Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

3D Scene Reconstruction from DJI Drone photos

Bachelor thesis

2024
Lukáš Bujňák

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

3D Scene Reconstruction from DJI Drone photos

Bachelor thesis

Study Programme:	Computer Science
Field of Study:	Computer Science
Department:	Department of Computer Science
Supervisor:	doc. RNDr. Andrej Ferko, PhD.
Consultant:	RNDr. Martin Bujňák, PhD.

Bratislava, 2024
Lukáš Bujňák

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Lukáš Bujňák
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: 3D Scene Reconstruction from DJI Drone photos
3D rekonštrukcia scény zo záberov dronu DJI

Anotácia: Vytvárame 3D modely snímané bežným dronom, ktory adaptívne sníma alebo označí príslušné časti scény. Zber dát i rekonštrukciu riadi používateľ mobilnou aplikáciou.

Cieľ: Špecifikácia a vývoj mobilnej aplikácie, ktora sprístupní pohľad z kamery drona a umožní používateľovi vhodné nastavenia.
Dátovo prepojíme aplikáciu s Reality Capture Node (ďalej iba RCNode).
Otestujeme možnosti na zatiaľ iba manuálne fotenie objektu, ktorého fotky budú spracované RCNode, ktorý ich následne spracuje a poskytne nám 3D model. Takto získaný model bude možné zobraziť priamo v aplikácii.
Navrhneme experimenty a overíme adaptívnosť procesov snímania a rekonštrukcie.
Vyhodnotíme prototypové riešenie s ohľadom na očakávaný vývoj technológie.

Literatúra: Hartley, R. - Zisserman, A. 2004. Multiple View Geometry in Computer Vision. Cambridge University Press.
SZ DJI Technology Co. Ltd. [online] https://developer.dji.com. 2024.
Najnovšie články z vedeckých konferencií v oblasti Computer Vision.

Kľúčové slová: 3D reconstruction, photogrammetry, remote sensing

Vedúci: doc. RNDr. Andrej Ferko, PhD.
Konzultant: RNDr. Martin Bujňák, PhD.
Katedra: FMFI.KAG - Katedra algebry a geometrie
Vedúci katedry: doc. RNDr. Pavel Chalmovianský, PhD.

Dátum zadania: 11.10.2023

Dátum schválenia: 30.10.2023 doc. RNDr. Dana Pardubská, CSc.
garant študijného programu

.....
študent

.....
vedúci práce

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Lukáš Bujňák
Study programme:	Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Bachelor´s thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: 3D Scene Reconstruction from DJI Drone photos

Annotation: We create 3D models captured by the out-of-a-shelve drone, which adaptively measures or signalizes the necessary parts of the scene. Data collection and reconstruction are controlled by the user with a mobile application.

Aim: We intend to create a mobile application that will project the view from the drone camera and basic UI settings.
Data from the mobile application with be shared with the Reality Capture Node (hereinafter referred to as RCNode).
We intend to combine manual photography of an object, whose photos will be processed by RCNode, which then exports a 3d model that can be displayed in application.
Enable transferring between the application and the running RCNode over the local network and the mobile network.
We will evaluate the prototype solution with regard to the expected development of the technology.

Literature: Hartley, R. - Zisserman, A. 2004. Multiple View Geometry in Computer Vision. Cambridge University Press.
SZ DJI Technology Co. Ltd. [online] https://developer.dji.com. 2024.
Recent papers from scientific conferences in the field of Computer Vision.

Keywords: 3D reconstruction, photogrammetry, remote sensing

Supervisor:	doc. RNDr. Andrej Ferko, PhD.
Consultant:	RNDr. Martin Bujňák, PhD.
Department:	FMFI.KAG - Department of Algebra and Geometry
Head of department:	doc. RNDr. Pavel Chalmovianský, PhD.
Assigned:	11.10.2023

Approved: 30.10.2023 doc. RNDr. Dana Pardubská, CSc.
Guarantor of Study Programme

.....
Student

.....
Supervisor

Acknowledgments: I would like to express my sincere gratitude to my supervisor, doc. RNDr. Andrej Ferko, PhD., for his invaluable guidance, support, and advice throughout this bachelor thesis. I also thank RNDr. Martin Bujňák, PhD., for his expert consultations and assistance with technical issues. I am grateful to all the members of the Department of Computer Science at the Faculty of Mathematics, Physics, and Informatics, Comenius University in Bratislava, for their contributions to my education and for providing the necessary resources for this work. Finally, I would like to thank my family and friends for their patience, support, and encouragement during my studies.

Abstrakt

Táto bakalárska práca popisuje vývoj iOS aplikácie CR Fly, ktorá je navrhnutá na zlepšenie procesu 3D rekonštrukcie scén pomocou snímok zachytených dronmi DJI. Aplikácia využíva softvérovú vývojovú sadu (SDK) DJI, ktorá umožňuje obojsmernú komunikáciu s dronom, a softvér RealityCapture, umožňujúci rekonštrukciu reálnych objektov do ich digitálnych dvojčiat prostredníctvom dronových snímok. Vývoj aplikácie bol realizovaný s dôrazom na modularitu a škálovateľnosť pomocou architektúry Model-View-Controller v programovacom jazyku Swift. Aplikácia taktiež poskytuje užívateľovi pohľad z kamery dronu v reálnom čase, ako aj analýzu a 3D rekonštrukciu snímaného prostredia, čo výrazne napomáha pilotovi pri snímaní jednotlivých objektov. Rozsiahle testovanie potvrdilo funkčnosť a schopnosť aplikácie rekonštruovať detailné 3D modely rôznych scén. Pre experimentálnu rekonštrukciu boli vybrané Pomník Mikuláša Koperníka a Kostol sv. Alžbety Uhorskej, ktoré reprezentujú jednoduchú a komplexnú scénu a slúžia na overenie praktickej funkčnosti aplikácie. Na riešenie ďalších limitácií existujúcich riešení, konkrétne autonómnych letov, bola aplikácia navrhnutá tak, aby sa v budúcnosti dala rozšíriť o modul autonómneho snímania. Táto rozšíriteľnosť je umožnená tým, že mobilné zariadenie už disponuje fotografiami, polohami kamier, 3D rekonštrukciou scény, a navyše, DJI SDK umožňuje nastaviť kontrolné body pre autonómny let dronu.

Kľúčové slová: 3D rekonštrukcia, fotogrametria, diaľkové snímanie

Abstract

This bachelor's thesis describes the development of the iOS application CR Fly, which is designed to enhance the process of 3D scene reconstruction using images captured by DJI drones. The application utilizes the DJI Software Development Kit (SDK), which enables bidirectional communication with the drone and RealityCapture software, allowing the reconstruction of real-world objects into their digital twins using images obtained from drones. The development of the application was carried out with an emphasis on modularity and scalability using the Model-View-Controller architecture in the Swift programming language. The application also provides the user with a real-time view from the drone's camera, as well as analysis and 3D reconstruction of the captured environment, significantly aiding the pilot in capturing individual objects. Extensive testing confirmed the functionality of the application and its ability to reconstruct detailed 3D models of various scenes. For experimental reconstruction, the Monument of Nicolaus Copernicus and the Church of St. Elizabeth of Hungary were selected, representing simple and complex scenes, which serve to verify the practical functionality of the application. To address further limitations of existing solutions, specifically autonomous flights, the application was designed to be extendable in the future with a module for autonomous scanning. This extensibility is enabled by the mobile device already possesing photographs, camera positions, 3D reconstructions of the scene, and additionally, the DJI SDK allows setting waypoints for the drone's autonomous flight.

Keywords: 3D reconstruction, photogrammetry, remote sensing

Contents

List of Figures

List of listings

List of acronyms

2D	two-dimensional
3D	three-dimensional
API	Application Programming Interface
GIS	Geographic Information System
GPS	Global Positioning System
GSD	Ground Sample Distance
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
LOD	Level of Detail
MVC	Model-View-Controller
RAM	Random Access Memory
SDK	Software Development Kit
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
UAV	Unmanned Aerial Vehicle
UI	User Interface
UML	Unified Modeling Language
VFX	visual effects
VPN	Virtual Private Network

Introduction

In the field of three-dimensional (3D) scene reconstruction, we are witnessing significant technological progress, which allows us to obtain accurate and detailed 3D models of real scenes. Models of these scenes are used in a wide range of applications across various industries, including creating visual effects (VFX), games, virtual production, geodetic measurements, and virtual environment visualization. One innovative method for attaining this objective involves the reconstruction of 3D models from image data acquired via Unmanned Aerial Vehicles (UAVs), particularly those that are commercially available for personal use.

The utilization of UAVs facilitates the generation of precise and detailed 3D models, which requires acquiring a sufficient number of images from various angles, often from locations inaccessible to humans. These images are captured from aerial perspectives, and such an approach is not feasible with traditional ground-based methods. They offer unparalleled views and data, allowing for detailed analysis and modeling that surpass what is achievable through conventional techniques.

Various UAV manufacturers provide proprietary applications for controlling UAVs and capturing images, featuring a range of camera settings options. However, a significant limitation of these applications is the need for manual image transfer into software designed for object reconstruction. Partial automation introduces significant efficiencies to the process, including actions like transferring images directly during flight or initiating reconstruction calculations automatically. The process of reconstruction can sometimes take several hours and by implementing these automation features, we could drastically reduce the time required, enhancing productivity and operational efficiency.

In our bachelor's thesis, we focus on utilizing the Application Programming Interface (API) from DJI [12], which grants us access to their Software Development Kit (SDK) for UAV control. This approach enables us to develop a specialized application that extends beyond the traditional functions of UAV operation. The application will be integrated not only with the UAV's control elements, but also with the processing capabilities of the RealityCapture software [4]. The synergy with RealityCapture, a software capable of effectively reconstructing 3D scenes from images, provides us with a unique opportunity to automate and optimize the 3D modeling process from UAV-captured images, thereby expanding the efficiency and application of this technology.

In the initial part of the bachelor's thesis, we will familiarize ourselves with the basic concepts and the motivation behind the creation of our application. Subsequently, we will examine existing UAV-based reconstruction solutions, based on which we will aim to specify our application in such a way as to overcome their limitations. Later, we will develop a functional prototype that communicates with DJI's UAVs and the RealityCapture software. The prototype will offer the capability for direct uploading of images from the UAV to RealityCapture, where they will be processed into a 3D model. This work lays the groundwork for future developments in fully automating the process, including autonomous flight. Finally, with the operational application in hand, we will conduct several experiments to demonstrate the variations in reconstruction quality under different settings and with varying densities of images.

Chapter 1

Background review

At the beginning of this chapter, we introduce the process of reconstructing objects into their digital twins, discuss the various methodologies employed, and describe the methodology most suitable for use with UAVs. Subsequently, we will explore compelling reasons for utilizing UAVs in reconstruction, and examine existing UAV-based solutions along with their limitations. The terminology used throughout this chapter is based on source [7]. Mathematically, we work in a 3D metric space (E^3, d), using the standard Euclidean metric. All measurements are specified in standard units, such as meters for length.

1.1 3D reconstruction

The process of 3D reconstruction is dedicated to accurately documenting the shape and appearance of real objects. The primary objective is to create a digital 3D model that closely emulates the original object, capturing both its geometry and surface details with high fidelity. Various methodologies are used to achieve this, broadly categorized into two groups:

- **Active scanning:** This approach involves projecting structured light, infrared light, or a laser beam onto the object to collect depth information. The interaction, typically through reflection, between the emitted light and the object's surfaces enables the detailed measurement of the object's geometry.

- **Passive scanning:** Conversely, passive scanning employs techniques such as stereoscopic vision, which requires the compilation of multiple images from different perspectives. These images are then analyzed to infer the object's depth, creating a depth map by comparing the variances and similarities among the images, without the necessity for direct light projection onto the object.

