BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

MOBILE APP FOR CAPTURING AND VIEWING PHOTOGRAPHS OF THE SAME OBJECT AT DIFFERENT TIMES

MOBILNÍ APLIKACE PRO POŘIZOVÁNÍ A PROHLÍŽENÍ FOTOGRAFIÍ STEJNÉHO OBJEKTU V RŮZNÝCH ČASECH

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. DOMINIK PLŠEK**
AUTOR PRÁCE

SUPERVISOR **prof. Ing. ADAM HEROUT, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Plšek Dominik, Bc.**
Program: Informační technologie Obor: Počítačová grafika a multimédia
Název: **Mobilní aplikace pro pořizování a prohlížení fotografií stejného objektu v různých časech**
Mobile App For Capturing and Viewing Photographs of the Same Object at Different Times
Kategorie: Zpracování obrazu

Zadání:

1. Seznamte se s problematikou vývoje pro iOS. Zaměřte se na pořizování a prohlížení fotografií.
2. Vyhledejte a analyzujte existující aplikace pro pořizování, prohlížení a porovnávání fotografií stejného objektu v různé časy.
3. Prototypujte dílčí prvky uživatelského rozhraní pro pořizování fotografií téhož objektu v různé časy. Testujte prototypy na uživatelích a iterativně je vylepšujte.
4. Navrhněte aplikaci pro pořizování fotografií téhož objektu v různé časy a pro prohlížení těchto fotografií.
5. Implementujte navrženou aplikaci, testujte ji na uživatelích a iterativně ji vylepšujte.
6. Demonstrujte funkčnost vytvořené aplikace na vhodných datech sbíraných po delší časový úsek.
7. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN: 978-0321965516
- Steve Krug: Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability, ISBN: 978-0321657299
- Matthew Mathias, John Gallagher: Swift Programming: The Big Nerd Ranch Guide (2nd Edition), Big Nerd Ranch Guides
- Soonmin Bae, Aseem Agarwala, Frédo Durand: Computational Rephotography, ACM Transactions on Graphics (TOG), Volume 29, Issue 3, June 2010

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2, značné rozpracování bodů 3 až 6.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 7. listopadu 2018

Abstract

Rephotography has been a popular research topic in the photography field for a long time. The purpose of rephotography itself is to repeatedly take photographs of the same scene at a different time. As a result, the sequence of rephotographs with the reference, often historical, the picture provides a compelling visualization of the evolution of the subject or capture its changes in time. However, the act of rephotography is difficult for the rephotographers as they have to cope with the ambiguous motions in six degrees of freedom and with the changes of the subject itself or its surrounding environment.

This thesis aims to create a mobile application that would help its users to capture a rephotograph more accurately and allow them to share the scenes amongst other users. The designed application uses available on-device sensors to navigate the user to the location and guide the user during the rephotography process to capture a precise rephotograph. Furthermore, the application contains user interface elements designed explicitly for rephotography. Moreover, the work describes topics about user interface design, iOS application development, and designing and deploying backend API for the mobile application.

Abstrakt

Refotografie je již dlouhou dobu zkoumaným tématem v oboru fotografie. Cílem refotografie samotné je opakovaně pořizovat fotografie stejné scény v různých časech. Výsledkem je kolekce fotografií s referenční, mnohdy historickou fotografií zachycující vývoj a případně změny focené scény. Nicméně refotografie samotná je pro fotografy nesnadným úkonem. Fotograf se musí vyrovnat s pohyby v šesti stupních volnosti a případnými změnami předmětu fotografie nebo změnou jeho okolního prostředí. V poslední době se výzkum zabývá výpočetní refotografií využívající výpočetní postupy a algoritmy z oboru počítačového vidění, se záměrem omezit překážky při pořizování opakované fotografie.

Cílem diplomové práce je vytvořit mobilní aplikaci sloužící uživatelům k zachycení preciznější refotografie a umožnit její sdílení s ostatními uživateli aplikace. Navržená aplikace využívá dostupných senzorů na zařízení k navigaci uživatele na odpovídající lokaci. Současně uživatele provází během procesu k vyfocení přesnější refotografie. Uživatelské rozhraní aplikace je specificky navrženo pro refotografii. Práce obsahuje témata návrhu uživatelského rozhraní, vývoje mobilních aplikací pro operační systém iOS a návrh a produkční nasazení serverové části služby a API pro mobilní aplikaci.

Keywords

rephotography, mobile applications, user interfaces, iOS, Ruby on Rails, photo and video applications

Klíčová slova

refotografie, mobilní aplikace, uživatelská rozhraní, iOS, Ruby on Rails, foto a video aplikace

Reference

PLŠEK, Dominik. *Mobile App For Capturing and Viewing Photographs of the Same Object at Different Times*. Brno, 2019. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor prof. Ing. Adam Herout, Ph.D.

Rozšířený abstrakt

Refotografie je již dlouhou dobu zkoumaným tématem v oboru fotografie. Cílem refotografie samotné je opakovaně pořizovat fotografie stejné scény v různých časech. Výsledkem je kolekce fotografií s referenční, mnohdy historickou fotografií zachycující vývoj a případně změny focené scény. Nicméně refotografie samotná je pro fotografy nesnadným úkonem. Fotograf se musí vyrovnat s pohyby v šesti stupních volnosti a případnými změnami předmětu fotografie nebo změnou jeho okolního prostředí. V poslední době se výzkum zabývá výpočetní refotografií využívající výpočetní postupy a algoritmy z oboru počítačového vidění, se záměrem omezit překážky při pořizování opakované fotografie.

Cílem diplomové práce je vytvořit mobilní aplikaci sloužící uživatelům k zachycení preciznější refotografie a umožnit její sdílení s ostatními uživateli aplikace. Navržená aplikace využívá dostupných senzorů na zařízení k navigaci uživatele na odpovídající lokaci. Současně uživatele provází během procesu k vyfocení přesnější refotografie. Uživatelské rozhraní aplikace je specificky navrženo pro refotografii. Práce obsahuje témata návrhu uživatelského rozhraní, vývoje mobilních aplikací pro operační systém iOS, návrh a produkční nasazení serverové části služby a API pro mobilní aplikaci.

