results to the user.

The user interacts with the system through the user interface, which allows the user to upload files and generate a panoramic image. The system validates and saves the uploaded files, and then generates SIFT features for each image. The system estimates the homography matrix using the matched features and applies RANSAC to get the inliers and outliers. The system then stitches the images together and generates the panorama. The system displays the results to the user, who can view the stitched images and the panorama. The loop indicates that the user can repeat the process to create additional panoramic images.

### 4.1.3. Refinement of Activity Diagram

Figure 4.4 a more thorough illustration of the activities and steps involved in the process or workflow. By adding extra information and processes to the basic activity diagram, it expands upon it and enables a more in-depth understanding of the activities.

Figure 4.4 shows the process of creating a panoramic image by stitching together multiple images. The user starts by selecting files to upload, and the system validates and saves the files. The system then generates SIFT features for each image and estimates the homography matrix using the matched features. The system applies RANSAC to the homography matrix to get the inliers and outliers, and then stitches the images together. Finally, the system generates the panorama and displays the results to the user.

The user selects files to upload, and the system validates and saves the files. The system then generates SIFT features for each image and estimates the homography matrix using the matched features. The system applies RANSAC to the homography matrix to get the inliers and outliers, and then stitches the images together. The system generates the panorama and displays the results to

the user. The loop indicates that the user can repeat the process to create additional panoramic images. The decision indicates whether the user wants to continue or exit the process.

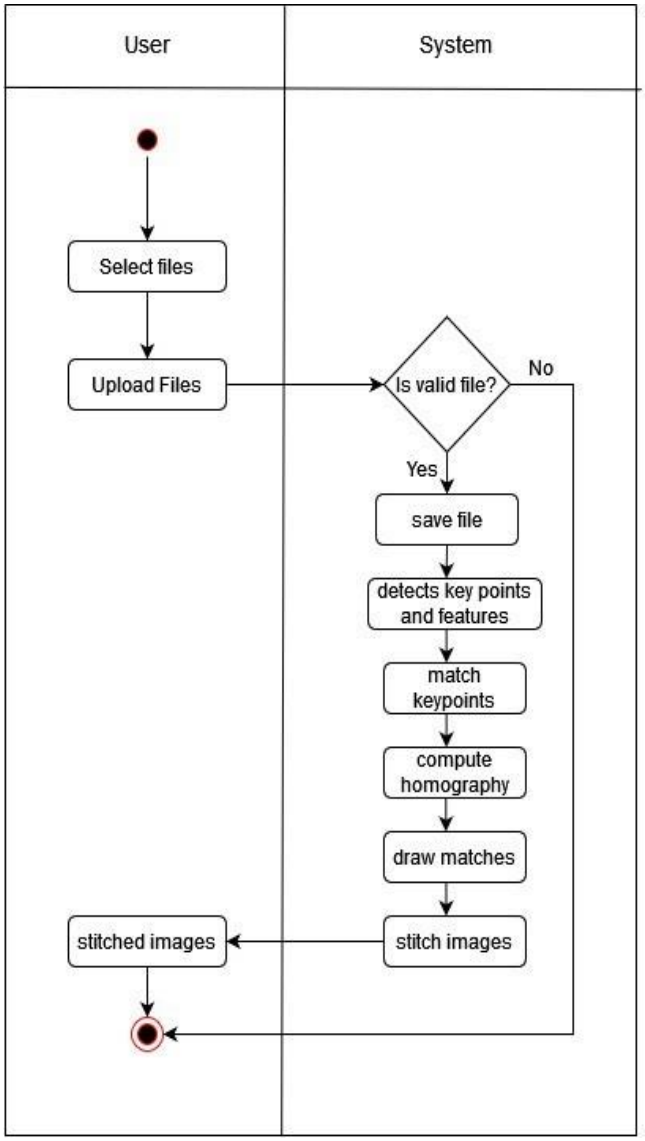



**Figure 4.4 Refined Activity Diagram of Blendify**

### 4.1.4. Component Diagram

Figure 4.5 referred to as a UML component diagram, describes the organization and wiring of the physical components within a system. Figure 4.5 consists of components, interface, and ports. The main purpose of the figure 4.5 is to show the relationship between the varying components in the system.

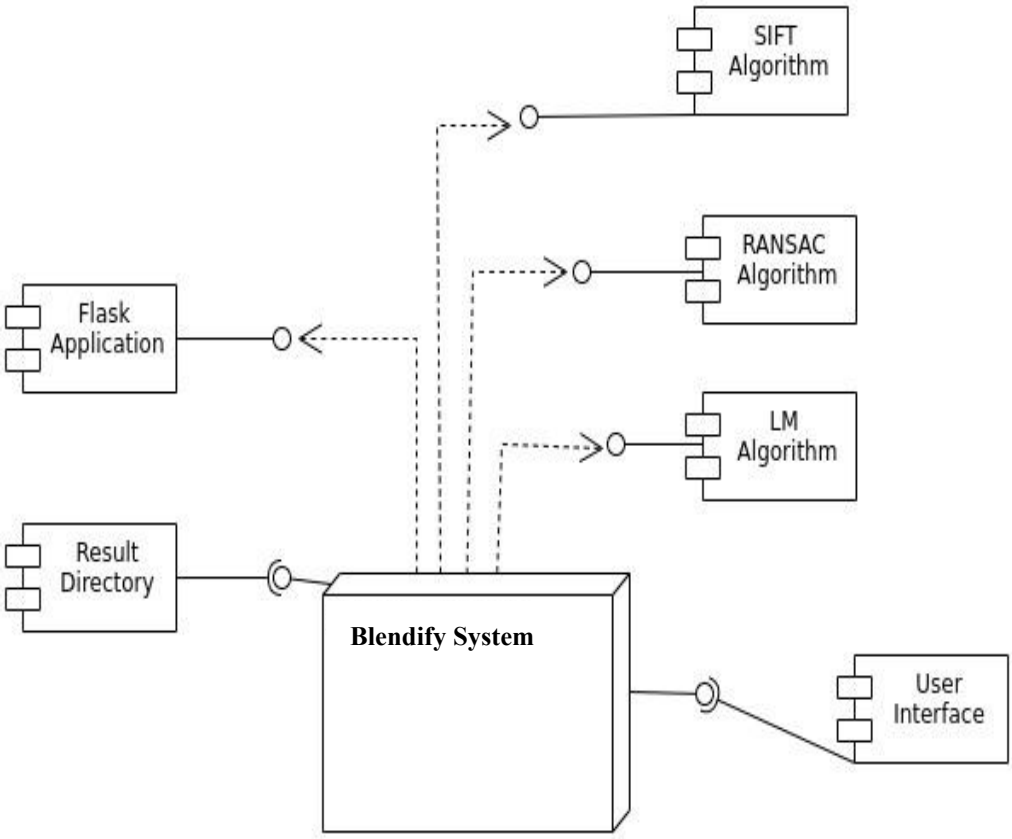



**Figure 4.5 Component Diagram of Blendify**

Figure 4.5 illustrates the process of stitching two images together to create a panoramic image. The process begins with the "Accept Multiple Images" component, which takes in two images as input. These images are then passed to the "Feature Extraction" component, which identifies and extracts distinctive features from each image. These features are then matched between the two images in the "Feature Matching" component. Once the features have been matched, the "Stitch Image" component takes over and aligns the images based on the matched features. This involves transforming one image so that its features align with the features of the other image. The transformed image is then blended with the original image to create a single, seamless image. The "Feature Extraction" and "Feature Matching" components are essential to the

image stitching process. By identifying and matching features between the two images, the system can accurately align and blend them together. The "Stitch Image" component is responsible for creating the final panoramic image by combining the two input images.