These methodologies play a crucial role in producing detailed and accurate 3D models that faithfully replicate the geometric and textural characteristics of the original objects. The accuracy provided by these methods enables the broad application of the resulting models in various domains, such as the development of virtual museums, the design of game maps and game assets, the creation of character models in video games, and geodetic measurements [13].

Photogrammetry, a leading passive 3D reconstruction method, leverages two-dimensional (2D) photographs to generate accurate 3D representations of scenes and objects within extended Euclidean 3D space or projective space, depending on the specific application requirements. This technique offers distinct advantages and has become an essential tool in the advancement of digital modeling practices.

1.2 Photogrammetry

Photogrammetry is an engineering discipline dedicated to the acquisition of reliable data, typically from a set of images, on the characteristics of surfaces and objects, without the need for direct physical contact with these entities. The results of photogrammetric processes typically manifest as maps, drawings, measurements, or 3D models, faithfully representing real-world objects or scenes [10].

At the core of photogrammetry is the use of photographic techniques to accurately determine the dimensions and locations of subjects. It is important to note that the quality of the photographs directly impacts the model's precision and intricacy, which can sometimes be more critical than the quantity of photographs.

While emphasis on photograph resolution is crucial for achieving precise and intricate 3D models, not all applications of photogrammetry demand such detailed focus. In fields such as cartography, the priority often shifts toward the quantity of objects being mapped as well as their coverage. This aspect underscores photogrammetry's broad applicability, notably in the creation of modern maps where aerial photography and photogrammetric techniques are combined to cover extensive geographic areas.

A noteworthy aspect of photogrammetry lies in its critical role in the preservation of natural and cultural heritage. By facilitating the detailed 3D modeling of historical monuments and archaeological sites, photogrammetry serves as a powerful tool for the documentation and digital preservation of these invaluable assets, ensuring their accessibility for future generations. This method not only supports research and education but also enhances restoration possibilities by offering a non-intrusive means to accurately examine and replicate the intricate details of our cultural heritage. Consequently, the application of photogrammetry in cultural heritage preservation highlights its invaluable capacity to forge a connection between the past and the future.

1.3 Aerial photography

In this work, we focus on scanning from drones, which can be considered as a form of closer-range airborne photogrammetry. For the sake of clarity, however, we will describe the principles of airborne photogrammetry.

Aerial photography is captured from an elevated or airborne perspective, unlike traditional photographs taken from ground level. To cover large areas efficiently, images need to be captured within a short timeframe. This often involves an aircraft flying at a predetermined altitude, taking a sequence of images.

The principal limitation of this imaging technique is intrinsically linked to the aircraft's altitude. As the distance between the camera's sensor (its focal point) and the photographed object increases, a single pixel on the image covers a larger surface area on the ground, which reduces the accuracy of the reconstruction.

The Ground Sample Distance (GSD) is a metric utilized to calculate the extent of the ground footprint covered by a single pixel in an image. GSD is commonly expressed as centimeters per pixel and is calculated using the following equations:

$$\begin{aligned} GSD &= max(GSD_H, GSD_W), \\ GSD_H = \frac{H \cdot S_H}{f \cdot I_H}, &\quad GSD_W = \frac{H \cdot S_W}{f \cdot I_W}, \end{aligned} \tag{1.1}$$

where GSD_H and GSD_W determine the size of the GSD, expressed in values of height and width. The variable H represents the aircraft's altitude above ground level in meters. The focal length of the camera's lens is denoted by f and measured in millimeters. The sensor dimensions, S_H for height and S_W for width, are also measured in millimeters, while I_H and I_W indicate the image's height and width in pixels. The meaning of these parameters is visually presented in Figure 1.1 [11].

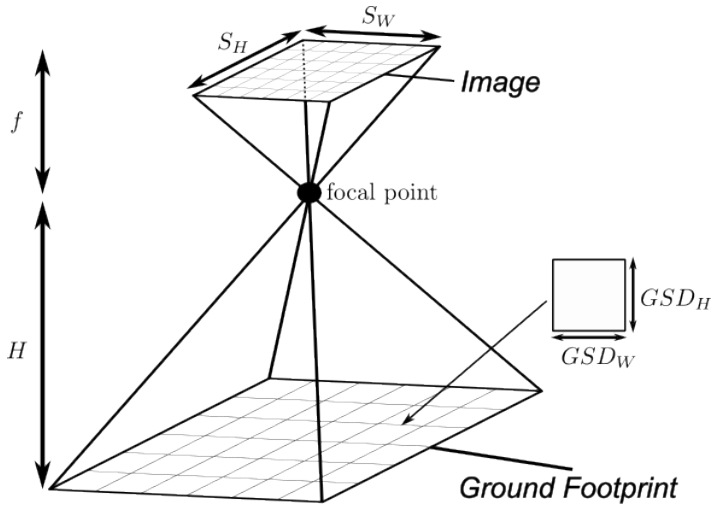

Figure 1.1: Visual representation of GSD parameters

From Equation 1.1, it is easy to recognize that an increase in the aircraft's altitude is inversely related to image detail. This is because as the aircraft's altitude increases, a single pixel will represent a larger surface area on the ground, potentially diminishing the fidelity of the reconstructed object. This interplay between the extent of ground coverage and image resolution is a pivotal consideration in airborne photogrammetry, as it significantly impacts the accuracy and detail of the derived maps. Let's demonstrate it on example.

1.3.1 Example of GSD calculation

Consider the use of a full-frame sensor utilized for airborne photogrammetry, where the sensor dimensions are given as $S_W = 36$ mm and $S_H = 24$ mm. With an image aspect ratio of 3:2 and a resolution of 50 megapixels, it follows that $I_W = 8660$ pixels and $I_H = 5774$ pixels. In airborne photogrammetry, the focal length f is minimized to reduce GSD, therefore, a typical focal length for a full-frame sensor is $f = 24$ mm. Let the flight altitude of the aircraft equipped with the described sensor be $H = 500$ m. Since $\frac{S_W}{I_W} > \frac{S_H}{I_H}$, it follows that $GSD = GSD_W$, and GSD_W is calculated as follows:

$$GSD_W = \frac{H \cdot S_W}{f \cdot I_W} = \frac{36 \cdot 10^{-3} \cdot 500}{8660 \cdot 24 \cdot 10^{-3}} \approx 0,0866 \text{ m/px} \Rightarrow GSD \approx 8,6 \text{ cm/px},$$

therefore, a single pixel captures an area of approximately $73,96$ cm^2. For the purpose of creating navigation maps, this resolution would be sufficient. However, for creating a 3D visualization of a city, such resolution would lead to an undesirable loss of detail.

1.4 The role of UAVs in 3D reconstruction

The calculation of GSD demonstrates how using UAVs for aerial photography dramatically improves image quality due to their significantly lower flight altitudes. Consequently, this enhanced image quality directly translates to a significant improvement in the quality of the reconstructed model. This is due to the capability of UAVs to operate at altitudes up to 100 times lower than traditional aircraft, allowing them to capture images from much closer distances to the object.

Furthermore, the use of UAVs in airborne photogrammetry opens up possibilities for accessing places that are inaccessible to traditional aircrafts equipped with imaging sensors, such as various castles located in forests.

Moreover, many modern UAVs are equipped with obstacle avoidance sensors. If fully autonomous flights become feasible, these sensors could maintain a controlled distance from objects, helping to automate remote sensing processes. This advancement could lead to faster, more precise, and autonomous scanning of objects and terrain.

1.5 Review of related works

Currently, there exists a considerable number of solutions for object reconstruction from aerial imagery, including those that facilitate the acquisition of these images directly from UAVs. However, some software programs encounter issues when attempting to automate the reconstruction process, while others are either limited by specific UAV manufacturers or designed for particular purposes, such as geodetic measurements only. Among the more effective solutions for scene reconstruction are Pix4Dcapture [9] and DroneDeploy [3]. These will be discussed in detail, with an analysis of their weaknesses aimed at suggesting potential improvements.

1.5.1 Pix4Dcapture

Pix4Dcapture is an application designed for mobile phones and tablets, developed and maintained by the Swiss company Pix4D S.A., which is recognized as a market leader in photogrammetry software technology. In addition to Pix4Dcapture, they provide reconstruction solutions for images captured from airplanes, mobile phones, and other camera-equipped devices. The application provides users with two capture modes: free flight mode and autonomous flight mode.

In the free flight mode, the user can manually control the UAV while the application ensures that its camera captures images. These images are acquired by altering the UAV's position, either every 5 meters horizontally or every 3 meters vertically. Subsequently, the application illustrates on a map the locations where each image was obtained.

In the autonomous flight mode, the flight itself is preceded by planning using a 2D planner. This 2D planner is utilized to create a flight plan for the UAV in the desired reconstruction area. It offers three flight plan modes, which can be customized to fit the specific requirements of the area. The most commonly used modes are represented visually in the planner interface by a grey background with a white path indicating the planned route (see Figure 1.2). These modes are listed as follows, from left to right:

- **Grid mode** – Creates an area where the UAV moves horizontally at a consistent altitude, covering the entire area in a grid pattern.
- **Orbit mode** – Establishes an area that the UAV circles at a consistent altitude, making one complete orbit around this area.
- **Cylinder mode** – Generates an area that the UAV orbits, but at multiple different altitudes.

Depending on the selected mode, users will gain access to various settings including flight altitude, camera angles relative to the horizon, and a percentage representation

of the overlap between individual image pairs. The execution of the flight itself can then be described according to these phases:

- *Planning phase.* To execute the flight, we must create a flight plan using a 2D planner, which will guide the UAV in flying and taking images over the designated area.
- *Flight and capture phase.* This phase involves data collection (in our case, photographing) from the UAV using an automated flight algorithm.
- *Processing and reconstruction phase.* After collecting a sufficient number of images, they are uploaded to the cloud for processing and reconstruction.

These non-overlapping phases are carried out sequentially, with the initiation of a new phase following the completion of the preceding one. Thus, it is evident that for the reconstruction phase to begin, the capture phase must be concluded.

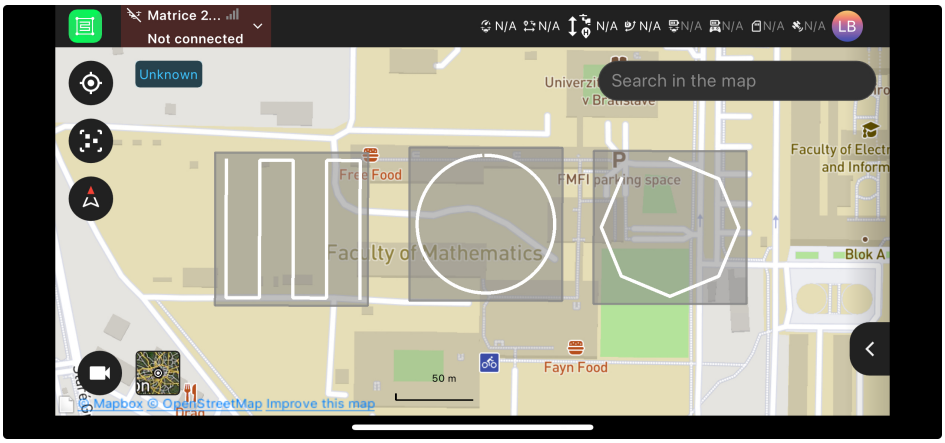

Figure 1.2: Visualization of individual modes in the Pix4Dcapture application

Since Pix4Dcapture is primarily designed for planning and controlling flight execution, we can view the reconstructed model, perform various measurements on it, and analyze it on the cloud after completing the processing and reconstruction phase. This cloud service, known as Pix4Dcloud, is described in detail on the Pix4D web portal [9].