Mobilní aplikace přistupuje k problematice výpočetní refotografie odlišným způsobem než články popisované v textu. Namísto aplikace algoritmů počítačového vidění, mobilní aplikace využívá dostupné senzory na mobilním zařízení. Na základě dat z akcelerometru, gyroskopu a magnetometru aplikace uloží rotaci zařízení při focení scény. Následně při refotografii je na základě uložených dat (kvaternionů) vypočítávána relativní rotace aktuálního natočení zařízení vůči referenčnímu. Tato informace je prezentována ve formě navigačního obdélníku, který navádí uživatele ke správnému natočení zařízení. V případě, že je navigační obdélník zarovnán ve středu mřížky, rotace zařízení odpovídá referenčnímu natočení. Při focení nové scény reprezentuje navigační obdélník rotaci okolo osy z a vede uživatele k zarovnání zařízení s mřížkou. Data ze senzorů jsou získávána pomocí frameworku *Core Motion*.

Uživatelské rozhraní aplikace je optimalizováno pro iOS zařízení s důrazem na uživatelskou přívětivost a jednoduchost. Některé ovládací prvky jsou navrženy s ohledem na průzkum jiných mobilních aplikací a projektů se zaměřením na fotografii a refotografii, které jsou diskutovány v textu práce. Jednotlivé prvky uživatelského rozhraní jsou navrženy tak, aby pomohli uživatelům v procesu refotografie zachytit přesnější refotografii dané scény, případně vizualizovali důležité informace o scéně. Příkladem je funkce pohledu nohou, která napomáhá při vyhledávání přesného místa focení původní scény. Překryv původní fotografie s pohledem z kamery slouží k preciznímu zarovnání při vytváření refotografie. Po vyfocení refotografie aplikace umožňuje porovnat rozdíl mezi právě pořízenou refotografií a původní fotografií. Uživatel může procházet existující scény ve dvou režimech zobrazení – galerii scén a anotované mapě. Obrazovka s dodatečnými informacemi o scéně obsahuje, kromě referenční scény a data pořízení, i adresu získanou reverzním geokódováním a galerii refotografií dané scény. Refotografie scény lze zobrazit v módu časové osy.

Při vytváření nové scény lze na mapě přesně definovat lokaci, odkud byla fotografie pořízena. K zobrazení mapových podkladů slouží framework *Map Kit*. Ke sběru dat o poloze uživatele je použita technologie *Core Location*. Aplikace k přístupu a konfiguraci kamery na mobilním zařízení a programování funkcí fotoaparátu využívá *AVFoundation*. Síťová komunikace mobilní aplikace se serverem s aplikačním rozhraním je implementována pomocí *URLSession* z frameworku *Foundation*. Persistenci uživatelských dat obstarává technologie *Core Data*.

Serverová část a aplikační rozhraní je implementováno prostřednictvím frameworku *Ruby on Rails*. Aplikační rozhraní je postaveno na architektuře *REST* a data jsou posílána ve formátu *JSON*. Ke snížení objemu zasílaných dat jsou fotografie přenášeny jako binární data a k tomuto účelu mobilní aplikace sestavuje odpovídající URL dotazy. V produkčním prostředí je použita databáze *PostgreSQL*. Multimediální data jsou ukládána v cloudovém úložišti *Amazon S3 Bucket*. Webová aplikace s databází je hostována na službě *Heroku*.

Výsledky testování potvrzují, že navržená mobilní aplikace pomáhá uživatelům zachytit přesnější refotografie, které mohou být použity ke sledování změn dané scény v různých časech. Aplikace byla v průběhu testování uživateli dobře přijata a její uživatelské rozhraní bylo hodnoceno jako jednoduché a uživatelsky přívětivé. V průběhu testování uživatelé nevyužívali všechny funkce aplikace. Zpětná vazba poskytuje informace o funkcích vyhodnocených uživateli jako účelných. Naopak poukazuje na potřebu případných změn v uživatelském rozhraní pro získání většího komfortu při ovládání aplikace. Jedná se například o přidání tlačítka k zobrazení záběru nohou přímo do obrazovky kamery, pokud je aplikace v režimu refotografie.

Výsledná aplikace poskytuje základ pro platformu využitelnou k refotografii ve větším měřítku. K dosažení daného cíle je třeba mobilní aplikaci zaměřit na focení venkovních scén, kde navržený a implementovaný navigační systém, na základě poznatků z testování, funguje nejlépe.

Mobile App For Capturing and Viewing Photographs of the Same Object at Different Times

Declaration

Hereby I declare that this thesis was prepared as an original author's work under the supervision of prof. Ing. Adam Herout, Ph.D. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
Dominik Plšek
May 21, 2019

Acknowledgements

I would like to express my gratitude to my supervisor prof. Ing. Adam Herout, Ph.D. for his guidance and continuous support.

Contents

Chapter 1

Introduction

Rephotography has been a popular research problem in the photography field for a long time. The rephotography aims to take a photograph from the same vantage point as the reference (historical) photograph [60]. However, the task of taking an accurate rephotograph is challenging and often imprecise. The rephotographer must identify correct motions in six degrees of freedom, including 3D translation and 3D rotation [45] [69]. Furthermore, the subject may look different from new photographs due to structural changes, weather, and other elements [45].

Nevertheless, when the modern rephotograph accurately aligns with the historical photograph, rephotography can provide a visualization of the progression of the photographed subject [45]. Rephotography is, in many cases, used to study history. Several rephotographers and rephotography projects capture changes in major world cities and landscapes [45]. However, rephotography is also used to study changes and the impact of external factors in the environment [45].

This thesis focuses on developing a mobile application applicable to rephotography. The application allows the users to capture new scenes, share scenes amongst users of the application, view the timeline that illustrates the changes of the scene at different times, and guide the users when rephotographing an existing scene. Furthermore, the application includes user interface elements explicitly devised for the task of rephotography.

The work covers topics about the design and implementation of the application. Individual chapters describe the iOS operating system, technologies and framework for persisting data, collecting motion data from device's sensors, determining the geographical location of the user, and showing the scene locations on a map. Furthermore, the text discusses programming server-side application and designing the API for communication between the mobile application and the server. Finally, the thesis presents results of the user interface evaluation and accuracy analysis of rephotographed photos.

Chapter 2

Rephotography and Capturing and Viewing Photographs

2.1 Rephotography

Rephotography, also known as repeat photography, is the action of repeatedly taking photographs of the same scene at a different time. The delay between the two images is commonly referred to as a then and now view of the scene. The overall precision and quality of the rephotographed photo depend on photographers skill and their selection of camera and lenses. Furthermore, to capture an accurate rephotograph, the photographer has to figure out various details [48].