### 4.1.5. Deployment Diagram

Figure 4.6 shows the execution architecture of a system that includes nodes such as hardware or software execution environments and the middleware connecting them.

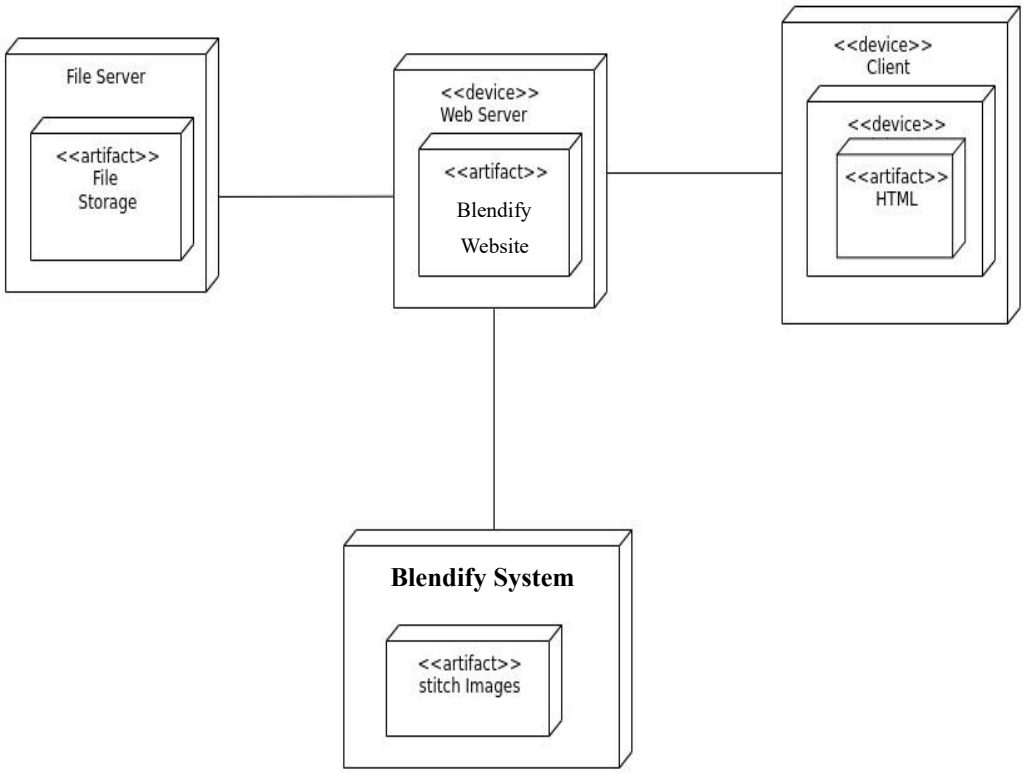



**Figure 4. 6 Deployment Diagram of Blendify**

Figure 4.6 represents the deployment architecture of an application that includes a desktop application, a web server, an API server, and a database server. The desktop application is installed on the user's computer and communicates with the web server to access the application's functionality. The web server hosts the application's front-end code and communicates with the API server to access the

application's back-end functionality. The API server is responsible for handling requests from the web server and communicating with the database server to retrieve and store data. The database server stores all of the application's data and is accessed by the API server as needed. The database server is typically a separate physical machine or a cloud-based service that is optimized for storing and retrieving data.

### 4.2. Algorithm Details

Blendify uses the following algorithms:

**SIFT:**

The Scale-Invariant Feature Transform (SIFT) [2] algorithm is a key component in image stitching, providing a robust method for detecting and matching distinctive key points across multiple images. SIFT's ability to handle variations in scale, rotation, and illumination makes it well-suited for identifying corresponding features in overlapping regions of images, facilitating accurate alignment. By describing key points based on local image gradients, SIFT contributes to the generation of seamless panoramic images by establishing correspondences and enabling the estimation of geometric transformations between different image frames.

Generating scale space and detecting extreme points: Two-dimensional image of a scale space of the image may be obtained with a Gaussian convolution kernel, namely:

$$L(x,y,\sigma)=G(x,y)*I(x,y) \quad \text{.....(i)}$$

$$G(x,y,\sigma)= \frac{1}{2\pi\sigma^2} \quad \text{.....(ii)}$$

**RANSAC:**

The Random Sample Consensus (RANSAC) [3] algorithm plays a pivotal role in image stitching by robustly estimating transformation parameters between image pairs. RANSAC excels at handling outliers, which commonly arise in

feature matching scenarios. In the context of image stitching, RANSAC iteratively selects random subsets of feature correspondences, estimates a transformation model, and then evaluates its consistency with the entire set of correspondences. This process is repeated to find the best-fitting model while minimizing the impact of outliers. In image stitching applications, RANSAC helps filter out mismatches and identify a reliable transformation, contributing to the creation of seamless panoramic compositions by aligning and blending images effectively.

Probability of choosing an inlier (w): w = (number of inliers in data)/(number of points in data) Probability that all n selected points are inliers ($w^n$) Probability that at least one of the n points is an outlier ($1– w^n$) Probability that the algorithm never selects a set of n points which are all inliers ($(1– w^n)^k$) Probability of the algorithm not resulting in a successful model estimation (1 - p) Formula to calculate k, the number of iterations:

$$K= \frac{\log(1\text{-p})}{\log(1\text{-}w^n)} \quad \text{……..……. (iii)}$$

Standard Deviation of k $SD(k)=\sqrt{\frac{(1\text{-}w^n)}{w^n}}$

**Levenberg-Marquardt Optimization:**

The Levenberg-Marquardt (LM) algorithm is an iterative numerical optimization method commonly used for solving nonlinear least squares problems. Its primary application is in finding the parameters that minimize the sum of the squares of the differences between the observed and predicted values in a system of nonlinear equations. Developed independently by Kenneth Levenberg and Donald Marquardt, this algorithm combines the strengths of the Gauss-Newton method and gradient descent.