1.5.2 DroneDeploy

DroneDeploy, similar to Pix4D, is an application designed for planning and controlling the execution of UAV flights. It is developed by DroneDeploy Inc., based in California. Their solution is relatively more user-friendly compared to Pix4D, as the entire project management is conducted on the cloud, accessible via their web portal. Conversely, flight planning takes place in the application, which is available for mobile devices and tablets. At the first glance at the UAV's camera view, it appears that DroneDeploy

places greater emphasis on the application's functionality for manually initiating image captures, where the images are generated through direct user control.

The various phases of flight execution in DroneDeploy occur similarly to those in Pix4Dcapture. A significant difference arises in flight planning, which offers only a grid mode, also providing settings for flight altitude and overlap of individual photographs. After completing the processing and reconstruction phase, we can proceed to the cloud and interact with the reconstructed model [3].

1.5.3 Software comparison and shortcomings

Pix4Dcapture provides more planning options, whereas DroneDeploy offers improved UAV control during flight. Notably, there are several common issues associated with these commercial products that will be discussed further.

Challenges of autonomous flight. Autonomous flight requires users to specify flight altitudes, which affects the utility of the capture. A prime example involves flying over terrain with significant elevation changes, where it is essential to determine the highest point to ensure complete aerial coverage. This requirement presents a challenge as increasing the distance from the target leads to a reduced pixel size in the final image. Consequently, there is a loss of detail and the creation of unusable data, particularly in elements such as the sky or the surrounding landscape.

Field application feedback deficiency. In some cases, such flights may require permissions from relevant authorities, which may not be readily available. Since we can only estimate, not know in advance, whether the captured images will be sufficient for reconstruction, there is a possibility that creating additional images could be delayed, thereby extending the time needed to reconstruct an adequately precise model. Additionally, while awaiting further permissions, vegetation, such as grass, may grow, significantly deteriorating the quality of the reconstruction outcome. Consequently, each restarted reconstruction should begin with an empty scene, meaning the process starts anew.

The inefficacy of cloud-based reconstruction. When utilizing the cloud, a stable internet connection is generally required, which may not always be available or may have limited speed in remote areas, such as woods.

1.6 Our approach

To address these challenges, our work aims to develop software that overcomes existing limitations and effectively implements essential components for UAV-based photogrammetry. Specifically, this software will facilitate the automatic transfer of images from the UAV to the reconstruction software.

Furthermore, the pilot can optimize flight planning by utilizing data obtained from a preliminary survey flight, as well as data generated during the reconstruction process in the reconstruction software, allowing for the creation of an efficient flight path based on this comprehensive dataset. Thanks to this approach, we can plan the flight route in a 3D space, which enables us to reduce unusable data and streamline the remote sensing process. Moreover, the software will not only plan and execute these flights but also allow users to stop the flight if external conditions require it.

In response to the inefficacy of cloud-based reconstruction, our approach eliminates the dependency on a global internet connection. Instead of relying on cloud services, the proposed system utilizes a local setup in which a laptop or a mobile device serves as an access point for Wi-Fi. The reconstruction process will be performed directly on this laptop, ensuring that the system remains fully operational even in remote areas with no or limited internet access. This approach not only enhances accessibility in various environments but also improves the reliability and speed of the process.

Finally, the software will provide users with real-time reconstruction results throughout the entire process, both during autonomous and manual operations, enabling them to make informed decisions during the flight.

Chapter 2

Specification

As highlighted in the previous chapter, existing software solutions encounter various shortcomings that limit their effectiveness in autonomous flights and field applications. These issues include difficulties with flight altitude specification, inefficiency of cloud-based reconstruction, and lack of real-time feedback.

The CR Fly mobile application aims to address these limitations by enhancing coordination and efficiency in autonomous flight operations. It facilitates automatic image transfer, eliminates dependency on cloud services, and provides real-time reconstruction feedback. To understand the application's capabilities and design principles, we will explore the product overview, objectives, features, operating environment, security measures, and interfaces in detail.

2.1 Product overview

The CR Fly mobile application will be developed using Swift [2], widely utilized for creating iOS applications. Swift is renowned for its performance and security, making it ideal for developing responsive and intuitive mobile applications. The choice of Swift and iOS was motivated by their advanced support for integrating graphic functionalities for interface design and imaging functionalities for data processing, essential for efficiently handling UAV data and visualizing reconstructed scenes.

The application will be optimized to provide high performance and reliability when used in field conditions, ensuring compatibility with the latest versions of iOS. This measure will guarantee broad availability and high efficiency of the application for users of Apple devices. Additional emphasis will be placed on developing an intuitive User Interface (UI), which will simplify the control of UAVs and the processing of image data, making the application accessible to a wide range of users, from amateurs to professionals in the field of scene reconstruction.

2.1.1 Product objectives

The primary objective of the CR Fly mobile application is to establish an effective communication bridge between UAV hardware and scene reconstruction software to provide real-time feedback. It will be designed to support interchangeable components, thereby enabling compatibility with various types of hardware and software, which enhances its universality and adaptability across different application environments.

The application will facilitate the management of photographs and videos stored on the UAV as well as on local devices. Furthermore, it will provide the capability to add photographs taken on mobile devices, enhancing coverage of areas that are inaccessible by the UAV itself. These photographs can then be uploaded to the reconstruction software project for use in the 3D reconstruction process.

Additionally, the application will supports project management within the reconstruction software, enabling not only the visualization of reconstructed scenes but also the management of multiple reconstruction projects. It will allow users to initiate the reconstruction process and adjust export settings, thereby optimizing the quality and detail of the 3D models.

Lastly, the application will enable users to control UAVs and execute operations such as photography and video recording. It will provide camera settings that allow customization based on current conditions, enabling users to adapt the camera for optimal performance under varying environmental factors.

2.1.2 Product features

The fundamental function of the application will be to enable user communication with the UAV, allowing the user to conduct flights, adjust camera settings, and manage the UAV's storage. Additionally, the application will facilitate direct communication with the reconstruction software, operating independently of UAV operations. It will also ensure that UAV control remains unaffected by its connectivity with the reconstruction software. This dual independence permits the separate or combined use of both components, preserving their full operational integrity without interference.

One of the key features of the application will be the ability to process photographs in real-time during flight. This will allow users to receive immediate feedback on the quality and coverage of the objects being reconstructed. Users will be able to identify areas of these objects that are insufficiently covered by photographs and can make immediate adjustments to flight procedures to ensure better quality reconstruction.

This feature will not only simplify user analysis but also facilitate the implementation of fully autonomous flights, requiring minimal user intervention. By allowing the analysis of the reconstructed scene during flight, it enables real-time planning and adjustment of the flight path. This dual capability ensures that users can efficiently

manage flight operations and optimize data collection simultaneously.

When implementing autonomous flight capabilities, this functionality allows us to obtain a descriptive reconstruction of the scene through a simple survey flight. Using appropriate analysis, this reconstructed scene can then be utilized for flight path planning.

2.1.3 Operating environment

The CR Fly application will be developed using the Model-View-Controller (MVC) architecture, a widely recognized standard for creating structured and modular applications. This design pattern organizes the application into three primary components, which will be detailed below. This separation not only enhances code management efficiency but also significantly simplifies maintenance.

- **Model** – This layer represents the application's data domain and business logic. The model includes the definition of data structures for storing information about flights, images, and reconstructed models. The model also ensures the processing and storage of data obtained from UAVs, as well as integration with external image processing software.

- **View** – This component deals with all aspects related to the UI. The view is responsible for displaying data provided by the model to the user and for capturing user inputs. This involves displaying flight paths, images, reconstruction status, and navigation elements that allow users to interact with the application.

- **Controller** – The Controller acts as a mediator between the model and the view. It manages the flow of data between the model and the view and processes all user interactions. Most importantly, it controls the application's logic, including initiating and monitoring flights, processing inputs from the user, and invoking updates in the model or view as needed.

The use of MVC architecture allows our application to be more flexible, scalable, and easier to test. Each component can be developed and tested independently, which enhances the efficiency of the entire development process and simplifies the addition of new features or updates.

As our work will involve communication with the RealityCapture reconstruction software, which provides an API accessible via internet communication, we will utilize a client-server model where the client is our application and the server is a machine running RealityCapture. This setup does not require global internet access, as it is sufficient to create a Wi-Fi access point from a laptop, allowing communication between the client and server even in remote areas without internet connectivity, such as forests.

Given the use of the DJI SDK, certain minimum application requirements for the hardware and operating system of the mobile device have emerged. Similarly, RealityCapture necessitates a network connection for API communication, compelling us to outline the following **minimum application requirements:**

- **Hardware requirements:** iPhone or iPad with an A10 Fusion chip or newer, with 4 GB of Random Access Memory (RAM) and 64 GB of internal storage.
- **Operating system:** Apple iOS 9 or newer.
- **Network requirements:** Stable internet connection with a minimum speed of 10 Mbps for both upload and download.

These specifications ensure that the mobile application can operate efficiently and leverage the full capabilities of the DJI SDK and RealityCapture software.

2.1.4 Documentation

Within the development of the CR Fly mobile application, we place a strong emphasis on high-quality and accessible documentation, which is essential for the proper understanding and maintenance of the software. The code documentation is created directly in the integrated development environment, Xcode, using the HeaderDoc tool [1]. This tool allows for the generation of detailed documentation from source code comments that describe functions, methods, classes, and other programming elements.

The documentation created using HeaderDoc provides clear and structured information about each software component, which simplifies orientation in the project for new developers or those performing maintenance and updates. Moreover, it ensures that all functionalities are thoroughly documented, contributing to better cohesion and reliability of the entire software solution.

The complete documentation created with HeaderDoc will be included as an Appendix A to this thesis. This will allow readers to view specific details of the implementation and provide a practical example of how to effectively use automatic tools for generating documentation in real projects.

2.2 External interfaces

Given that we design our application to be extendable and modifiable for use with various hardware and software, it can be characterized by its three main components:

- **Core Component:** The core component of the application is tasked with implementing fundamental logic, managing data, and overseeing communication

among various components. The user interfaces, designed to interact directly with data from the core, will ensure efficient management and transmission of information.

- **Drone Component:** This component will ensure bidirectional communication with UAVs. It will not only receive data from UAVs but will also generate and dispatch requests to them. To minimize dependencies between the UAV and the reconstruction software, the Drone Component will not communicate directly with the Scene Component.

- **Scene Component:** Responsible for interfacing with the reconstruction software, this component will handle tasks such as uploading images, initiating reconstruction processes, and transmitting data directly to the mobile phone's display. Similar to the Drone Component, it is designed to restrict communication flow solely through the application's core.

The Drone Component will necessitate a SDK for communication due to its involvement in hardware interaction. Most manufacturers do not provide direct software-based UAV control. Instead, they use a controller that connects to mobile devices via a USB port, facilitating newer and more efficient communication technologies.

In contrast, the Scene Component will interface with software that may adhere to various communication standards, depending on the software solution employed. The specifications of the hardware, software, and communication interfaces will be directly tailored to our software, utilizing the DJI SDK and RealityCapture, to enhance the compatibility and performance of our application.

2.2.1 User interface

The UI of the CR Fly mobile application will be implemented to provide intuitive and efficient control of UAVs during scene reconstruction. It will be characterized by a simple and clean design, with logically organized controls and information, enhancing user comfort and work efficiency.

On the home screen, the current Global Positioning System (GPS) location of the mobile device will be displayed, which is essential for identifying the functionality of determining the pilot's position relative to the UAV. This feature is crucial as the Drone Control View includes a map showing both the pilot's and the drone's locations. This capability facilitates the identification of the drone's position if the pilot loses visual contact. From the screen, users will be able to easily navigate to three main views:

- **Drone Control View:** This view will incorporate a live video feed from the UAV camera, accompanied by all essential components for UAV control. It will

be expanded with camera settings to produce the highest quality images and basic UAV settings supported by the UAV model. The interface will become accessible only after the device has been successfully connected to the mobile device, ensuring communication with the UAV is established.