First of all, the photographer has to decide which photo they want to rephotograph. Rephotographers generally capture a building or other significant object in the photograph which is present in the current view to show the continuity between the two photographs [45]. Additionally, in order to create a precise rephotograph, the photographer has to reproduce a vantage point of the original photograph. Reproducing the original viewpoint can be challenging because historical images can look different from new photographs due to architectural changes, weather, daytime, and other factors [45]. Furthermore, as modern digital cameras and lenses differ from older equipment, the photographer has to consider the fact that the historical photographs are captured by a camera of unknown calibration [45]. If the rephotograph of the original image is accurate, cross-fading or placing the original and rephotographed image next to each other reveals which scene elements changed and which have stayed the same [45].

2.1.1 Computational Rephotography

Computational rephotography is an approach of using a computer or smartphone with a camera to assist humans with the rephotography process. Rephotography without computational assistance relies on human's subjective judgment by eye, which is very challenging as the user has to deal with motions in 6 degrees of freedom. As a result, recaptured photographs are very inaccurate or take considerable time and skill to capture precisely [69].

Reliable rephotography can be done using a robot platform mounted with a camera. The platform can dynamically adjust the pose of the camera. Therefore, the scene can be consistently rephotographed with a high degree of accuracy. However, due to the size and cost of such platform, this solution is not easily transportable nor accessible to the majority of photographers [45].

To overcome disadvantages of the human performed rephotography and the robotic platform with a mounted camera, researchers focused on more accessible technology such as computers with connected camera or smartphones to make the process of capturing a precise photograph of the same scene easier. Notable papers in this field are discussed in section 2.4.

2.1.2 Applied Rephotography

Comparison of an original and rephotographed image provides a then and now view of the scene. Rephotography can show the progress or decline of the subject and be used as a tool to study history and changes of the subject in some circumstances [45]. Examples of rephotography used as a tool to study history include *Second View*, *New York Changing*, *Then and Now* and *Time After Time*. *Second View* is a rephotographic survey of landscape images of the American West [45]. *New York Changing (2004)* is a rephotograph project of photographer Douglas Levere who rephotographed photographs from Berenice Abbot's *Changing New York* (1939) [61]. *Then and Now* is a series of 40 books each containing rephotographs of a major world city [45]. Lastly, *Time After Time* is a rephotography project from Mark Hersch in which the photographer merges historic images with photographs from the present into a single image [33].

In addition to history, rephotography can be used to document environmental changes such as glacier melting or geological erosion [45]. Furthermore, rephotography is used in science, for example, to capture vegetation phenology [64]. Moreover, rephotography can be applied to monitor personal, health, or cultural changes [45].

rePhoto

rePhoto is an application accompanied by a website designed for repeat photography. The application shows a transparent overlay of the previous photograph on the camera screen, to help the user to capture accurately aligned rephotograph. The user can see all nearby sites for rephotography on a map. When the user selects a specific location, they can rephotograph the rephotography subject. As mentioned earlier, the application only shows the transparent overlay of the previous photograph. After the user finishes the rephotography process, the application uploads the image with all relevant metadata to a database [27].

The rePhoto project consists of mobile applications for iOS and Android and a website. Visitors of the website can browse through the rephotography projects using the rePhoto applications. The website also allows users to create a new project or manage an existing one [27]. At the time of writing, the application for iOS is not available.

WhatWasThere

WhatWasThere project displays historic photographs which can be blended in with photographs from Google Street View. The website supports browsing, uploading and managing photographs and suits as an online archive which reflects the history of the location. The visitor of the website can also switch between photos from the same location and see the differences [26]. The user interface of WhatWasThere's website can be seen in Figure 2.1.

The website on itself does not support capturing new photographs, but the project has a companion application for mobile devices with the iOS operating system. The iPhone and iPod Touch application should show historic photographs on a map based on the current user location. Furthermore, the application offers an augmented reality experience using

Figure 2.1: Screenshot from WhatWasThere website. The screenshot shows the historic photograph placed in the location where it was taken but fitted in current time. WhatWasThere makes use of Google Street View technology to show the current context in which the historic photographs are placed manually by users.

the device embedded camera and offer a historical experience of locations surrounding the user. Moreover, the application also uses the camera to obtain the current state of the location and offers a *screen fader* feature to transition between past and present on locations containing historic photographs in WhatWasThere database [26]. However, none of the features nor the application could be tested as it is not available on the App Store in the region of the author. Additionally, even after switching to the US App Store, the application was advertised as not available on the store.

Historypin

HistoryPin collects local history through contributions from the community. In comparison with the other projects, HistoryPin does not lead its users to rephotograph locations in the archive. The project aims to collect diverse and broad historic content and group related subjects into collections which users can explore [25].

At the time of writing, HistoryPin does not offer a mobile application. However, the project's website states that a new application with a redesigned interface is being developed.

Retake Melbourne

Retake Melbourne is a project of Deakin University from Victoria, Australia. The project was successfully crowd-funded at a Pozible platform. The goal of the initiative is to rephotograph more than a fifty-year-old archive of photographies of Melbourne. Money raised from the crowd-funders is planned to use to develop a mobile application which should help volunteers with the rephotography process. Through the power of community effort, the project wants to retake about 5000 original photographs of Melbourne [62].