Given a nonlinear model function where f(x, θ) x is the input and θ are the parameters to be optimized, and a set of observed data points yi, the goal is to minimize the sum of squared Residuals: $S(\theta)=\sum_{i=1}^{n}[yi - f(xi,\theta)]^2$……(v)

The LM algorithm iteratively adjusts the parameters θ to minimize this sum. It starts with an initial guess for θ and then updates it iteratively. The update rule combines elements of the Gauss-Newton method and gradient descent.

# CHAPTER 5 IMPLEMENTATION AND TESTING

## 5.1. Implementation

The incremental development model is a method for developing software products that involves designing, implementing, and testing the final product one by one. When the requirements are known before the product is developed and can be implemented in stages, the incremental development technique is used.

For our project, the incremental approach includes gradually updating the model with updated data over time to increase its accuracy. This can be accomplished either by using different approach, model as well as new algorithms. In order to increase the apps accuracy, refinement of algorithms is done. Over time, the incremental approach can produce images increasingly accurate by continuously refining the algorithms.

### 5.1.1. Tools Used

**a. Front End Tools**

The front end of Blendify is built with ReactJs, a popular and versatile library for building web applications. Leveraging ReactJs efficiency in developing user interfaces, Blendify ensures a seamless and responsive experience for users. With addition to ReactJS, Tailwind CSS was used for the basic styling and structure of the webpage. Furthermore, Axios was used to make HTTP requests from the frontend to the backend.

**b. Back End Tools**

Blendify's backend, which handles server-side tasks, is built using the Python programming language along with the Flask framework. Python was selected for its versatility, readability, and extensive libraries that support object detection. adds a modern touch, making the backend speedy and efficient, ensuring quick responses for handling request and image processing.

## 5.2 Module Description

This project implements the following modules:

- **Estimate Homography Module**

The Estimate Homography Module comprises a collection of functions tailored for image alignment and stitching tasks.

```
def calculate_homography(in_pts, out_pts):
    """
    Calculate the homography matrix that maps points from the source image to the destination image.
    """
    mat_A, mat_b = build_system_equations(in_pts, out_pts)
    H = np.matmul(np.linalg.pinv(mat_A), mat_b)
    H = np.reshape(np.hstack((H, 1)), (3, 3))
    return H

def convert_to_homogeneous_coordinates(inp, axis=1):
    """
    Convert a set of 2D points to homogeneous coordinates.
    """
    r, c = inp.shape
    if axis == 1:
        out = np.concatenate((inp, np.ones((r, 1))), axis=axis)
    else:
        out = np.concatenate((inp, np.ones((1, c))), axis=axis)
    return out

def get_pixel_coordinates(mask):
    """
    Extract the pixel coordinates of white pixels in a binary mask as homogeneous coordinates.
    """
    y, x = np.where(mask)
    pts = np.concatenate((x[:, np.newaxis], y[:, np.newaxis], np.ones((x.size, 1))), axis=1)
    return pts

def fit_image_in_target_space(img_src, img_dst, mask, H, offset=np.array([0, 0, 0])):
    """
    Fit the source image into the destination image within the specified mask region using homography.
    """
    pts = get_pixel_coordinates(mask)
    pts = pts + offset
    out_src = np.matmul(H, pts.T)
    out_src = out_src / out_src[-1, :]
    out_src = out_src[0:2, :].T
    pts = pts[:, 0:2].astype(np.int64)
    get_pixel_value(img_dst, img_src, pts, out_src, offset)
    return img_dst
```

**Figure 4.7 Estimate Homography Module**

Figure 4.7 contains `calculate_homography` which computes the homography matrix mapping points from the source image to the destination image, relying on a set of corresponding points for estimation. 'convert_to_homogeneous_coordinates' facilitates the conversion of 2D points to homogeneous coordinates. 'get_pixel_coordinates' extracts pixel coordinates from a binary mask, indicating the region in the destination image for fitting the source image.

The `fit_image_in_target_space` function fits the source image into the destination image within the specified mask region, utilizing the calculated homography. `get_pixel_value` interpolates pixel values from the source to the destination image based on the homography and mask region. `build_system_equations` constructs equations for homography matrix

calculation, while `get_perp_bisectors` extends corresponding points with perpendicular bisectors for non-aligned regions.

The main entry point, `run`, orchestrates image fitting by loading source and destination images, generating masks, calculating homographies, and performing image transformations. `plot_req_images` aids in visualization and debugging by displaying images, masks, and regions of interest.

- **RANSAC Module:**

The Random Sample Consensus (RANSAC) [3] Module plays a pivotal role in image stitching by robustly estimating transformation parameters between image pairs. RANSAC excels at handling outliers, which commonly arise in feature matching scenarios. Figure 4.8 defines a RANSAC (Random Sample Consensus) class for robust estimation of a homography matrix. The class takes in parameters p, eps, n, and delta in its constructor. The compute_N method calculates the number of iterations required for RANSAC to achieve a desired probability of success. The sample_n_datapts method randomly selects n data points from a total of n_total points. The get_inliers method calculates the inliers and outliers based on a given homography matrix and a set of correspondences. The run_ransac method performs RANSAC to estimate the homography matrix and returns the inliers and outliers. The code has some syntax errors and missing functions, such as convert_to_homogenous_crd and calculate_homography, which need to be defined for the code to run properly.

```python
class RANSAC:
    def __init__(self, p=0.99, eps=0.6, n=6, delta=3):
        self.n = n
        self.p = p
        self.eps = eps
        self.delta = delta
        self.N = self.compute_N(self.n, self.p, self.eps)

    def compute_N(self, n, p, eps):
        N = np.round(np.log(1 - p) / np.log(1 - (1 - eps) ** n))
        return N

    def sample_n_datapts(self, n_total, n=6):
        idx = np.random.choice(n_total, n, replace=False)
        n_idx = np.setdiff1d(np.arange(n_total), idx)
        return idx, n_idx

    def get_inliers(self, H, pts_in_expected, delta):
        pts_in = convert_to_homogenous_crd(pts_in_expected[:, 0:2], axis=1)
        est_pts = (H @ pts_in.T).T / (H @ pts_in.T)[-1, :]
        dst = np.linalg.norm(est_pts[:, 0:2] - pts_in_expected[:, 2:], axis=1)
        inliers = pts_in_expected[dst <= delta]
        outliers = pts_in_expected[dst > delta]
        return inliers, outliers

    def run_ransac(self, correspondence):
        if isinstance(correspondence, list):
            correspondence = np.array(correspondence)

        n_total = correspondence.shape[0]
        self.M = (1 - self.eps) * n_total
        no_iter = 0

        current_inliers = []
        current_inliers_cnt = 0
        current_sample_pts = []
        current_outliers = []

        while no_iter <= self.N:
            idx, n_idx = self.sample_n_datapts(n_total, self.n)
            sample_pts = correspondence[idx]
            other_pts = correspondence[n_idx]
            H = calculate_homography(in_pts=sample_pts[:, 2:], out_pts=sample_pts[:, 0:2])
            inliers, outliers = self.get_inliers(H, other_pts, delta=self.delta)
            inlier_count = inliers.shape[0]

            if (inlier_count > self.M) and (inlier_count > current_inliers_cnt):
                current_inliers = inliers
                current_outliers = outliers
                current_inliers_cnt = inlier_count
                current_sample_pts = sample_pts

            no_iter += 1
```