- **Photo Album View:** This view will be designated for managing photos stored on the UAV and on the mobile device. It will allow performing basic life management operations such as preview, save, delete, download images to the mobile device, and upload images to the reconstruction software.

- **3D Scene View:** This view will facilitate user interaction with 3D reconstruction software. It will display a visualization of the reconstructed object, either as a mesh consisting of a point cloud, which is formed from vertices identified across multiple photographs, or as a triangulated mesh that represents a fully realized 3D object. In this latter format, vertices are interconnected to establish a coherent and textured 3D structure. Additionally, this view will include a link to return to the Photo Album View, allowing users to select and upload photographs that will be incorporated into the reconstruction process.

Interaction with the application will be facilitated through a touchscreen interface, allowing users to engage with the application via buttons and text fields for streamlined operation. The buttons will be large and have maximum contrast, simplifying their use even in challenging conditions such as direct sunlight, rainy weather, or dust. These features guarantee that application control remains both intuitive and effective, particularly for users in field conditions where screens can be challenging to read and interfaces difficult to manipulate due to environmental influences.

The application will support display in dark and light modes, which will mirror the settings of the mobile device, making the application suitable for both light and dark environments. The design of each interface is optimized for responsiveness, ensuring that the application maintains both aesthetic appeal and functional integrity across a range of devices, from smartphones to tablets.

2.2.2 Hardware interface

Our work will implement hardware communication with DJI UAVs using the DJI SDK. The specific version of the DJI SDK that will be used in the implementation supports and requires the UAV – DJI Air 2S and older models. All supported UAVs are listed on SDK's documentation website[1].

Since the application does not necessarily require a connection to the reconstruction software, we can consider this part as a recommendation. Should the decision be made

[1] `https://developer.dji.com/document/2c6f3a26-412e-45d2-a312-eb82e72411e7`

to utilize the application for reconstruction purposes as well, an additional limitation will be introduced by the RealityCapture software. This software requires a 64-bit machine with 8GB RAM, Windows 7 or later, an NVIDIA graphic card with 1GB video RAM and CUDA[2] 3.0+, and a 10 Mbps internet connection. Updates to these hardware requirements can be monitored on the developer community page provided by Epic Games.[3]

2.2.3 Software interface

To leverage the reconstruction capabilities offered by RealityCapture, it is essential that this software is installed on a computer or server accessible to the device operating the application. This accessibility can be ensured either through a connection to the same local network or via an internet connection, facilitating communication over broader network infrastructures.

Furthermore, it is advisable to ensure that any UAV being operated by our application is equipped with the most recent firmware versions officially released by the manufacturer. Updating the UAV firmware not only enhances the operational capabilities of the drone but also ensures compatibility with the latest enhancements and security features introduced by the manufacturer. This practice minimizes potential technical issues and maximizes the performance and reliability of the UAV.

2.2.4 Communication interface

Communication with the UAV will be facilitated through the use of objects and methods that adhere to the specifications outlined by the DJI SDK. These objects and methods can be strategically delegated within the SDK's shared instance, enabling the effective monitoring of changes in the drone's status. This gives us direct control over reading and changing the state of the UAV, such as camera settings, changing the shooting mode, or altering the UAV's GPS coordinates.

Communication between the client (application) and the server (RealityCapture) will be conducted via the Hypertext Transfer Protocol (HTTP). The server will receive communication through RealityCaptureNode, which is capable of controlling RealityCapture and providing an API with precise specifications for GET and POST requests. Should there be a change in status on the RealityCapture side, the data in our application will also be updated.

[2] `https://developer.nvidia.com/cuda-toolkit`

[3] `https://dev.epicgames.com/community/learning/knowledge-base/Wj7B/capturing-reality-realitycapture-os-and-hardware-requirements`

2.3 System features

2.3.1 Error handling

The application will be designed to remain responsive in the event of an unexpected error. It will undertake specific steps aimed at resolving the error. Below are four common issues that may occur and their corresponding handling strategies:

- **Loss or interruption of connectivity:** In the event of connectivity issues, the application will not notify the user of each interruption. Instead, it will attempt to re-establish the connection and resume operations to restore the previous state. If the connection cannot be restored within a few tens of seconds, the application will conclude that a complete loss of connectivity has occurred and will prepare to establish a new connection.

- **Errors originating from the DJI SDK or RealityCapture:** Should errors arise from the DJI SDK or RealityCapture, the application will attempt to retry the task. If the error persists after multiple attempts, a notification will be sent to the user. These errors may occur due to system overload, with no expected response forthcoming.

- **Dependency error in task execution:** When a series of tasks are interdependent, and one of them fails, subsequent dependent tasks will be automatically canceled to prevent compounding errors.

- **Application recovery after returning from background:** If the application reactivates after being in the background, it will attempt to restore the user's original session. To achieve this and prevent errors, the system will detect such changes and maintain a memory of tasks assigned to the application.

To enhance error detection, repeatability, and traceability of changes, the Command Design Pattern will be implemented. This approach will facilitate the resolution of issues or help avoid them altogether. Conversely, errors that cannot be resolved will be communicated to the user through alerts.

2.3.2 Security

The security measures for drone communications are inherently incorporated within the SDK used. Therefore, our primary focus will shift to RealityCapture, whose network communications could potentially be intercepted.

Rationale for using HTTP

The decision to use HTTP instead of Hypertext Transfer Protocol Secure (HTTPS) is primarily influenced by the application's design for communication within a local network. Importantly, the data transmitted over these networks are not of a sensitive nature, thus not necessitating encrypted communications under current usage scenarios. However, the benefits of local networks go beyond security, offering significantly faster transmission speeds which are advantageous for real-time data processing.

Security considerations in a local area network

In local area network environments, it is essential to recognize that although HTTP does not inherently include encryption, the configurations of the internal network, including firewalls and network access controls, are crucial in safeguarding communications. Should the demands for network security increase, the implementation of additional protective measures, such as Virtual Private Network (VPN) or encrypted network segment, may become necessary.

Future proofing for HTTPS

RealityCapture supports API communication via Secure Sockets Layer (SSL), but requires keys, which is considered outside the scope of this work. Although, looking to the future, if the expansion of the application's scope requires internet-based communication, the implementation of HTTPS will become imperative. The plan is to integrate HTTPS seamlessly, enabling it automatically when the application detects that communication extends beyond the local network. This proactive strategy guarantees that the application remains secure and flexible, adapting to changes in network environments and user requirements.

Chapter 3

Implementation

This chapter delineates the implementation details of the CR Fly mobile application, which are pivotal for facilitating robust interaction between the UAV and the reconstruction software. It encompasses an examination of the system architecture, communication protocols with both the UAV and the reconstruction software, user interface design, and the strategies employed for testing and optimizing the application. Each section aims to furnish a comprehensive understanding of the configuration and interaction of the system's components, thereby clarifying the technical foundations underpinning the application's functionality and performance. Moreover, this detailed exploration not only illustrates the practical implementation of the theoretical concepts discussed in previous chapters but also sets the stage for the experimental validation described in the subsequent chapter.

3.1 System architecture

To familiarize ourselves with our software, we will provide an overview of the overall architecture of the application and the structure of its components. This overview will be detailed in subsequent sections, enabling a better understanding of the interactions between individual components and the identification of parts that offer modifiable aspects of the implementation.

Utilizing Unified Modeling Language (UML) diagram in Figure 3.1 and concise descriptions of each part, we will provide insights into how the various components of the application are organized and how they interact with one another. This overview is crucial for understanding the fundamental principles that govern the operation of the entire application. Additionally, it aids in illustrating the modular nature of the software, highlighting how components can be independently modified or enhanced to adapt to evolving requirements. This approach not only elucidates the structural complexities but also highlights the design considerations crucial for ensuring robustness

and scalability within the software's architecture. It is important to note that the presented UML diagram does not encompass the complete structure of the application but rather focuses on key classes whose implementation and architecture are crucial. This selective representation is designed to highlight the essential elements that are most impactful for the application's functionality. By concentrating on these critical components, we provide a clearer and more manageable understanding of the application's architecture without overwhelming details, ensuring that the foundational concepts are comprehensively conveyed and easily grasped.

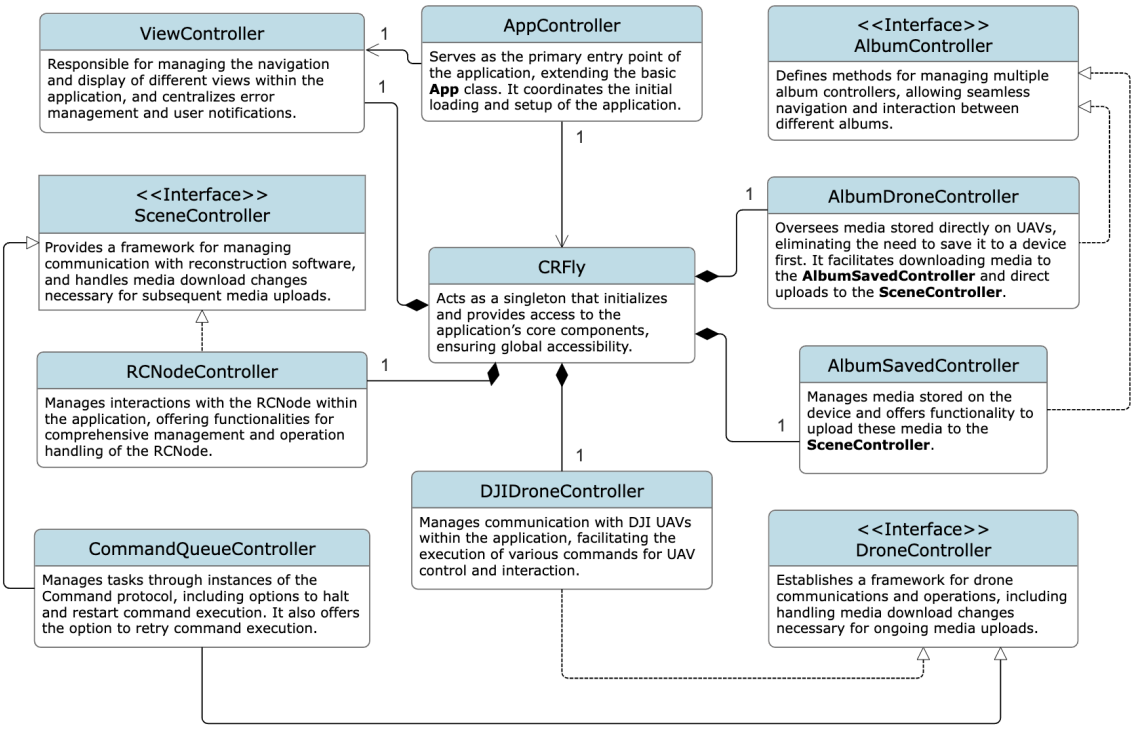

Figure 3.1: UML diagram that details the architecture of the CR Fly

During the design of the application, we encountered specific communication limitations that influenced our architectural decisions. For instance, in the case of RealityCapture, which interacts via an API using the HTTP, it is not feasible to execute two independent requests simultaneously. Although it supports communication with multiple clients at once, it processes tasks serially, so the use of parallelism would not benefit our case. Additionally, when communicating through a single connection, we cannot send two independent network packets simultaneously.

Similarly, the DJI SDK does not support parallel operation – while downloading media through this SDK, it was also impossible to load new content into the album.

These constraints necessitated the development of a universal class, designed to manage and queue commands effectively, thereby circumventing the issues related to parallel command execution.