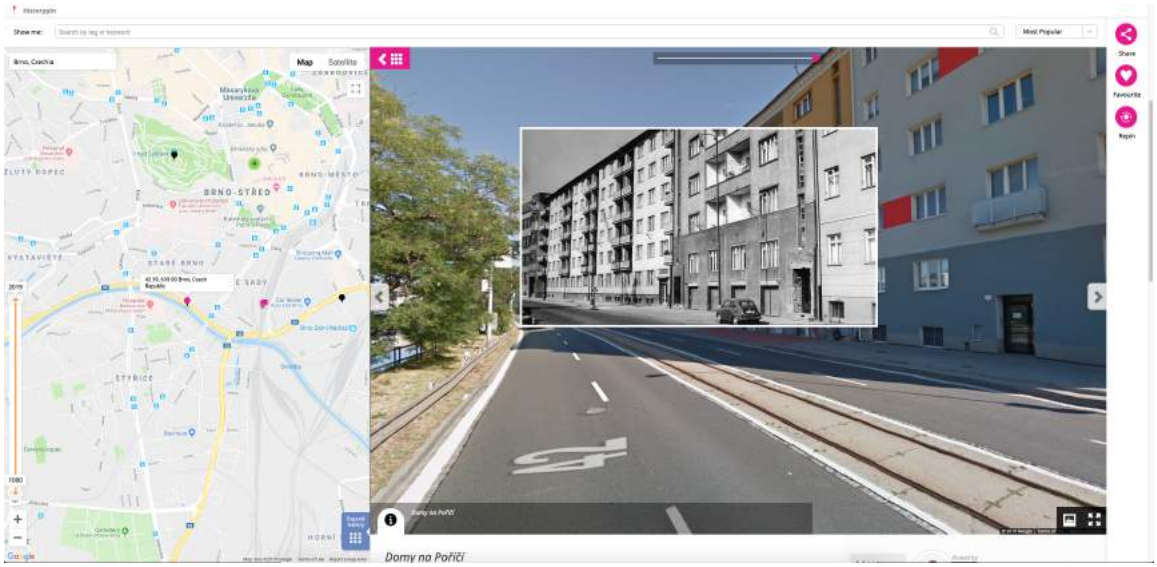

Figure 2.2: HistoryPin project is very similar to WhatWasThere. The website offers similar features, but HistoryPin's core functionality is collections of content (photos, sounds, videos) with a similar theme. The Figure shows an overlay of a historical photograph placed in the Google Street View.

Retake Melbourne was successfully funded on 18th of June, 2013 and raised 6417 AUD [62]. An update from 10.12.2014 posted on the project's funding page states that the mobile application development is in progress. However, a comment posted on 12th of March, 2015 from one of the project supporters reveal that the application is still non-existent. As of 4th of January, 2019, the application is not published on App Store, and the project's page does not contain additional information about the state and the progress of the application nor the overall rephotography project.

TimeLens

TimeLens is a mobile application for creating time-lapse videos. The application allows adding new sites to photograph or submitting new photographs to already created one. After the site is created and photographs of the site are submitted by users, the application generates a time-lapse video from photographs collected by one or multiple users [66].

In the application, users can browse already created sites on a map. When a detail of a selected site is opened, the user can play the created time-lapse video, see a comparison of two site's photographs and star the site which adds the site to the user starred list. Furthermore, when the user taps on the info button, the application displays location and additional information about the site [66]. Screenshots of the discussed screens can be seen in Figure 2.3.

The application helps users to find already photographed locations or create new locations to photograph. It provides useful information about the location and helps other users to reach the location. However, while taking a new photograph of the same location, the user is not provided with any computational navigation, and thus, the photograph is captured based on the eye judgment from the user. Nevertheless, the user can use the comparison feature to compare their photograph with an already captured photograph. The

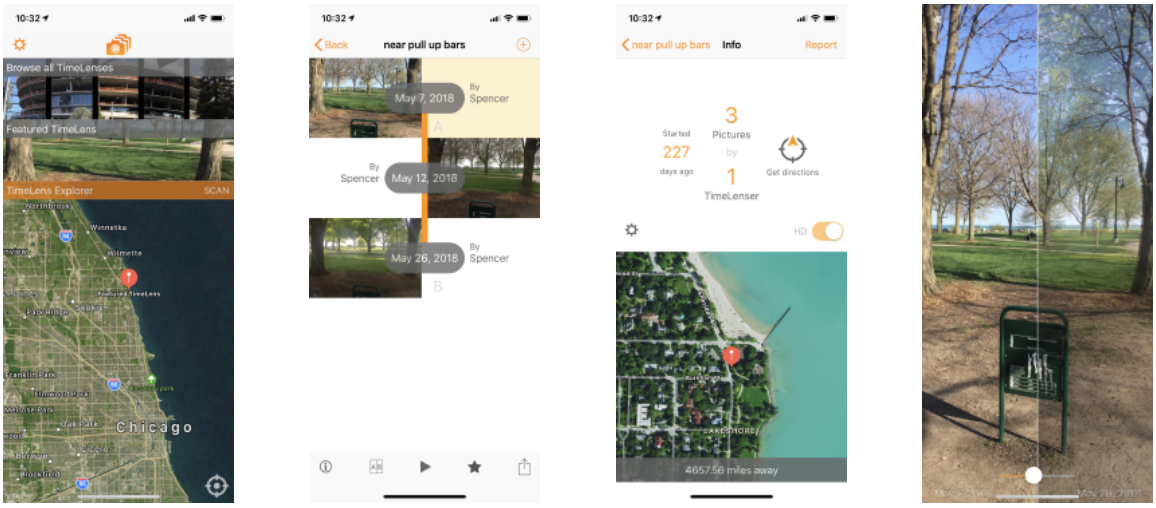

Figure 2.3: Screenshots from TimeLens application. The first screenshot shows a map with pins representing locations already photographed from the users. The second screenshot displays detail of the location. In the third screenshot, location and additional information about the location can be found. The last screenshot shows the comparison between the two photographs.

ability to collaboratively photograph different locations allows anyone to contribute and add to the variety and length of the created time-lens video.

2.2 Camera applications

2.2.1 Camera (iOS)

Camera is default photography and video application on iOS devices. It offers a convenient way to capture high-quality photos and videos. Moreover, the application has a clean and straightforward user interface and essential configurability. Furthermore, the application supports multiple capture modes – time-lapse, slow-motion, video, photo, portrait, square, and pano [72].

In the time-lapse mode, Camera application captures multiple photos of the scene over some time and then combines the photos into a sped-up video [53]. Slow-motion mode captures video at a faster frame rate. Afterward, when the video is played at a standard frame rate, time seems to be slowed down [54]. Video mode allows capturing of video footage with a selected frame rate and video resolution.