**Figure 4.8: RANSAC Module**

- **Levenberg-Marquardt Optimization Module:**

The Levenberg-Marquardt (LM) algorithm is an iterative numerical optimization method commonly used for solving nonlinear least squares problems.

```python
# Optimize the homography using Levenberg-Marquardt optimization
x = np.concatenate((inliers, sample_pts), axis=0)
opt_obj = OptimizeFunction(fun=fun_LM_homography, xO=final_H.flatten(),
                           args=(x[:, 0:2], x[:, 2:]))
LM_sol = opt_obj.levenberg_marquardt(delta_thresh=1e-24, tau=0.8)

H_all[key] = LM_sol.x.reshape(3, 3)
H_all[key] = H_all[key] / H_all[key][-1, -1]
```

**Figure 4.9: Levenberg-Marquardt Optimization Module**

Initially, RANSAC is employed to estimate homography matrices between pairs of consecutive images, effectively filtering out outlier correspondences and providing an initial homography estimate. Subsequently, the Scale- Invariant Feature Transform (SIFT)

is utilized for feature detection and matching between image pairs. Following the RANSAC step, the LM optimization comes into play to further refine the homography matrices. The code utilizes the Optimize Function class for LM optimization, specifying the objective function representing the difference between observed and predicted feature points (fun_LM_homography) and its Jacobian matrix (jac_LM_homography). The initial guess for the optimization is the homography matrix obtained from RANSAC. The refined homography matrices are then used to stitch the images together, creating a seamless panoramic mosaic. Throughout the process, visualizations of intermediate results, such as inlier and outlier correspondences, aid in analysis and debugging. Overall, LM optimization enhances the accuracy of homography matrices, contributing to the successful generation of a visually cohesive mosaic from the input image sequence.

## 5.2. Testing

Testing is the process of evaluating and verifying whether the developed software or application works properly or not i.e., whether there is match between the actual results and expected results or not. Testing is carried out during the development of the software.

### 5.2.1. Unit Testing

Unit testing is a part of the testing methodology which includes testing of individual software modules as well as the components that make up the entire software. The purpose is to validate each unit of the software code so that it performs as expected

**Table 5.1: Test Cases for Input Validation**

| S.N | Test Cases | Features | Test Description | Steps To Execute | Expected Result | Remarks |
|---|---|---|---|---|---|---|
| 1. | TC-I-1 | User Interface | Check all the Buttons | Check pages | UI Should be able to handle photo upload. | Pass |

| | | | | | | |
|---|---|---|---|---|---|---|
| 2. | TC-I-2 | Required Image Field | Try to generate image without image file. | Not uploading image | Ui should throw error. | pass |
| 3. | TC-I-3 | File format | Try to upload invalid file format | Uploading pdf | Ui should throw error. | Pass |

**Table 5.2: Test Cases for Performance**

| S.N | Test Cases | Features | Test Description | Steps To Execute | Expected Result | Remarks |
|---|---|---|---|---|---|---|
| 1. | TC-P-1 | Waiting Period | System should work normally under peakload. | Check loading time. | Generate panorama image in minimum time. | Pass |
| 2. | TC-P-2 | Waiting result. | System should perform calculation efficiently. | Checking execution time | Perform calculation in minimum time | pass |

### 5.2.2. System Testing

System testing is a black box testing method used to evaluate the completed and integrated system, as a whole, to ensure it meets specified requirements. The functionality of the software is tested from end-to-end and is typically conducted by a separate testing team than the development team before the product is pushed into production.

**Table 5.3: Table For System Testing**

| S.N | Test Cases | Features | Test Description | Steps To Execute | Expected Result | Remarks |
|---|---|---|---|---|---|---|

| | | | | | | |
|---|---|---|---|---|---|---|
| 1. | TC-ST-1 | Checking Homograph y coordinate. | System should calculate coordinate accurately. | Check calculation | Generate Homography coordinate. | Pass |
| 2. | TC-ST-2 | Checking if previous inlier count is calculated. | System should calculate inline count | Checking calculation | Perform calculation in minimum time | pass |

## 5.3. Result Analysis

In the conducted project, the process was initiated by capturing images and arranging them from left to right, ensuring a minimum 40% overlap between adjacent pictures. The Scale-Invariant Feature Transform (SIFT) was then employed to extract features within each image, establishing correspondences between them. Subsequently, the Random Sampling and Consensus (RANSAC) algorithm was applied to eliminate outliers from the identified correspondence points. The remaining inliers were utilized to generate an initial estimate of the Homography, representing the geometric transformation between images. To refine this estimate and enhance accuracy, the Levenberg-Marquardt optimization technique was employed. This entire procedure was iteratively applied to each pair of adjacent images. Following the acquisition of Homographies for all picture pairs, they were consolidated with respect to the central image. Finally, all images were projected onto a blank canvas using inverse warping, incorporating bilinear interpolation to achieve a seamless panoramic result. The results obtained from unit testing indicated that the system effectively executed its intended functions and met its goals. However, there remains room for improvement in terms of expanding the system's capabilities.

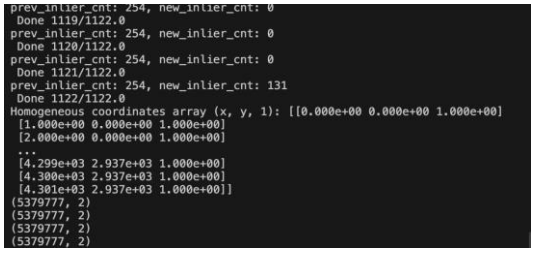



**Figure 5.1: Generating Inlier count and homography Coordinate.**

**Figure 5.2: Points calculation Though inlier**

The Random Sampling and Consensus (RANSAC) algorithm is applied to eliminate outliers from the identified correspondence points. The remaining inliers are used to generate an initial estimate of the Homography, representing the geometric transformation between images.

**Figure 5.3: Shift Alignment**

The Inlier function takes the estimated Homography H, the expected correspondence points, and a threshold. It calculates the distance between the expected transformed points and the actual corresponding points and identifies inliers based on whether the distance is below the

threshold. It essentially shifts points from one image coordinate system to another, ensuring alignment.