3.1.1 CommandQueueController

This class is a key component of the application, as it manages the majority of tasks via implementations of the *Command* protocol, which are maintained by its queue. The primary function of this class is the synchronous execution of these commands and verification of their successful completion. It also offers the possibility of task repetition in case of failure and ensures the display of errors in the UI. The most essential capability of this class is to halt command execution, which is necessary in the event of connection loss, and to potentially schedule commands for execution upon reconnection.

Command is a protocol that prescribes the mandatory structure for its implementations (commonly referred to as an interface). It requires that its implementations incorporate the following method:

> ***func execute(completion: @escaping(success: Bool, retryable: Bool, error: (String, String)?) -> Void)***

This method encapsulates the executable code for the command and involves a single parameter, `completion`, which is expected to be called upon the command's completion. Through this closure, we can specify the action to be taken once the execution completes (in our case, the *CommandQueueController* verifies the success and initiates the execution of the next command). For such verification, the `error` parameter is required, defined as a tuple where the first element is the error name and the second is its description. The parameters `success` and `retryable` are considered intuitive and self-explanatory. Lastly, the **@escaping** notation is crucial as it allows the closure to be stored and used at a later time, which is particularly vital for operations that do not complete immediately, such as network requests or extensive data processing tasks.

As outlined in Section 2.1.3, to ensure scalability and modularity, we can characterize the application by its three main components: Core, Scene, and Drone. In the following sections, we will explore each of these components in detail, highlighting their specific functions and interactions within the system.

3.1.2 Core Component

The model of the Core Component is composed of seven data classes, each capable of notifying the corresponding view which, in turn, redraws itself if currently displayed. This division into seven independent classes ensures that data, which do not affect the components displayed in the current view, do not trigger unnecessary redrawing.

Additionally, it defines the *MediaTransferState* protocol, which is crucial for defining diverse implementations of the media transfer state across different system components, each tailored to specific operational needs.

The view within the Core Component includes multiple views, each designed according to specifications and intended to operate independently of the Scene and Drone Component implementations.

For enhanced scalability, the Photo Album View facilitates the management of multiple albums. This is accomplished through the *AlbumController* protocol, whose implementations display content from designated albums.

To address the specifications requiring media management both on the device and on the UAV, we developed the *AlbumSavedController* and *AlbumDroneController*, which incorporate all necessary functionalities for efficient media management.

The controller is pivotal as it orchestrates the data flow and UI interactions. It comprises several key classes:

- **AppController** implements the *App* protocol, serving as the gateway for application initialization and the integration of the DJI SDK, which is essential for accessing drone functionalities. This controller contains the `body` variable, which renders content on the device's screen and monitors the application's state transitions, coordinating with other controllers as needed.

- **ViewController** manages the content displayed by the *AppController*. It incorporates a sophisticated navigation system that logs user transitions between views, providing the option of navigating to any desired view and seamlessly returning to the original one. Additionally, this controller dynamically maps views based on their types, specifically designed to ensure that if a user navigates away from a view and subsequently returns, the view will maintain its previous state. It also centralizes error reporting and the status of media transfers, ensuring consistent and accurate updates. These functionalities collectively enhance navigational efficiency and significantly improve the overall user experience.

- **AlbumController** defines essential methods and properties for its implementations, facilitating media loading into albums and generating displayable content within *AlbumView*. It plays a central role in media management across the application.

- **AlbumSavedController** implements the *AlbumController* protocol, offering functionalities for managing the content of media stored on the device.

- **SceneController** describes how the Core Component communicates with the Scene Component. It specifies that the class implementing it must contain methods capable of handling changes in media downloads and responding by adjusting uploads. For instance, if an upload is contingent on downloading a particular media that was canceled, the upload will also be canceled. It also specifies data classes and methods that must be implemented, as they are utilized in the UI to facilitate interactions with the Scene Component.

- **DroneController** specifies the manner in which the core will communicate with the Drone Component. It does not impose as stringent conditions as the *SceneController* due to the impracticality of creating a universal view for UAV control. However, it provides methods for managing the media download state, such as resuming, pausing, and canceling downloads.

3.1.3 Drone Component

The model, unlike the Core Component's model, includes only the implementation of the *MediaTransferState* protocol, which manages the media download process. This protocol maintains crucial information necessary for overseeing the entire download process effectively.

The view comprises a single, specifically designed Drone Control View, crafted according to specifications. The creation of a universal view within the Core Component was deemed impractical due to limitations imposed by the DJI SDK.

The controller in the Drone Component comprises controllers specifically designed for communication with DJI UAVs, including:

- **AlbumDroneController** is a specialized implementation of the *AlbumController* protocol that facilitates the creation and management of new album for images stored directly on DJI UAVs.

- **DJIDroneController** implements the *DroneController* protocol and is designed to interact with the necessary classes and data within the Core Component. This interaction often involves using methods defined within the Core Component or modifying its data class to ensure seamless communication and functionality. Its primary function is to delegate methods directly to the classes provided by the DJI SDK, enabling it to establish and maintain communication with the UAV and execute user commands. Additionally, this delegation simplifies the process of obtaining a live video feed from the UAV's camera, allowing for the live display of this stream.

3.1.4 Scene Component

The model, similar to the model in the Drone Component, includes an implementation of the *MediaTransferState* protocol, which manages the media upload process. This protocol also maintains crucial information necessary for overseeing the entire upload process, specifically tailored for media uploads to RealityCapture.

The view does not contain any specific views as they are not required within this component. The display of relevant views is handled by the Core Component, and the generation of project information is managed by the implementatio of SceneController.

The controller in this component contains a single controller specifically designed for communication with RealityCapture:

- **RCNodeController** is a class that implements the *SceneController* protocol and communicates with the Core Component similarly to the *DJIDroneController*. This controller is responsible for establishing, maintaining, and terminating connections and informing the corresponding view of these actions by modifying the data class in the Core Component. Additionally, within its implementation of the *SceneController* protocol, it provides project management functionalities for RealityCapture.

3.1.5 Architecture summary

As we have previously suggested, the *CommandQueueController* class plays a pivotal role in communication with external interfaces. Consequently, the *DroneController* and *SceneController* protocols will require that classes implementing them inherit from the *CommandQueueController*. This requirement underscores the importance of conducting communication synchronously. However, should communication occur via external interfaces that support parallelism, modifications to the implementation of this class would be necessary. In addition to ensuring synchronicity, we have created a control point within which we can monitor, limit, and manage the flow of communication.

This strategic approach not only enhances modularity and maintainability but also effectively addresses key issues previously outlined in the specification, which will be discussed in the following subsections.

Errors originating from the DJI SDK or RealityCapture

The described implementation offers the advantage of allowing requests to be repeated in the event of an unexpected server-side error through a retryable parameter, while invoking the ***completion*** method within the *Command*'s execute method.

Dependency Error in Task Execution

It is crucial not to overlook that the structure in place ensures that at the start of each new command, verification is possible to determine whether the previous command has fulfilled the prerequisites required by the subsequent one.

3.2 Command execution

The *CommandQueueController* class, along with the *Command* protocol, has been previously described. Our interest lies in how to execute an implementation of this protocol effectively. In Listing 3.1, we can observe how a command is simply created.

```
public class ExampleCommand: Command {
  public func execute(completion: @escaping (Bool, Bool, (String, String)?) -> Void){
    if 1==1 {
      print("Hello World!")
      completion(true, false, nil)
    } else {
      completion(false, true, ("What happened","Unknown Error"))
    }
  }
}
```

Listing 3.1: Example of command implementation

The sole purpose of this command is to print "Hello World!" to the console during its execution. The parameters specified in call ***completion(true, false, nil)*** inform the method that the command execution was successful. Conversely, an alternate independent call ***completion(false, true, (...))*** would indicate to the method that the execution was unsuccessful but can be retried.

In Listing 3.2, we illustrate the process by which this command is executed.

```
public class CommandQueueController {
  public func pushCommand(command: Command) {
    this.commandQueue.append(command)
    if !self.isExecutingCommand && self.commandExecutionEnabled {
      self.processNextCommand()
    }
  }

  public func runExampleCommand() {
    self.pushCommand(command: ExampleCommand())
  }

  public func processNextCommand() {
    if !self.commandQueue.isEmpty {
      self.commandQueue[0].execute(completion: {(a1,a2,a3) in print("Status: \(a1)")})
    }
  }
}
```

Listing 3.2: Source code clipout of CommandQueueController class

Upon invocation of the ***runExampleCommand()*** method, the specified command is enqueued. Subsequently, the ***processNextCommand()*** method is executed, which in turn invokes the ***execute(...)*** method with the predefined ***completion*** parameter. The completion handler, invoked with specific parameters, assesses the success of the command's execution. Based on this assessment, it determines the subsequent actions – it may execute another command, retry execution of current command or display an error message when provided the number of attempts has surpassed the retry limit.

Each command implementation is comprehensively documented in the attached documentation (see Appendix A).

3.3 Communication with the UAV

Let us analyze the communication process with the drone from connection initiation to command execution. Establishing a connection with the drone, along with detecting UAV connectivity, can be performed as follows:

```
public class DJIDroneController: CommandQueueController, DroneController,
  DJISDKManagerDelegate {
  // Data class that holds information about drone connection and download status
  public let droneData = DroneData()
  public func connectToProduct(){
    // Check if the device is already connected to prevent SDK Errors
    if(self.droneData.deviceConnected) { return }
    DJISDKManager.stopConnectionToProduct() // Stop any previous attempts to connect

    // Attempt to start a new connection to the UAV
    DJISDKManager.startConnectionToProduct()
  }

  // This function is delegated to DJISDKManager and handles the connection of
  // components such as the UAV and the UAV's camera.
  public func componentConnected(withKey key: String?, andIndex index: Int) {
    // Initialize connection if the UAV is newly connected and valid
    if(!self.droneData.deviceConnected && DJISDKManager.product() != nil
      && DJISDKManager.product()!.model != "Only RemoteController"){
      self.droneConnected() // Begin initialization of the connection
    }
  }

  // Also delegated to DJISDKManager and handles the disconnect of components.
  public func componentDisconnected(withKey key: String?, andIndex index: Int) {
    // Cancel connection if the UAV is newly disconnected or invalid
    if(self.droneData.deviceConnected && (DJISDKManager.product() == nil
      || DJISDKManager.product()!.model == "Only RemoteController")){
      self.droneDisconnected() // Begin cancellation of the connection
    }
  }
  // Additional functions are omitted for brevity...
}
```

Listing 3.3: Source code clipout of DJIDroneController showing connection handling

This clipout outlines the interaction of the code with library calls via the shared *DJISDKManager* class. It begins by invoking a method to initiate a search for a UAV connection and delegates methods to monitor connection and disconnection events, ensuring immediate notification of any changes in connection status. When a connection is established, the **droneConnected()** method enables the execution of commands in the queue by setting the `commandExecutionEnabled` variable to true. Conversely, if a disconnection occurs, the **droneDisconnected()** method sets this variable to false, effectively disabling command execution.

Once the UAV is connected, communication is facilitated through command implementations that reside in the project directory at **"CR Fly/DroneComponents/-DroneCommands"**. This directory can be accessed in Appendix B.

3.4 Communication with RealityCapture

As previously mentioned in Section 2.1.3, communication with RealityCapture is conducted via the HTTP, which does not require maintaining an active connection. The standard library of the Swift language does not contain classes capable of capturing connection states. To fully utilize the features provided by the *CommandQueueController* in the implementation of *RCNodeController*, we were compelled to implement a class capable of communicating with the server via the HTTP while also notifying about the state of the connection.

This necessity led to the development of the *HTTPConnection* class to ensure the required functionalities were met. Its implementation includes creating an instance of *NWConnection* from Swift's standard library, which can establish a Transmission Control Protocol (TCP) connection with the server and send packets through it. Once the connection with the server is successfully established, the *HTTPConnection* class allows us to use methods we implemented for sending and receiving data via the HTTP. It also supports the creation of a custom handler for connection status changes, prompting us to develop the *HTTPConnectionStateObserver* protocol, which notifies the implementing class of any connection updates.