When the Camera is in photo mode, the user can capture live photos. The *live photo* is 1.5 seconds long video with audio before and after the user pressed the shutter button to capture a photo. After capturing a live photo, a video with a total duration of 3 seconds is saved to the user's photo gallery while one frame is selected as a title photo [73]. Photo and square mode allows the user to hold down on a shutter button to activate burst mode in which the camera takes ten photos every second [75]. In photo, portrait and square mode the camera allows to capture HDR photos, use a self-timer, and apply filters with a live and thumbnail preview located under the camera view-finder. Portrait mode detects a subject and a background and then blurs the background [51]. Furthermore, Camera

Figure 2.4: Screenshots of Camera (iOS) application from Best Camera App For iPhone: Compare The 4 Best Camera Apps [72]. The first screenshot shows manual controls over focus and exposure in the application. The second screenshot displays Camera's Portrait mode with the option to choose from several light effects.

supports multiple light effects in portrait mode. Lastly, the pano mode allows capturing of panoramic photographs.

During the video and photo capture, the user can manually set focus and adjust exposure. Additionally, the user can customize the camera view-finder to display an adaptive grid with levels. Finally, on dual camera devices Camera can zoom in using a telephoto lens [72].

2.2.2 Halide

Halide is a camera application for iOS devices with a gesture-based interface. The user interface of the application is designed specifically for iPhone 8 and iPhone X devices. On iPhone X and newer devices, the application is usable with one hand due to the custom-designed user interface [3]. Halide provides an intelligent automatic mode as a default option and a manual mode with control over ISO sensitivity, shutter speed, and white balance. Additionally, the application supports depth capture and has a dedicated portrait mode. Furthermore, features include live histogram for checking exposure, adaptive level grid, and focus peaking. Controls available in *Quick Bar* are customizable, and the application supports location data removal before sharing an image to social networks or messaging applications.

Images of the application in Figure 2.5 shows the user interface and selection of features from the application. The first screenshot from the left shows Halide's application view-finder. The screen contains an adaptive grid, a live histogram (located in the top left corner of the device) and exposure value in the right top corner. User customizable *Quick Bar* is located directly underneath the camera view-finder. There are two toggles located under the quick bar – autofocus or manual focus and depth capture toggle. The bottom of the screen contains a preview of the last picture, the shutter button, and a zoom toggle. The image thumbnail also acts as an entry point to the photo gallery. The zoom and depth toggle is available only on devices with a dual camera. The second screenshot of the application shows the focus peaking feature. Focus peaking shows highlighted edges of objects in focus [71]. In Halide detected edges are highlighted in green color.

Figure 2.5: Halide application screenshots from Halide's Press Kit [3]. The leftmost picture shows a screen of Halide's user interface. The camera view-finder contains the adaptive level grid. In the top left corner, a live histogram indicates the current exposure. In the right corner, the application shows the current exposure value. The second screenshot from the applications displays the focus peaking feature. The last screenshot shows the portrait view while taking a photo from the front camera.

In comparison with the first screenshot, the second screenshot reveals that the application is switched into manual mode and ISO sensitivity, white balance, and shutter speed is configurable. ISO and white balance are configured by tapping on corresponding buttons while shutter speed can be changed by scrolling up and down or left to right on the camera view-finder. The last screenshot displays a portrait mode with a depth capture.

Halide supports three different depth visualizations in the view-finder. The first one is *depth peaking*, Halide's unique depth visualization, which is similar to the focus peaking. Depth peaking overlays detailed depth information while still allowing the user operating the application to view the subject. The depth peaking visualization is the default option in Halide. The second visualization is *portrait preview*, which displays the effects before capturing the photo and can be seen in the last screenshot in Figure 2.5. The third visualization is called *depth view*. Depth view shows the depth map. In this visualization, the user loses details, but it can help to see if there is enough information to determine foreground and background while capturing a photo with depth [68].

In addition to the three depth visualizations in the view-finder, the application allows to view the depth information in augmented reality through *Augmented Reality Viewer* feature after the image is captured. Halide's AR Viewer projects depth capture into augmented reality and lets users explore the photo in three dimensions. AR Viewer is built on top of Apple's ARKit framework to display content in augmented reality [68].

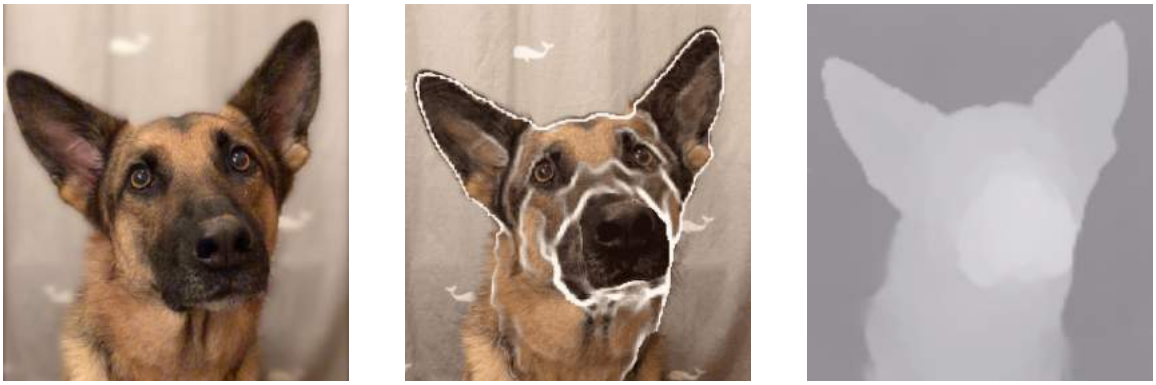

Figure 2.6: Halide various depth visualizations. Images taken from Halide's blog article *Halide 1.7: In Depth* [68]. The first picture shows the portrait view. Portrait view shows the depth effect before capturing a photo. The second image displays depth peaking which overlays detailed depth information while still allowing to see the subject while capturing a photo. The third image shows a depth map showing which pixels are located in foreground and background. In the last visualization, the user loses details about the subject, but the depth map helps to identify foreground and background [68].

2.2.3 ProCamera.

ProCamera. is iOS mobile photography and video application with full control over the camera. The application is highly customizable, and users can select features available in the camera view-finder. The user interface provides seamless control and access to the application's functions. The application provides manual control over the shutter speed, ISO sensitivity, and white balance. Also, the user can optionally set exposure and focus separately. Furthermore, the application provides modes for HDR photography, video recording in HD and 4K resolutions and special modes for lowlight photography. Additionally, the application supports *Portrait mode* and has a dedicated *Selfie mode* for front-camera photography and *Scan mode* for scanning QR codes and other types of barcodes. Lastly, ProCamera. includes a set of photo editing tools and convenient share options such as removing geographical metadata and resizing of the image before sending them to social media [72, 57, 15].