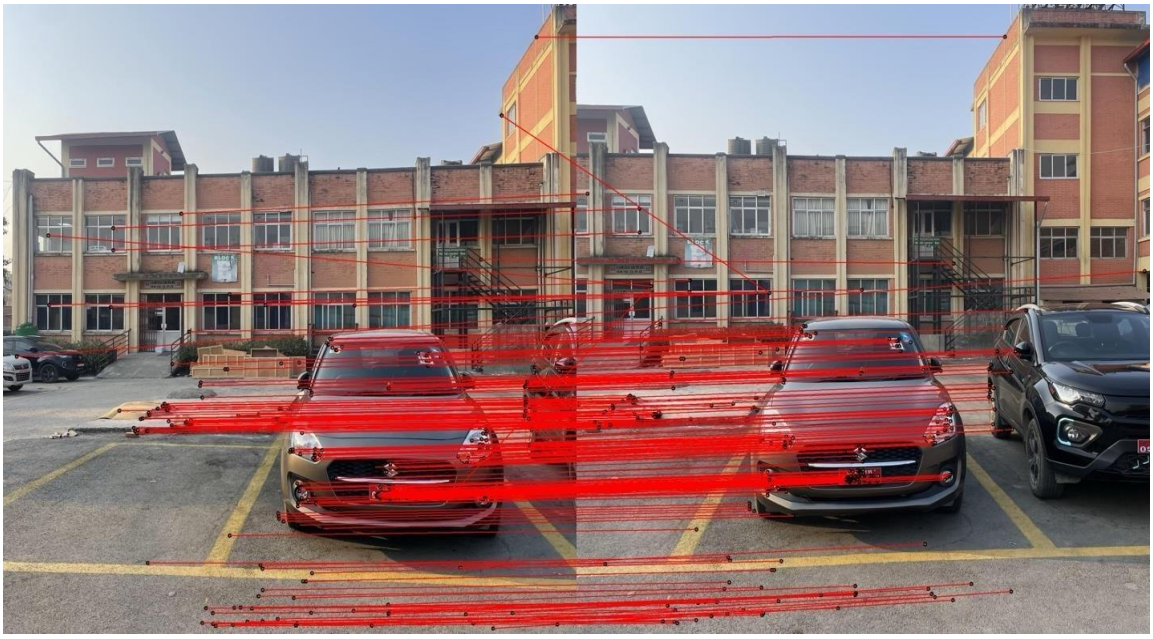


**Figure 5.4: Identifying outliers based on the distance threshold.**

The Inlier function also identifies outliers based on the distance threshold delta. Points with distances above this threshold are considered outliers.

The Draw lines function in code visualizes the correspondences by drawing lines between inliers and circles around corresponding points with different colors.

# CHAPTER 6 CONCLUSION AND RECOMENDATION

## 6.1 Conclusion

In conclusion, Blendify app provides a user-friendly and accessible solution for combining multiple images of the same category into a single, blended output. Using practical algorithms like SIFT, RANSAC, and a straightforward blending approach, this app offers a reliable tool for users seeking a quick and easy way to create panoramic images. The SIFT algorithm aids in identifying key features across images, allowing for basic alignment and minimizing distortions. RANSAC contributes to the overall robustness of the stitching process by refining the alignment model, improving the accuracy of the final composition. The simple blending algorithm ensures a smooth transition between adjacent images, offering users a visually cohesive panorama without the need for complex adjustments.

While this app may not boast the complexity of high-end solutions, its simplicity makes it accessible to a broader audience, catering to users who prioritize ease of use over advanced features. Whether it's for personal photography projects or casual image stitching needs, this app provides a practical and straightforward tool for creating blended images with minimal effort.

In summary, Blendify app may not be the most sophisticated, but it serves as an approachable and effective solution for users looking to combine multiple images seamlessly. By incorporating these basic algorithms, the app aim to provide a practical tool that meets the needs of users seeking a quick and straightforward way to create visually appealing panoramic images but will not work if object in images are moving.

## 6.2 Future Recommendation

There are several potential future recommendations that could be implemented in the Blendify project:

- **Performance Optimization:** Consider implementing optimizations to enhance the app's speed and efficiency. This could involve refining the algorithms for faster processing, ensuring a smoother user experience.

- **Creating panorama for moving images:** Currently app works if image and object in images are not moving i.e. for fix so app could involve for refining to create panorama for moving object in an image in near future.
- **Integration of Additional Algorithms:** Explore the possibility of integrating more algorithms or alternative techniques to provide users with a variety of stitching options. This could cater to different user preferences and types of image sets.
- **Automatic Image Correction:** Implement automatic image correction features to address common issues such as exposure variations, color inconsistencies, or perspective distortions. This would enhance the overall quality of the stitched output.
- **Collaboration and Sharing Features:** Introduce features that enable users to collaborate on stitching projects or share their creations on social media platforms directly from the app. This could enhance the app's social aspect and broaden its user base.
- **User Guidance and Tutorials:** Incorporate in-app guidance and tutorials to assist users in maximizing the potential of the app. This could include step-by- step instructions, tips for optimal image capture, and troubleshooting guidance.
- **Platform Expansion:** Explore the possibility of expanding the app to different platforms, such as mobile devices or web applications, to make it more accessible to a wider audience.

By focusing on these future recommendations, our image stitching app can evolve into a more versatile and feature-rich tool, meeting the diverse needs of users and further establishing itself as a reliable solution in the field of basic image stitching

# REFERENCES


[1] J. L. L.-y. G. Zhong Qu, "A method of image stitching with partition matching and direct detection for rotated image," December 2022. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0141938222001342?via%3Dihu b.

[2] K. Joshi, "A Survey on Real Time Image Stitching," ijers, Pune, 2020.

[3] Z. Zhang, "ieexplore," IEEE, 06 October 2020. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9213616.

[4] Y. Chang, "Drone-based panorama stitching:A study of SIFT, FLANN, and RANSAC techniques," 2023.

[5] S. M. A.-B. A. A. M. Mohammed Gheni Alwan, "Automatic panoramic medical image stitching improvement based on feature-based approach," 2022.

[6] B. V. R. L. a. M. E. L. S. Seibt, "Dense Feature Matching Based on Homographic," IEEE, 2022.