Communication through the command queue is initiated only after a stable and active connection has been established, ensuring that all command transmissions are reliable and secure. All implemented and usable commands that communicate with RealityCapture can be found in the project directory at **"CR Fly/SceneComponents/SceneCommands"**. This directory can be accessed in Appendix B.

The implementation details of the initialization and notification processes conducted by the *HTTPConnection* class are presented:

```
public class HTTPConnection {
  private var stateObservers: [HTTPConnectionStateObserver] = []

  public init(host: String, port: UInt16, ... ) async throws {
    // Define connection parameters with specified timeout settings
    let options = NWProtocolTCP.Options()
    let params = NWParameters(tls: nil, tcp: options)

    // Initialize the network connection with the provided host and port
    self.connection = NWConnection(host: NWEndpoint.Host(host),
      port: NWEndpoint.Port(rawValue: port)!, using: params)

    // Setup a state update handler to manage changes in connection state
    self.connection.stateUpdateHandler = { state in
      switch state {
        case .ready:
          // When connection is ready, proceed with operations
          self.connectionReadyContinuation?.resume()
          self.httpConnectionState = .connected
        case .failed(let error):
          // Handle failures or waiting state by passing the error
          self.connectionReadyContinuation?.resume(throwing: error)
          self.httpConnectionState = .disconnected
        case .waiting(let error):
          self.connection.forceCancel()
          self.connectionReadyContinuation?.resume(throwing: error)
        case .cancelled:
          self.httpConnectionState = .disconnected
        default:
          break
      }
      // Notify all observers of the current state change
      self.notifyObserversStateChange()
    }
    // Start the connection with a global dispatch queue
    connection.start(queue: .global())

    // Await the connection to be fully ready or end in error state
    try await withCheckedThrowingContinuation
    { (continuation: CheckedContinuation<Void, Error>) in
      self.connectionReadyContinuation = continuation
    }

    // Reset the continuation once the connection is established
    self.connectionReadyContinuation = nil
  }

  private func notifyObserversStateChange(){
    for observer in self.stateObservers {
      // Notify all registered observers about the current connection state
      observer.observeConnection(newState: self.httpConnectionState)
    }
  }
}
```

Listing 3.4: Source code clipout of HTTPConnection initialization demonstrating the setup and management of TCP connections

3.5 User interface

In the preceding chapter, Section 2.2.1, the requirements for the UI implementation were delineated. This section aims to visually demonstrate how these requirements have been achieved within the application by including screenshots that are optimally selected to elucidate the UI. These images not only illustrate the individual views but also provide critical insights into the functionality, aesthetics, and user interaction mechanisms embedded in the UI. Screenshots serve as a pivotal component of the documentation, offering a tangible representation of the abstract concepts discussed and facilitating a deeper understanding of the application's operational dynamics.

3.5.1 Home Screen

Before we proceed to the specified views, we first examine the foundational interface displayed immediately prior to entering the application. This screen serves as a transitional interface between the other views. The background image used was generated using ChatGPT [8].

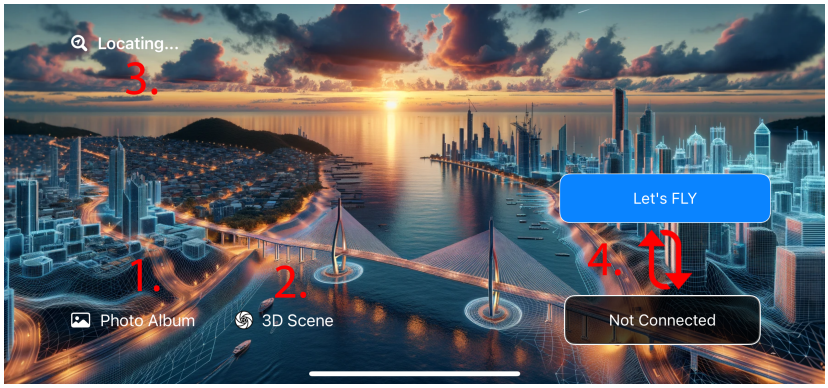

Figure 3.2: Home Screen Screenshot – A screenshot of the Home Screen illustrates the initial interface encountered by users, highlighting the primary functions and navigational aids of the application.

The elements marked in Figure 3.2 are described as follows:

1. **Photo Album View access:** This button allows users to switch to the Photo Album View, facilitating the management and review of stored photographs.
2. **3D Scene View access:** This button enables navigation to the 3D Scene View, allowing users to interact with and explore 3D reconstructed scenes.
3. **Location indicator:** This feature displays the current location of the user's device. For optimal functionality, it requires that location services be enabled, which ensures accurate navigation and enhances context-aware features within the application.

4. **Connection status button:** This button reflects the UAV's connection status. When a connection is established, it changes to the "Let's Fly" button, enabling the user to transition to the Drone Control View for active flight and management. If the UAV is not connected, it displays "Not Connected", prompting the user to troubleshoot or re-establish the connection.

3.5.2 Drone Control View

The Drone Control View is a versatile library component provided by the DJI SDK which supports extensive customization to meet specific application needs. It enables comprehensive UAV control, providing the user with real-time information regarding the UAV's operational status. Additionally, this interface offers a variety of adjustable settings, enhancing the ability to customize behavior for specific tasks or conditions. Further details on these settings and their implications for operation will be elaborated in subsequent sections.

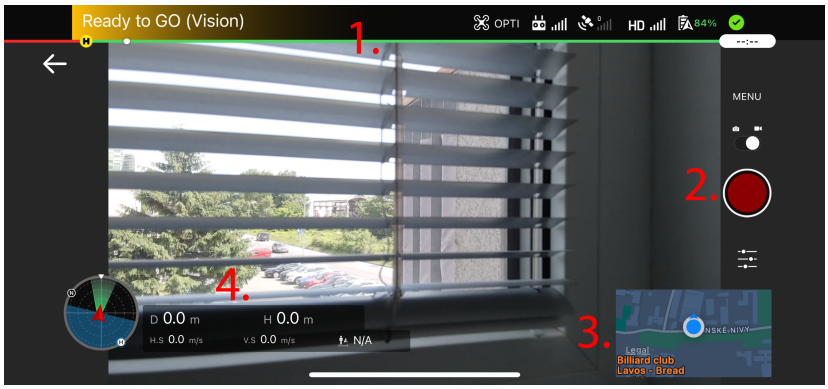

Figure 3.3: Drone Control View Screenshot – This image showcases the customized, versatile library component provided by the DJI SDK, illustrating the user interface used for detailed UAV control and adjustments.

The key components highlighted in Figure 3.3 are detailed below to illustrate their specific functionalities within the Drone Control View:

1. **Drone status bar:** This component displays critical information about the UAV's current status, including battery life, signal strength, and pre-flight checklist. It is designed to provide quick, at-a-glance insights that are essential for safe and efficient operation, ensuring the operator is always aware of the UAV's health and readiness.

2. **Interactive camera controls:** These controls offer the operator the ability to adjust camera settings dynamically during flight. This includes modifications to zoom, focus, exposure, and camera orientation. Additionally, operators can

switch between photo and video modes, allowing for versatile media capture that suits different documentation and inspection needs. The interactive nature of these controls allows for precise adjustments to be made in response to varying conditions and requirements, thereby enhancing the quality of the imagery captured and providing flexibility in how visual data is collected.

3. **Location map and heading indicator:** This feature displays a small, interactive map that shows the current geographical position of the UAV, marked by a blue dot. This tool is vital for navigation, especially in complex environments or when the UAV is out of visual line of sight. It helps operators ensure that the UAV is on the correct flight path and assists in making informed navigational decisions.

4. **Distance and speed metrics:** This component displays essential flight metrics, including horizontal speed (H.S), vertical speed (V.S), distance (D), and height (H) from the take-off point, all measured in meters or meters per second as appropriate. This information is crucial for maintaining control during flight, particularly in precision tasks or when maneuvering in tight spaces. It ensures that the operator is continually updated on the UAV speed and distance metrics, enabling more accurate and safe UAV operations.

3.5.3 Photo Album View

The Photo Album View is designed as a user-friendly interface for managing and reviewing all captured photographs and videos stored either on the UAV or on a local device. This view supports sorting, tagging, and organizing media, features that are particularly beneficial for simplifying media selection. It supports batch operations, including deletion, downloading from the UAV, and uploading to RealityCapture.

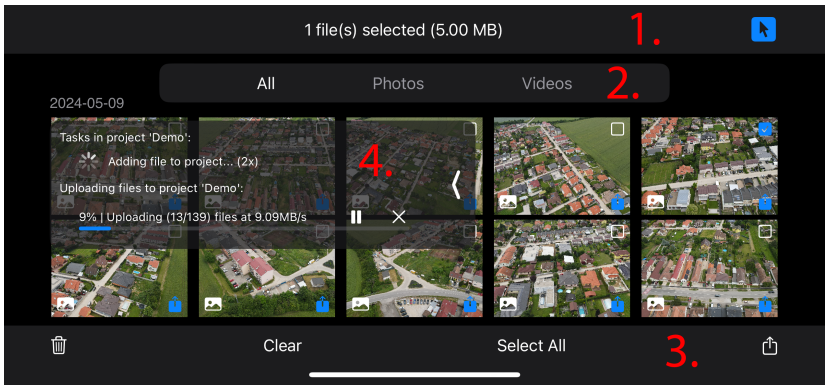

Figure 3.4: Photo Album View Screenshot – This image depicts the view in media selection mode during an active upload process.

The Photo Album View, detailed in Figure 3.4, streamlines media management that enhances user interaction and operational efficiency by incorporating the following key features:

1. **Album status bar:** This bar allows users to toggle between the UAV album and the stored photos album, and it facilitates the switch to selection mode, where it displays data about the selected media. Additionally, it allows the inclusion of photos from mobile devices for uploading to RealityCapture.

2. **Album filter:** This feature permits users to apply filters for sorting or locating specific media within the album, thereby enhancing the manageability of media. The available filters allow for segregation by media type, enabling efficient organization and access based on content type.

3. **Album selection option bar:** This bar provides options for managing selected media, including deletion, uploading to RealityCapture, and downloading from the UAV, thus optimizing workflow efficiency.

4. **System-wide status indicator:** This indicator displays real-time information on ongoing downloads/uploads and the status of tasks in RealityCapture. It is consistently visible throughout the application and can be concealed via a button located on the right edge.

3.5.4 3D Scene View

The 3D Scene View offers a sophisticated platform for visualizing meshes and 3D models, complete with project management features customized for RealityCapture. This interface allows users to manipulate the models by zooming, rotating, and dissecting, enabling detailed analysis of specific aspects of the terrain or structure.

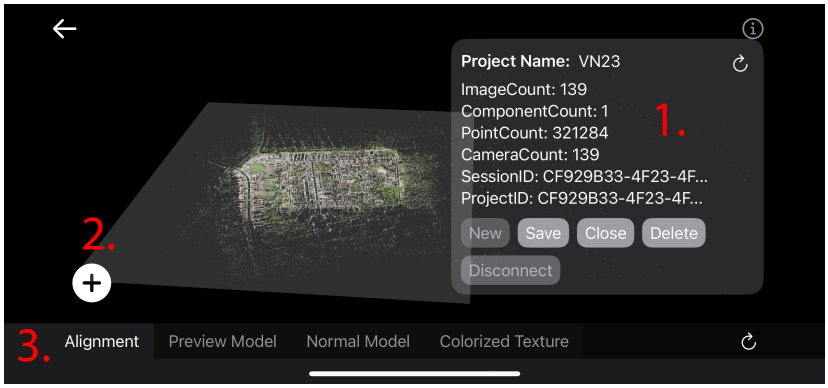

Figure 3.5: 3D Scene View Screenshot 1 – This screenshot captures fundamental control elements such as project management and scene mode switching.