Screenshots of the application shown in Figure 2.8 present a selection of the application features. The leftmost screenshot shows the customizability of ProCamera. application. The top part of the screen displays a button for flash settings and information about shutter speed, exposure, and ISO sensitivity. The top bar ends with a button to switch between front and back camera. Bottom bar contains a button to access application settings (leftmost button), and on devices, with various cameras, the bottom bar offers an option to selected a specific type of camera. The rightmost button acts as a control panel which is visible as an overlay over the camera view-finder and contains shortcuts to various functions of the application. The second screenshot displays detail of the selected image with geographical metadata and navigation compass. The third screenshot shows a *Tiltmeter* functionality. Tiltmeter visualizes the device's attitude as a point moving on two axes. The last screenshot displays a live histogram and the ability to set focus and exposure separately (blue rectangle represents a focus, and the yellow circle represents exposure) [15].

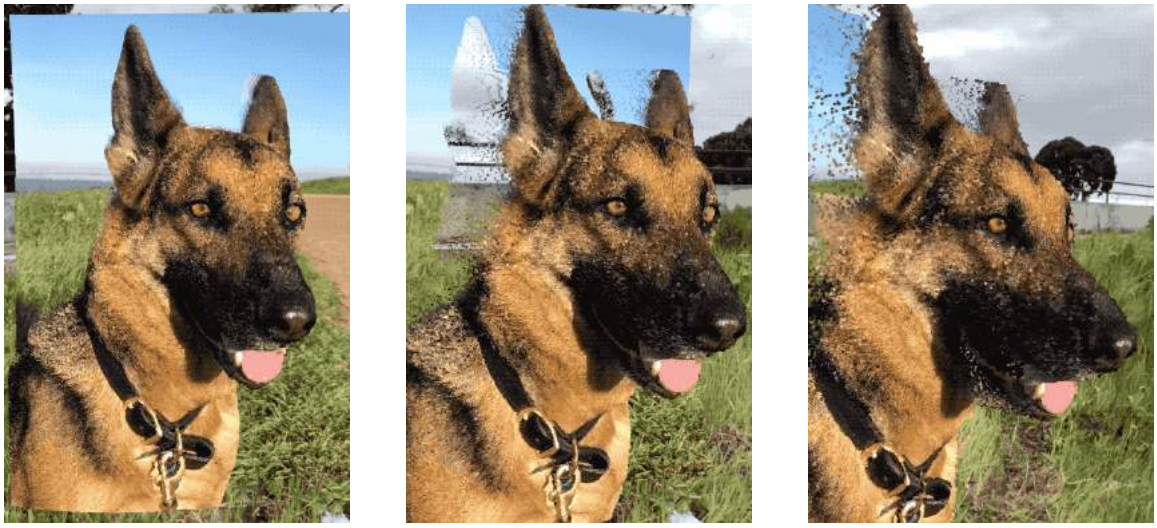

Figure 2.7: Screenshots from Halide's Augmented Reality Viewer. Images taken from an article *Halide 1.7: In Depth* [68]. Augmented Reality Viewer projects depth capture into augmented reality and allows the user to explore the photo in three dimensions. The technology is using ARKit framework from Apple to display the content in augmented reality [68].

2.3 Applications for sharing and viewing photographs

2.3.1 Instagram

Instagram is a social network focused on photo and video sharing. Instagram launched as a mobile application for iOS in October 2010. The application was released only for iOS, and the Android version was released in about year and a half later in April 2012. A limited web application was released in November 2012 followed by a release of Windows Phone application in November 2013 [6, 5, 1, 10].

The application initially started solely as a photo sharing social network. From the beginning, Instagram offered many photographic filters to modify the photo. Users can add tags and hashtags to help other users to discover relevant content and other users to follow. In June 2013, the application gained a video sharing functionality [13]. The application obtained more features such as private messaging between users (Instagram Direct, 2013), video and photo story sharing similar to Snapchat's Stories (Instagram Stories, 2016), and vertical video (IGTV, 2018) [9, 11, 20]. Until August 2015, photos and videos on Instagram were shared in 1:1 aspect ratio (square) and the square format was the distinguishing factor from other social networks [18].

Apart from the main application (Instagram), the company also released several stand-alone mobile applications – Boomerang, Hyperlapse, Layout, and IGTV. Boomerang is an application for creating short videos that loop back and forth [7]. Hyperlapse creates automatically stabilized time lapse videos [8]. Layout helps users to create personalized layouts from their photographs [12]. IGTV is a video application primarily focused on smartphones with videos solely in vertical format [20]. Length of videos is limited from minimum 15 seconds to a maximum of 10 minutes and 60 minutes for users with popular or verified accounts [21]. The stand-alone applications from Instagram offer an option to

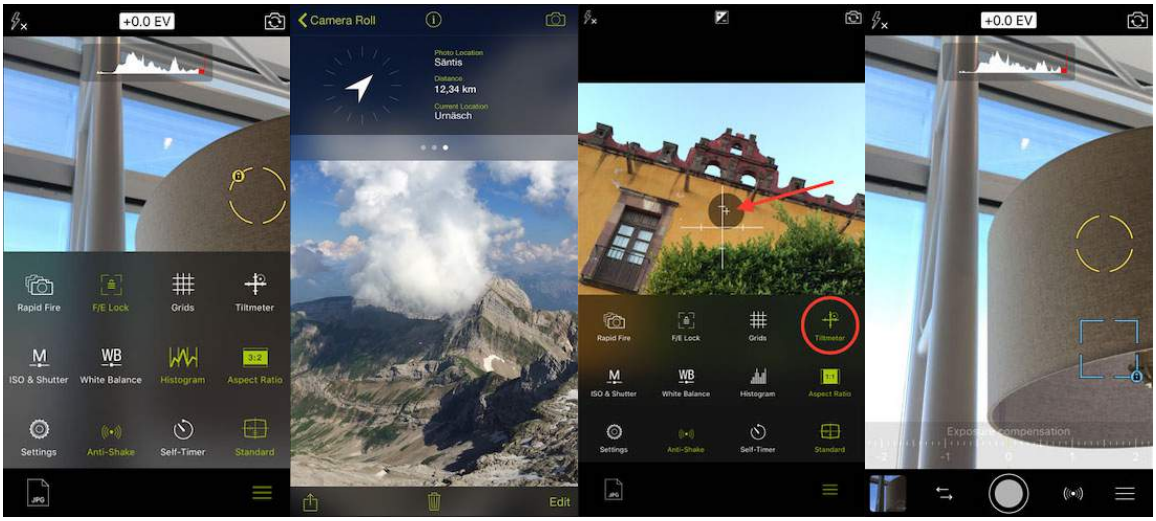

Figure 2.8: Screenshots from ProCamera. application (How To Use ProCamera App To Shoot Stunning iPhone Photos, 2017) [57]. The leftmost screenshots shows opened control panel containing shortcuts and options for various application's features. The second screenshots (What's New in ProCamera 8 v6.3, 2015) displays an image detail with geographical metadata and navigation compass [65]. The third screenshots shows Tiltmeter which represents current device attitude as a moving point on two axis. The last screenshots shows a live histogram and the focus (blue rectangle) and exposure (yellow circle) configured separately [14].