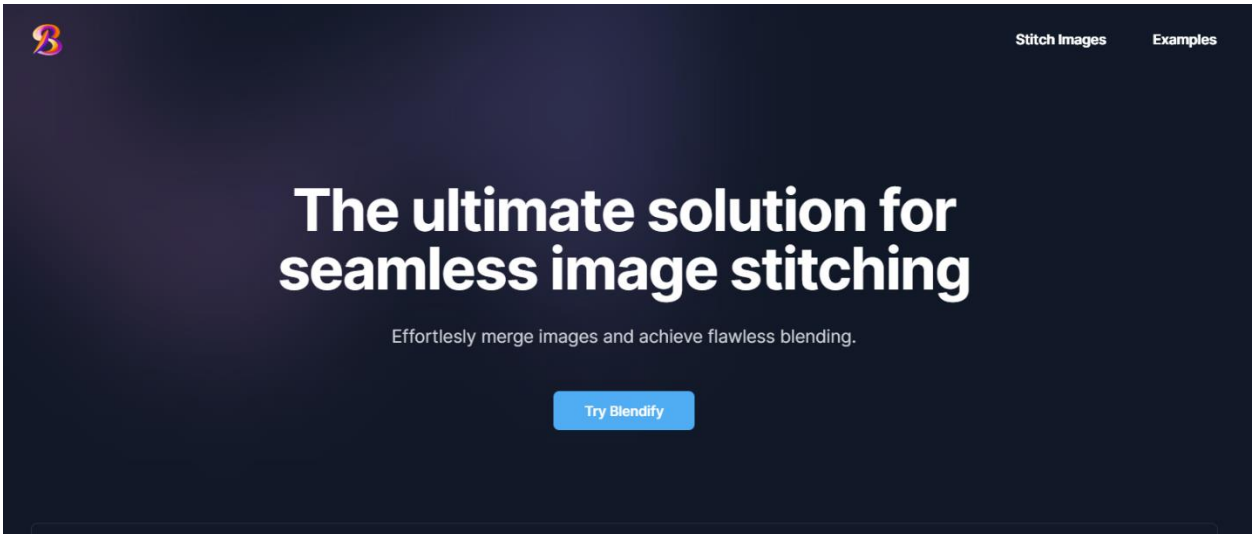
[7] J. K. J. P. B.-R. L. A. M.-H. E. R.-O. a. K. A. C.-G. Gómez-Reyes, "Image Mosaicing Applied on UAVs Survey," Mexico, 2022.

[8] M. ABBAS, S. SALEEM, F. SUBHAN and A. and BAIS, "Feature points-based image registration between satellite imagery and aerial images of agricultural land," Turkish Journal of Electrical Engineering and Computer Sciences, 2020.

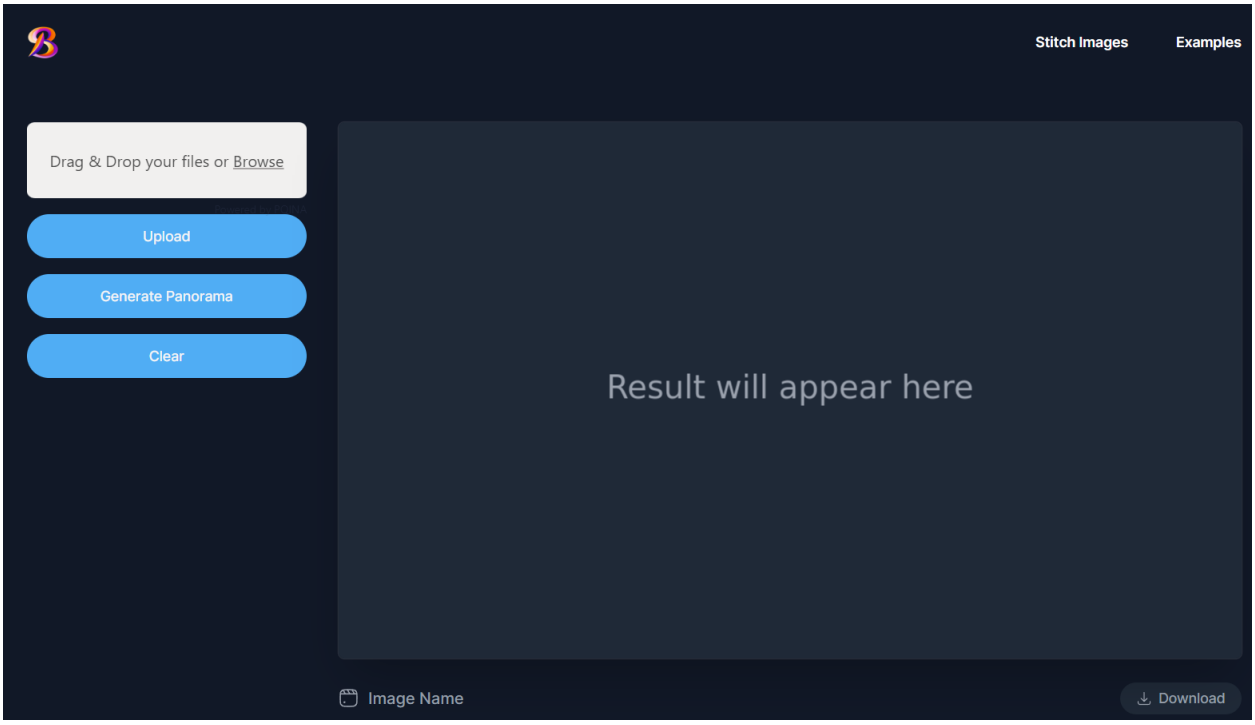
[9] J. L. X. W. X. N. Z Wang, "Underwater Terrain Image Stitching Based on Spatial Gradient Feature Block.," Computers, Materials & Continua, 2022.

# APPENDICES

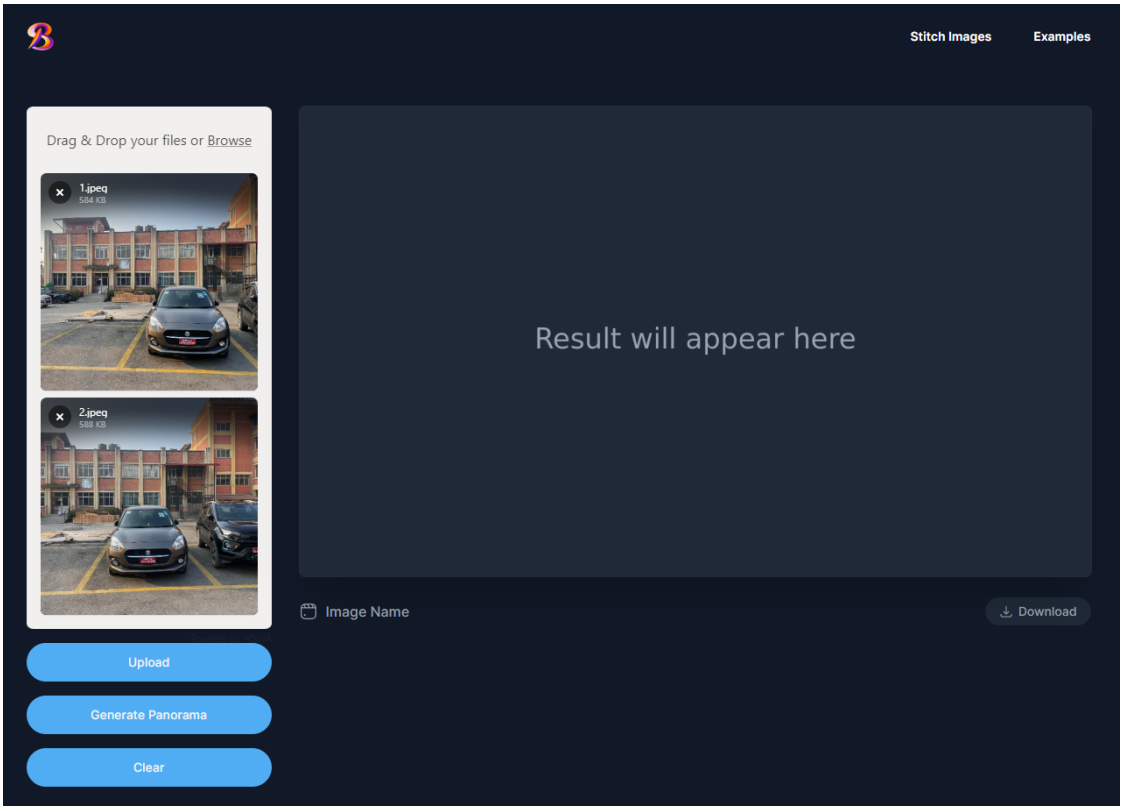
(Screenshots)