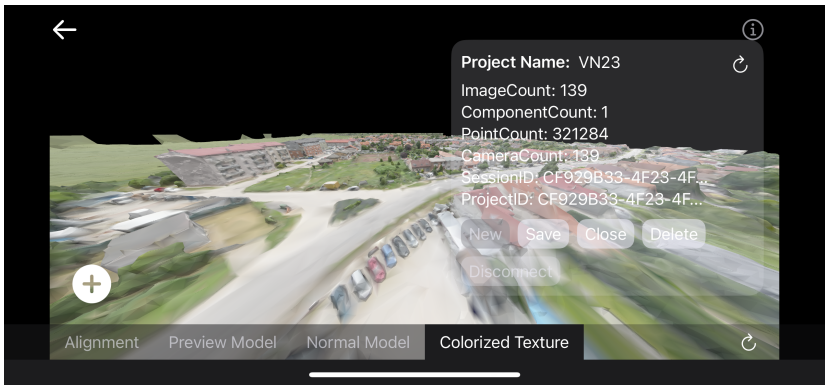

Figure 3.6: 3D Scene View Screenshot 2 – This screenshot displays a reconstructed model of a village, showcasing the colorized texture mode.

The 3D Scene View enhances 3D modeling and project interaction, as shown in Figures 3.5 and 3.6. It incorporates:

1. **Project management bar:** Displays comprehensive project details that assist in analyzing the quality and coverage of the images used in reconstruction.

2. **Shortcut to Photo Album View:** This shortcut enables immediate access to the Photo Album View from within the 3D Scene View, allowing users to efficiently add photographs to the project that will be used in the reconstruction process.

3. **Model type selection bar:** Offers various modes for viewing the 3D model, including alignment checking, low-detail preview, normal, and colorized texture models. This selection bar enables users to choose the appropriate visualization mode for their specific analysis or presentation needs. Additionally, located at the right corner of the bar is a button for initiating the reconstruction process anew, enhancing workflow flexibility.

3.6 Testing and optimization

Testing is a cornerstone of software development that ensures an application meets its specified requirements and performs reliably in various conditions. For this project, a comprehensive approach was adopted that combined manual testing with automated unit and mock tests, ensuring a broad coverage of the majority potential use cases and scenarios.

3.6.1 Manual testing

Manual testing was prioritized to mimic real-world usage as closely as possible. This hands-on approach involved:

Scenario-based testing – Running through all potential user scenarios to ensure every aspect of the application functioned as expected. This included testing typical day-to-day operations as well as less common tasks to ensure comprehensive coverage.

Edge case testing – Deliberately testing the limits of the application by simulating rare conditions and potential error states that could occur in production. This helped identify and resolve issues that Scenario-Based tests might miss, ensuring the application's stability and robustness under unusual conditions. The success of these tests confirmed the application's functionality and usability in realistic settings, significantly enhancing user confidence in the product.

3.6.2 Automated testing

In addition to manual testing, automated tests played a crucial role in maintaining the integrity of the application, especially during code expansion. Automated testing ensured that new changes did not disrupt existing functionalities. This section details the specific automated testing methodologies used, namely unit testing and mock testing, which were critical in ensuring the application's robustness and reliability.

Unit testing: Focused tests were developed for critical components using Swift's XCTest framework. These tests were designed to ensure that individual units of code performed correctly in isolation from the rest of the application. Unit tests were instrumental in verifying the discrete functionality of modules, helping to quickly identify and isolate faults during development.

Mock testing: Certain components required interaction with external systems, which were not always accessible for testing. Using mock objects, we simulated these external interactions to test the interfaces and data handling capabilities of our components. This approach allowed for comprehensive testing of system logic without the need for actual external dependencies, thus speeding up the testing process and reducing external influences on test results.

3.6.3 Optimization

Following the testing phases, optimization efforts were made to address any performance issues discovered. This included refining algorithms and enhancing the efficiency of data processing, which improved both the speed and responsiveness of the application.

An example of necessary optimization involved the use of the 'Simplify' functionality in RealityCapture. This step was crucial because the reconstruction process generated models of immense size and detail when dealing with point clouds containing more than 100,000 points. Such complexity led to application crashes when attempting to display these models. Implementing the 'Simplify' feature effectively reduced the model complexity, ensuring stable performance and preventing the application from crashing during visualization.

3.6.4 Summary and impact of testing

The dual strategy of combining manual and automated testing methodologies ensured a thorough validation of the application across all scenarios. Manual testing allowed for real-world emulation and direct feedback on the application's performance, while automated tests provided continuous regression checks. This comprehensive testing not only prepared the application for successful deployment but also established a robust framework for future updates and enhancements.

3.7 Potential enhancements to the application

As technology and user needs evolve, there are numerous opportunities to enhance the application's functionality and user experience. One significant improvement could involve the development of a smart autonomous flight capability, which uses real-time 3D data to enhance the efficiency and effectiveness of aerial photogrammetry tasks.

3.7.1 Test coverage

To further improve the reliability and robustness of the application, enhancing test coverage is essential. Expanding the range and depth of test scenarios will ensure that more functions are evaluated under a variety of conditions, reducing bugs and improving overall software quality.

3.7.2 Model rendering using level of detail

Improving the visualization of 3D models without the need to simplify them can be achieved by implementing Level of Detail (LOD) rendering techniques. This approach allows for dynamically adjusting the complexity of a 3D model's rendering based on the viewer's distance, which can significantly enhance performance and maintain visual fidelity.

3.7.3 Autonomous flight functionality

The proposed enhancement involves introducing an autonomous flight mode to the application. This mode would operate in several phases to optimize data collection and model reconstruction:

1. **Initial (manual) flight and data capture:** Initially, the user would manually pilot the UAV to conduct a preliminary survey flight. Alternatively, this flight could be performed autonomously by selecting an area on the map, which would initiate an autonomous flight in a grid pattern. During both manual and autonomous flights, the UAV would perform imaging at timed intervals and data collection, ensuring comprehensive coverage of the targeted area.

2. **Data reconstruction and path planning:** The images obtained during the initial flight would be quickly processed and reconstructed into a 3D model of the scene. Using this model, the application would perform an analysis to determine the optimal flight path for further data collection. This path aims to cover all necessary angles and areas that require additional detail or were underrepresented in the initial pass.

3. **Path review and approval:** Before the autonomous flight is executed, the user would have the opportunity to review and approve the suggested path. This step ensures that the proposed route meets all safety and coverage requirements, providing an additional layer of oversight.

4. **Autonomous flight execution:** Upon approval, the UAV would autonomously follow the planned route, capturing photographs at strategic points where capturing images would yield the highest quality data,. This phase would be fully automated, with the UAV adjusting its flight pattern as needed to optimize image quality and coverage.

5. **Final model export:** After the autonomous flight, the collected data would be used to update or refine the existing 3D model. Finally, the final model would be exported, providing the user with a high-quality representation of the surveyed area.

This autonomous flight functionality would not only streamline the process of data collection and model generation but also enhance the accuracy and reliability of the 3D models produced. By automating the flight path based on initial reconstructions, the application can ensure optimal photo coverage and significantly reduce the manual effort required from the user. Additionally, this feature would expand the application's usability in complex or large-scale surveying projects, where manual flight planning and execution might be impractical or inefficient.

Chapter 4

Experiments

In this chapter, we propose and complete three experiments utilizing the CR Fly application to reconstruct real-world objects. Each experiment includes a detailed procedure and the motivation for selecting the specific object for reconstruction. The aim is to demonstrate the practical application of the CR Fly mobile application in diverse environments and evaluate the effectiveness of the data collection and reconstruction processes facilitated by the application.

4.1 Hardware setup

For the successful execution of these experiments, the following hardware was employed:

- **Computer:** The RealityCapture software was run on a computer equipped with an Intel i7-10700K 3.8 GHz processor, 16 GB of RAM, and an NVIDIA Quadro RTX 6000 graphics card. This setup ensured efficient processing and high performance during the reconstruction tasks.

- **UAV:** A DJI Mavic Air 2 drone was used for capturing the images. It is equipped with a 1/2-inch CMOS sensor capable of capturing 48MP still images and 4K video at 60fps, providing detailed and clear images necessary for accurate 3D reconstruction. Additionally, the drone can fly for up to 34 minutes on a single charge, allowing sufficient time to capture the required images.

- **Mobile device:** The CR Fly mobile application was operated on an iPhone 13 Pro, which provided robust performance and seamless integration with the DJI Mavic Air 2.

The following experiments were selected to demonstrate and highlight the applicability and flexibility of the CR Fly application in two distinct scenarios. One scenario requires detailed reconstruction of an object to showcase the application's precision

and accuracy, while the other focuses on quick coverage of an area with reconstruction detailed enough for various Geographic Information System (GIS) tasks, emphasizing efficiency over fidelity. These experiments aim to illustrate the diverse capabilities of the CR Fly application, making it suitable for a wide range of practical applications from detailed cultural heritage documentation to efficient large-scale mapping projects.

4.2 Experiment 1: Monument of Nicolaus Copernicus

The Monument of Nicolaus Copernicus, created by academic sculptor Tibor Bartfay in 1974 [6], was selected for this experiment due to its intricate design and slightly greater height, which make it an ideal subject for testing the application's ability to capture detailed features. Its historical significance and unique architectural elements further enhance the value of this experiment, ensuring a comprehensive assessment of the application's 3D modeling capabilities.

4.2.1 Procedure

The goal of this experiment was to produce a highly detailed 3D model of the monument, accurately capturing its geometric features and surface textures. By meticulously planning the flight and image capture process, we aimed to minimize errors and optimize the quality of the final model.

Flight plan

We conducted a manual circular flight around the monument, starting at an initial altitude of 0.5 meters. The altitude was incrementally increased by 0.5 meters, reaching a maximum height of 3 meters, to capture images at various layers. The entire flight and image capture process was completed within 15 minutes.

Image capture

A total of 115 images were captured at distances of 1 to 1.5 meters from the object, ensuring approximately 80% overlap for accurate reconstruction. The upload of the images from the UAV to RealityCapture was completed in 4 minutes.

Reconstruction

The mesh representing the point cloud contained 114,370 points, with the reconstruction process taking 40 seconds. Creating the colorized texture model with normal quality, which comprised 4.4 million triangles, took 6 minutes. Producing the high-quality colorized texture model, consisting of 17.5 million triangles, took 12 minutes.

4.2.2 Results and analysis

The 3D model of the monument was highly detailed, with an accurate representation of geometric features and surface textures. Some minor artifacts were observed in areas with significant shadowing, which could be corrected by adding additional photographs taken from closer distances. Despite these minor issues, the model successfully highlighted the monument's intricate details, demonstrating the effectiveness of our approach in capturing complex structures.

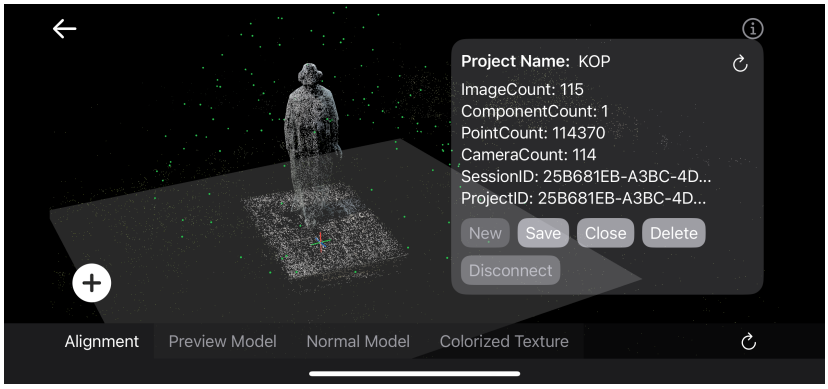

Figure 4.1: Scene analysis of the Monument of Nicolaus Copernicus

4.3 Experiment 2: Church of St. Elizabeth

The Church of St. Elizabeth, also known as the Blue Church, located in Bratislava, was designed by the Budapest architect Ödön Lechner and constructed at the beginning of the twentieth century [5].