share the content to Instagram with IGTV being tightly integrated into Instagram's primary application [20].

2.3.2 Snapchat

Snapchat is a messaging application for sending multimedia content which is usually available only for a short period. The original application, first publicly available in 2011, focused primarily on person-to-person messaging – Android version of the application launched in 2012. The application gradually evolved and introduced *Stories* in which people share chronologically ordered content available for 24 hours and *Discover* providing an advertising platform for brands [17].

Snapchat is known for its mobile-first approach for social media, and a vital part of the application is the focus on users interaction. A well-known example of this is the augmented reality content users can interact within the Snapchat's camera and virtual stickers [46].

The application's version 5.0 for iOS released in June 2013 introduced new swipe navigation between the application screens [49]. Until then, the application honored a more traditional approach of screen navigation, as shown in Figure 2.9. The novel user interface design with swipe navigation unlocked new possibilities as Snapchat applications obtained more features. Figure 2.10 shows screenshots from Snapchat's application page on the App Store. The user interface, in comparison with the application's interface displayed in Figure 2.9, changed significantly. The navigation is multidimensional to the point that on the camera screen, swipes from left to right and up to down navigate the user to various application parts. Furthermore, instead of using multiple buttons, Snapchat makes use of

Figure 2.9: Screenshots of Snapchat application from 2011 (Snapchat's History: Evolution Of Snapchat And Timeline (2018) [17]). Back then, the application used common navigation between the application screens. The leftmost and middle screenshot reveals the use of a navigation bar to indicate the position in the application's flow to the user with back button returning the user to the camera screen. Even in the first version, the application's initial screen is the camera for the reason to empower users to create content instead of just consuming content of others [50].

gestures such as double tapping and holding for their functionality, changing the traditional meaning of the gesture at the same time [50].

By breaking the design rule *Don't Make People Think*, Snapchat requires to some extent a tribal knowledge to use. The application abandoned *intuitive user interface* in favor of *shareable user interface*. As a result, Snapchat makes people talk and consult their friends on how to use the application, which in the context of Snapchat being a social network, may, in the end, help Snapchat [50].

2.3.3 Flickr

Flickr is a company providing image and video hosting services through its website and mobile applications for iOS and Android. The website allows users to upload and manage their collection of photos, browse and add photos of other users into their favorites list and join groups with a common interest. Also, users can create photo books online from their albums on Flickr. Furthermore, the website allows users to view supported equirectangular images in 360° without any additional equipment. Additionally, Flickr offers *Flickr VR* application for compatible Galaxy smartphones. If the user owns a compatible Samsung Galaxy smartphone with Samsung Gear VR headset, they can explore supported photos and experience a 360° view of the image [19].

Flickr's mobile application offers almost the same features as the website, but the interface is optimized for smartphones. In comparison with the website, users of the mobile application can not order photo books or view equirectangular images [1]. However, the application allows users to take photos through the camera of the smartphone directly in the application. The application's camera supports several image filters and offers image enhancing, cropping and editing saturation, color balance, levels, and sharpness of the image.

To put the importance of smartphone photography into context, the *Most Popular Cameras in the Flickr Community* are iPhone 6, iPhone 6S, iPhone 5S, iPhone 7, and

[1] Flickr's application in version 4.10.1 was tested on iPhone X with iOS 12 (version 12.1.2.)

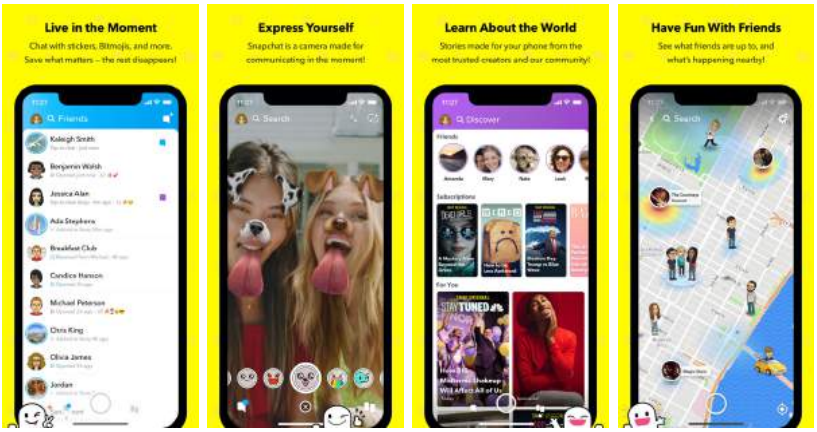

Figure 2.10: Snapchat's screenshots of version 10 from App Store [16]. The leftmost screen shows a chat screen. The second screenshot displays a camera using Snapchat's camera lenses. Camera lenses display augmented reality content on the subject. Users can access camera lenses on the camera screen by holding their finger on the screen. The third screenshot shows *Discover* screen. Stories from user's friends are displayed on top of the screen while subscription from brands and algorithmically generated content is displayed below. The last screenshot shows nearby friends and stories on a map.

iPhone 7 Plus. Same smartphones are the most popular camera phones. The rank 1 in *Camera Brands used in the Flickr Community* is Apple with top models consisting of iPhone 6S, iPhone 6, and iPhone 7. Samsung has a fourth position with top models being Galaxy S7, Galaxy S6, and Galaxy S5 [2]. Relevant charts from the Flickr's website are shown in Figures 2.11 and 2.12.