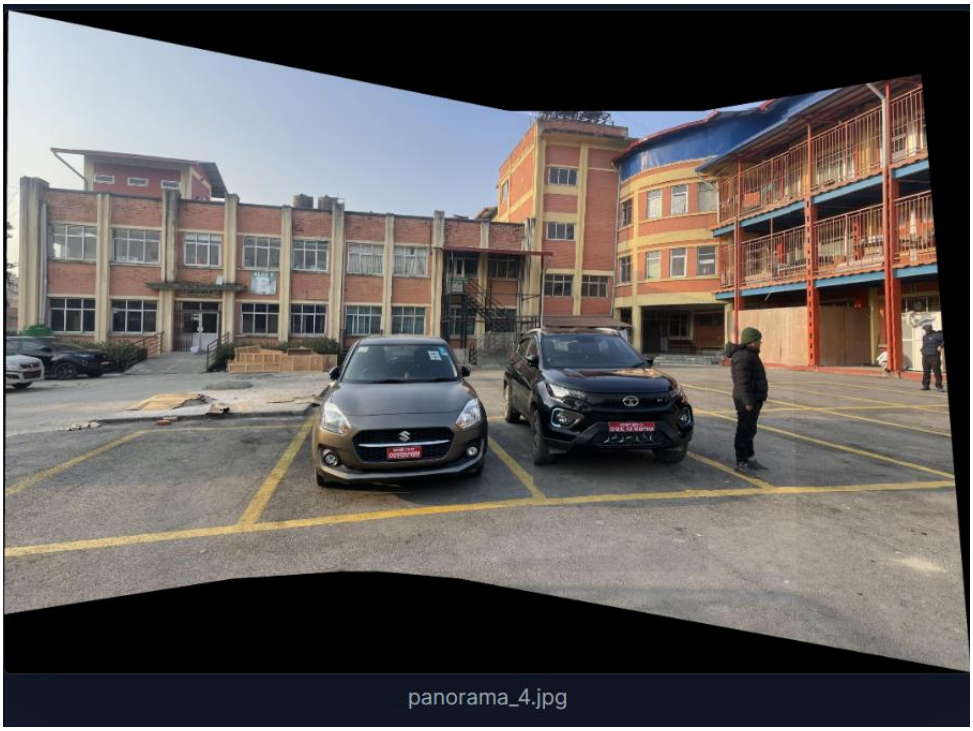

**Home page**




**User Interface**



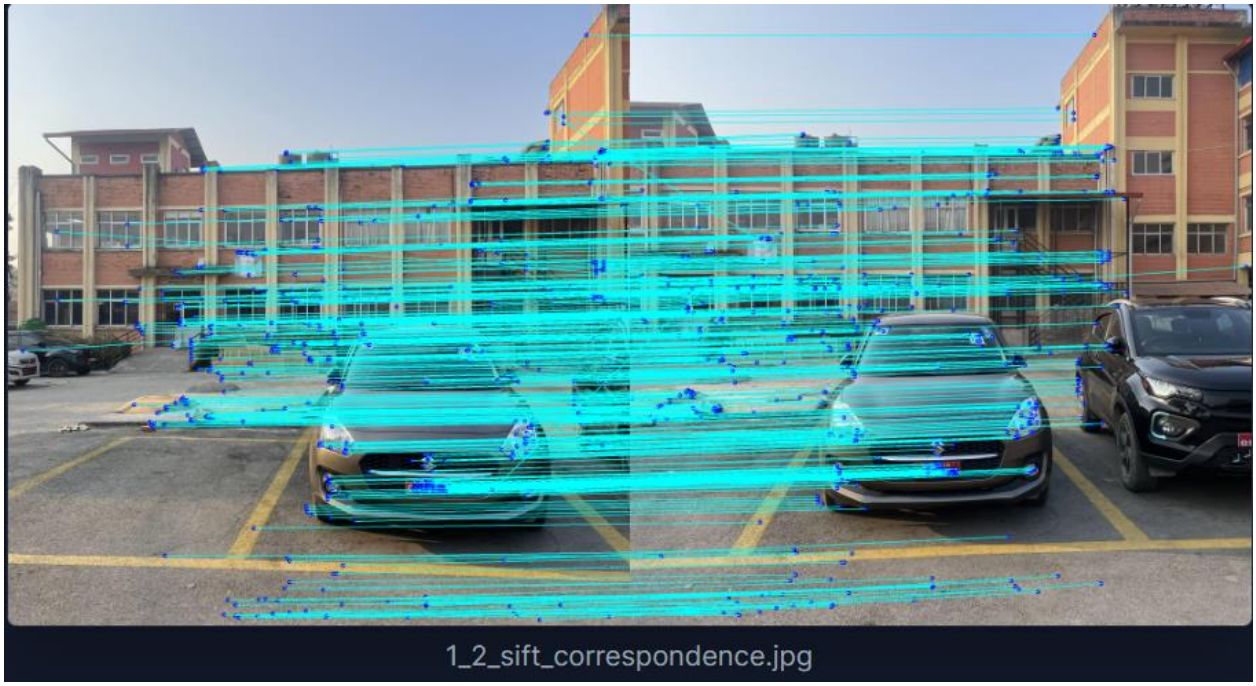

Uploading Images




Panorama Generated

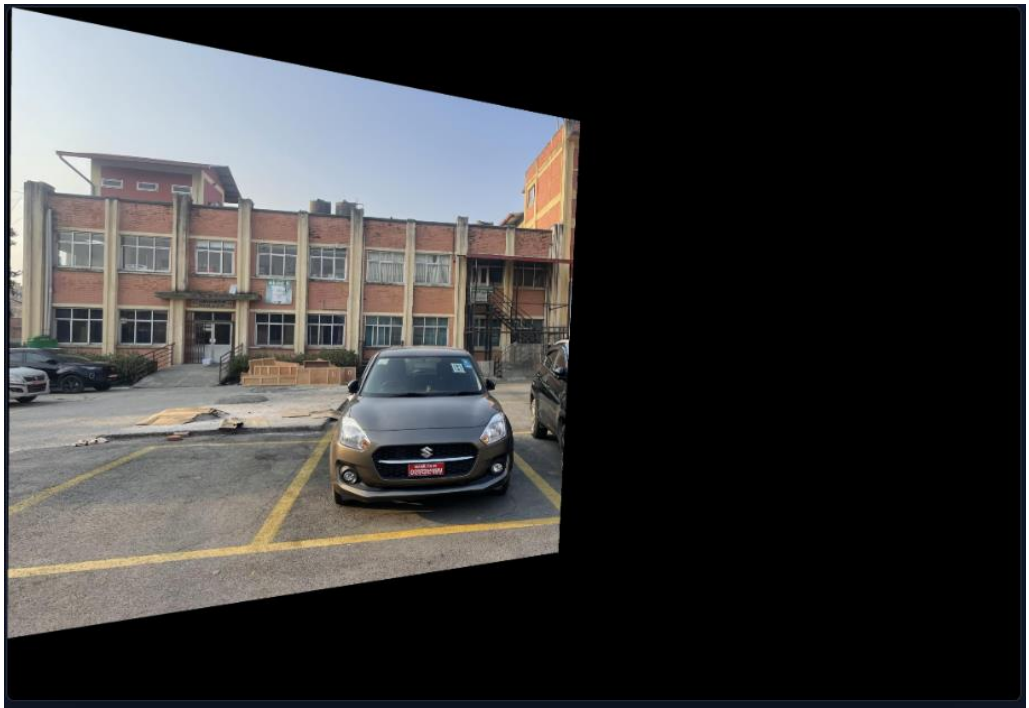

Inliers


Outliers