It was chosen for its intricate design and substantial size, which present unique challenges for reconstruction. This experiment aims to showcase the effectiveness of the CR Fly application in capturing detailed architectural features and the overall structure using remote sensing techniques, which can be more challenging than traditional ground-based methods. Additionally, the experiment seeks to demonstrate the application's versatility and reliability in various architectural settings.

4.3.1 Procedure

The objective was to produce a highly detailed 3D model of the Blue Church, accurately representing its complex architectural features and overall structure. Given the building's complexity and extensive size, the task posed significant challenges in capturing all intricate details and ensuring comprehensive coverage, which further underscores the capabilities of our application.

Flight plan

We conducted a manual circular flight around the building, capturing images at distances ranging from 2.5 to 5 meters. The altitudes varied from 4 meters to 40 meters, increasing in increments of 4 meters, to ensure images were captured at various layers. This method ensured comprehensive coverage, allowing for the capture of both detailed architectural elements and the overall structure.

Image capture

A total of 526 images were captured, ensuring approximately 70-80% overlap for accurate reconstruction. The upload of the images from the UAV to RealityCapture was completed in 20 minutes.

Reconstruction

The mesh representing the point cloud contained 1,180,966 points, with the alignment process taking 5 minutes. Creating the colorized texture model with normal quality, comprising 48 million triangles, took 36 minutes. Producing the high-quality colorized texture model, consisting of 169.5 million triangles, took 118 minutes.

4.3.2 Results and analysis

The 3D model of the Blue Church was highly detailed, with an accurate representation of its intricate architectural features and structural elements. Some errors were observed due to insufficient photographic coverage, which can be corrected by adding additional photographs taken under similar external conditions. Nevertheless, the model effectively highlighted the church's complex design, underscoring the potential of our application for detailed architectural reconstruction.

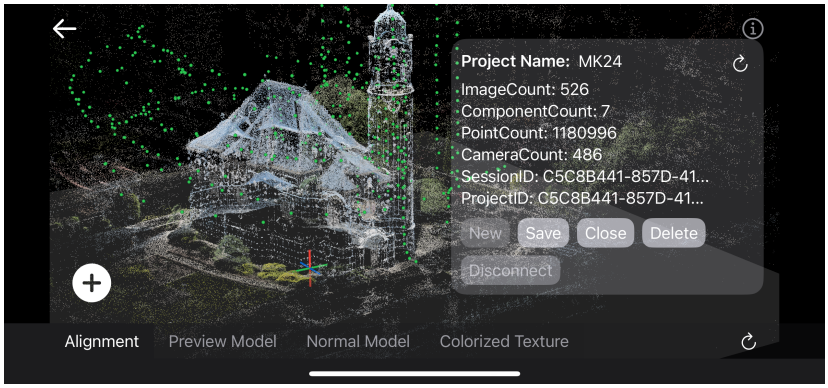

Figure 4.2: Scene analysis of the Blue Church

4.4 Experiment 3: Part of Viničné village

This experiment was conducted to demonstrate the capability of our software to map areas and perform tasks associated with geodetic surveying, which is directly supported by RealityCapture. By focusing on a real-world environment with various structures and natural features, we aimed to validate the software's practical applications and effectiveness in diverse surveying scenarios.

4.4.1 Procedure

The objective was to create a 3D model of a part of Viničné village, accurately representing the area's geographical and structural features, with enough detail to ensure individual buildings are recognizable.

Flight plan

We conducted a manual flight at an altitude of 100 meters, following a grid pattern to ensure comprehensive coverage of the designated area. The flight path was meticulously planned to cover the entire reconstruction area and provide a detailed representation of the surroundings.

Image capture

A total of 139 images were captured, ensuring approximately 70% overlap for accurate reconstruction. The upload of the images from the UAV to RealityCapture was completed in 5 minutes.

Reconstruction

The mesh representing the point cloud contained 321,734 points, with the alignment process taking 2 minutes. Creating the colorized texture model with normal quality, comprising 17.7 million triangles, took 10 minutes. Producing the high-quality colorized texture model, consisting of 69.6 million triangles, took 27 minutes.

4.4.2 Results and analysis

The 3D model of the Viničné village area was sufficiently detailed to accurately represent its geographical and structural features, allowing for the recognition of individual buildings and streets. The model successfully demonstrated the potential of our application for detailed area mapping and geodetic surveying, with applications in city planning, highlighting its versatility and reliability in various surveying contexts.

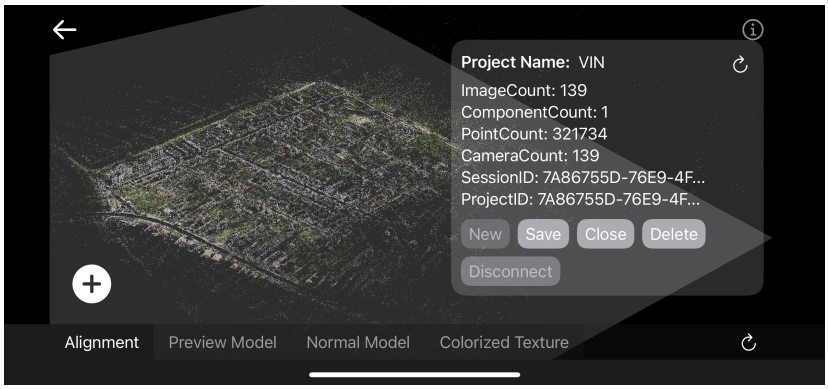

Figure 4.3: Scene analysis of the part of Viničné village

4.5 Summary of results

The experiments demonstrated that the number of images significantly impacts the final quality of the 3D model, particularly when aiming to achieve highly accurate wall structures or precise object shapes. As the number of images provided for the reconstruction process increases, so does the number of triangles that constitute the 3D model, which consequently leads to an increase in the model's detail and file size.

Through experimentation, we found that our computer struggled to display models with over 5 million triangles in common applications designed for viewing 3D models, such as SketchUp, Microsoft 3D Paint, and Microsoft 3D Viewer. This finding underscores a current limitation in our ability to create models that can be optimally viewed in these applications.

Additionally, based on the conducted experiments, we can estimate that scene analysis consisting of a point cloud, hundreds of images require approximately tens of minutes, with the majority of this time being spent on the transfer process. The duration of the alignment itself is thus negligible. Moreover, it is evident that the application's ability to transfer images during flight utilizes time to minimize the user's waiting period for alignment.

Consequently, we conclude that while we are capable of creating highly detailed models, there is a need to manage the complexity and file size of these models to enhance their usability across various platforms. As future work, we higly recommend implementing model rendering using level of detail in our application. This approach will ensure efficient rendering and viewing of models by dynamically adjusting the level of detail based on the viewing context, thereby improving the overall performance and usability of the generated models.

The previewable 3D model files and the physical model created in this part of the work are included in Appendix C.

Conclusion

The primary objective of this bachelor thesis was to develop a mobile application, CR Fly, to facilitate the 3D reconstruction of scenes using images captured by DJI drones. The application aimed to streamline the process of image capture, transfer, and reconstruction, ultimately enhancing the efficiency and quality of 3D model generation by leveraging the DJI SDK as the drone controlling API and RealityCapture software as a 3D reconstruction API. Additionally, the application simplified the scanning process by providing real-time feedback.

The work comprehensively reviews existing methodologies and technologies in 3D reconstruction and UAV-based photogrammetry. We identified key limitations in current software solutions, particularly in terms of autonomous flight capabilities and the seamless integration of data transfer and processing workflows. These insights guided the specification and development of the CR Fly application, which addresses these gaps through several innovative features.

Our implementation involved the use of Swift programming language and the MVC architecture to ensure a modular and scalable design. The application supports real-time image processing, intuitive UAV control, and efficient communication with RealityCapture over a network. Key components, such as the *CommandQueueController*, were introduced to manage command execution and handle potential errors robustly.

Extensive testing, both manual and automated, was conducted to validate the functionality and performance of the application. The experiments demonstrated the application's capability to produce high-quality 3D models under various conditions, highlighting its adaptability and effectiveness.

The results of our experiments, including the reconstruction of the Monument of Nicolaus Copernicus, the Church of St. Elizabeth of Hungary, and part of the Viničné village, confirmed that the application is easy to use and has the potential to deliver detailed and accurate 3D models. These findings underscore the application's utility in fields such as cultural heritage preservation, urban planning, and geodetic measurements.

In conclusion, the CR Fly application represents a significant advancement in UAV-based 3D reconstruction. By addressing the limitations of existing solutions and providing a comprehensive tool for image capture and processing in real-time, this work

lays the foundation for future enhancements, such as fully autonomous flights and real-time reconstruction. The application's modular design and robust architecture ensure its scalability and adaptability, positioning it well for future technological developments in the field. Further research and development could focus on optimizing autonomous flight algorithms and expanding the application's compatibility with a wider range of UAV models and reconstruction software.

References

[1] Apple Inc. HeaderDoc User Guide. https://developer.apple.com/library/archive/documentation/DeveloperTools/Conceptual/HeaderDoc/intro/intro.html.

[2] Apple Inc. Swift Documentation. https://www.swift.org/documentation.

[3] DroneDeploy Inc. DroneDeploy Aerial and Help Center. Available at: https://dronedeploy.com.

[4] Epic Games, Inc. RealityCapture - 3D Models from Photos and/or Laser Scans. Available at: https://www.capturingreality.com.

[5] J. Hal'ko and J. Sedlák. *Modrý kostol: dejiny Kostola sv. Alžbety v Bratislave*. LÚČ, Bratislava, 2006.

[6] J. Hamšíková. Pamätník Mikuláša Koperníka. https://muop.bratislava.sk/vismo/dokumenty2.asp?id_org=600176&id=4158, August 2012.

[7] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2nd edition, 2003.

[8] OpenAI Inc. ChatGPT 4. https://www.openai.com, 2023.

[9] Pix4D S.A. Professional photogrammetry and drone mapping software - Pix4Dcapture. Available at: https://www.pix4d.com.

[10] T. Schenk. Introduction to photogrammetry. *The Ohio State University, Columbus*, 106(1), 2005.

[11] SZ DJI Technology Co., Ltd. Ground Sample Distance. Available at: https://enterprise-insights.dji.com/blog/ground-sample-distance.

[12] SZ DJI Technology Co., Ltd. Mobile SDK V4 - Documentation and API Reference. Available at: https://developer.dji.com/mobile-sdk-v4.

[13] Linglong Zhou, Guoxin Wu, Yunbo Zuo, Xuanyu Chen, and Hongle Hu. A comprehensive review of vision-based 3d reconstruction methods. *Sensors*, 24(7), 2024.

All online references cited in this thesis were accessed on May 24, 2024.

Appendix A: Software development documentation

This appendix includes the developer's documentation of the software, which contains detailed comments on individual classes and methods. The documentation is provided as a single `.doccarchive` file in the electronic attachment.

Appendix B: Source code of the application

This appendix contains the complete source code of the application. The source code files and test files are included as electronic attachments.

The latest updates and revisions of the application are available on GitHub at: `https://github.com/lbujnak/CR-Fly`.

Appendix C: Reconstructed objects and physical models

This appendix presents the reconstructed models of objects discussed in Chapter 4. All relevant previewable files are included as electronic attachments.

Additionally, physical models of the experiments discussed in Section 4.2 and Section 4.3 are included as physical attachments.