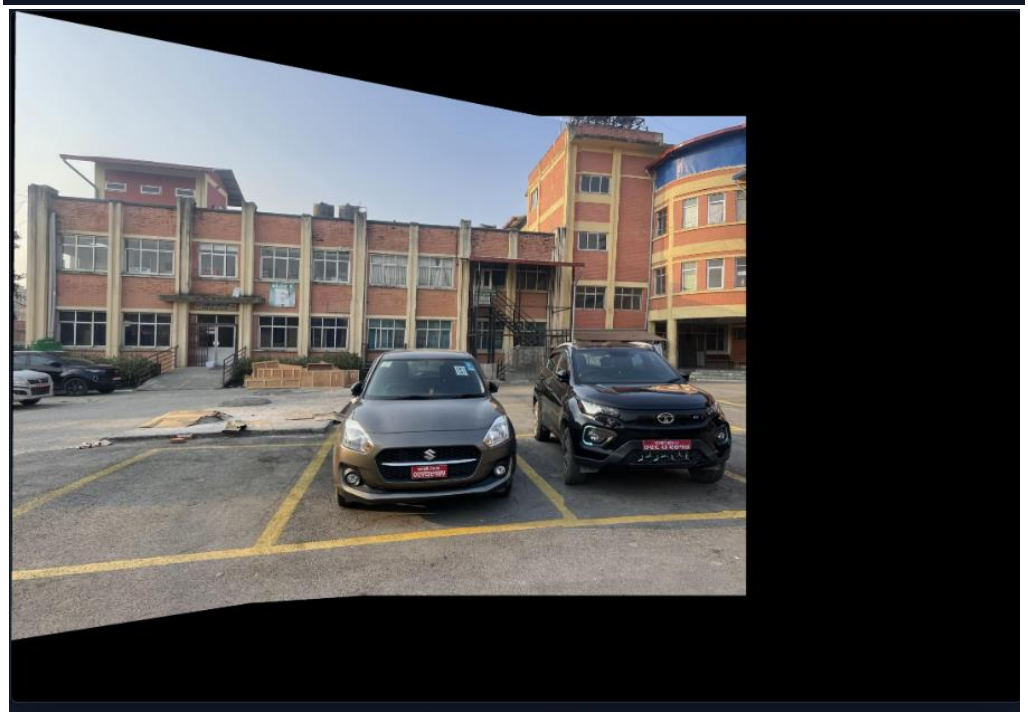


SIFT Correspondences

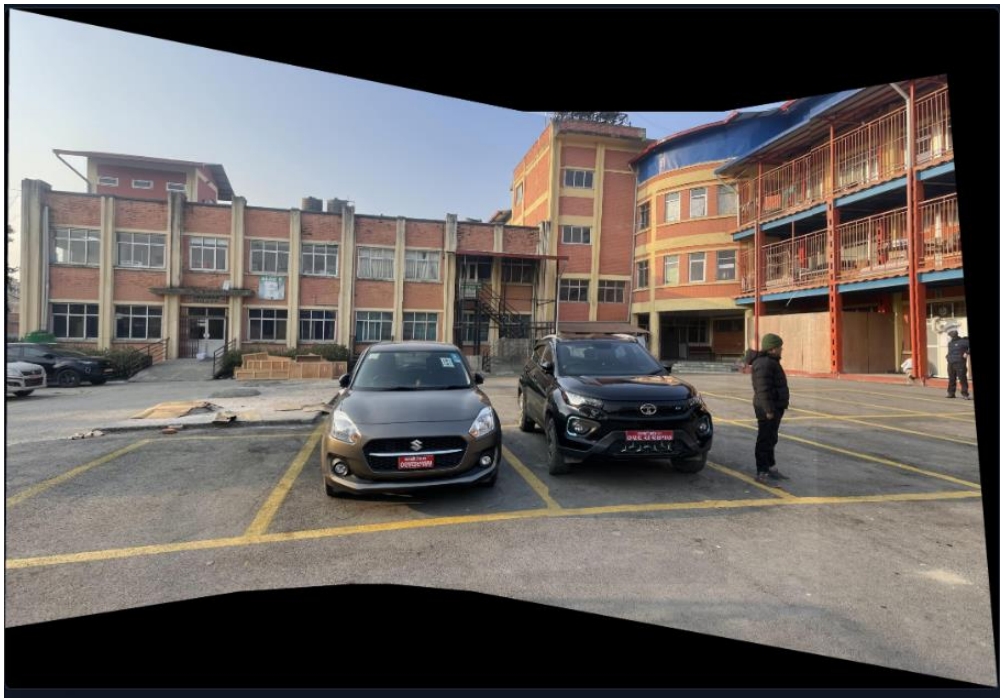
## Panorama Generation Process



panorama_0.jpg



panorama_2.jpg



panorama_4.jpg

final_panorama.jpg

Download