Data represented in charts in Figures 2.11 and 2.12 are normalized and shows the number of Flickr members who have uploaded at least one photo or video with a specific camera on a given day over last year. The up and down movement in the chart represents the change of camera's popularity relative to all other cameras used by Flickr members [2].

2.4 Applications for capturing the same object in different times

2.4.1 Computational Rephotography

Computational Rephotography is an article which presents an interactive, computational technique for rephotography. The solution focuses on the task of matching the viewpoint of a reference photograph at capture time. The software allows users to point the camera towards the scene shown in the reference image and in real-time it estimates and visualizes the camera motion to reach the desired viewpoint [45].

From the implementation point, the technique is built on top of existing computer vision algorithms. Camera pose estimation is computed by two techniques - robust and lightweight estimation. The robust camera pose estimation is based on computing current camera location relative to the first camera location. The use of the first frame instead of the reference frame is to avoid degeneracy when the user gets close to the desired viewpoint. Correspondence estimations between the first and the current frame are computed by using SIFT feature points. Real-time estimation of the essential matrix between two

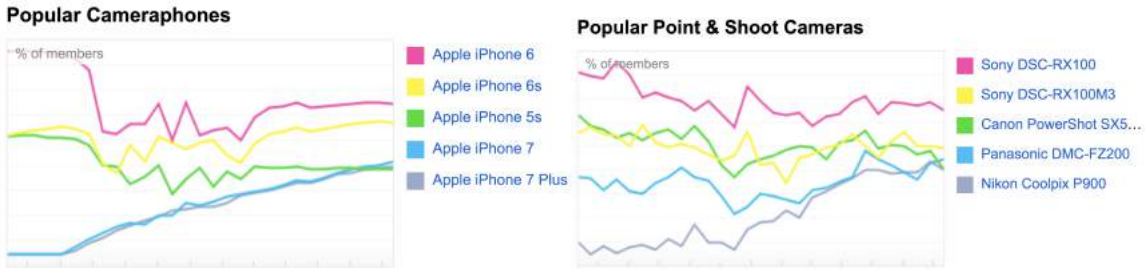

Figure 2.11: The first chart (Flickr Camera Finder [2]) shows the most popular cameraphones in the Flickr Community. The total number of photos captured by many cameraphones can be higher as Flickr relies on automatically detecting the camera used to take the photo, but this is not usually possible with camera phones. The second chart (Flickr Camera Finder [2]) shows the most popular point and shoot (compact) cameras [2].

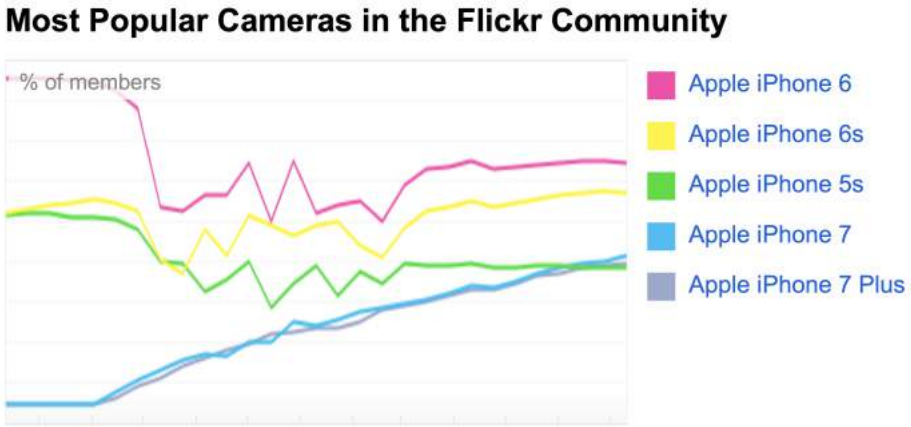

Figure 2.12: The chart (Flickr Camera Finder [2]) shows that Apple's smartphones are the most popular cameras in the Flickr Community, surpassing DSLR cameras, compact cameras and the competition's smartphone models [2].

calibrated cameras is done by using Stewénius's five-point algorithm. Inliers and the best fitting essential matrix are found by using m-estimator sample consensus (MSAC) [45]. The computation flow of the camera pose estimation is shown in Figure 2.13.

The lightweight pose estimation is computed by using KLT implementation for tracking feature points. The implementation performs an affine check and multi-scale tracking that refines from coarse to fine resolution. This lightweight computation runs at more than ten frames per second [45].

Robust and lightweight estimations are interleaved in order to provide real-time updates to the user. Each of the estimations runs on its dedicated thread. Third thread communicates with a camera. At the end of the robust estimation, a set of inliers is passed to the lightweight estimation process. After the lightweight estimation acquires a new set of inliers, it starts tracking from the next frame of the key frame [45].

Prototype implementation relies on a laptop connected to a camera. The computer vision tool running on the laptop shows the user arrows directing him towards the computed camera pose. Also, the tool allows switching between multiple visualizations – edge visualization, flipping visualization and visualization with a reference camera projected onto the

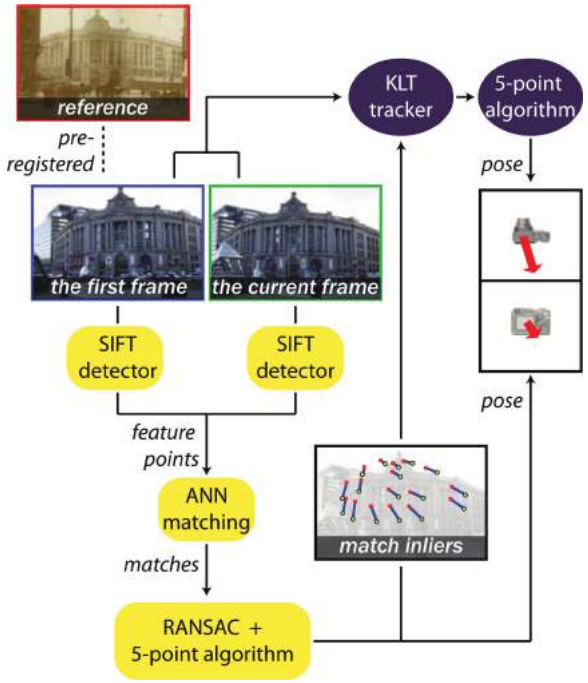

Figure 2.13: The diagram (Computational Rephotography, 2010 [45]) shows the process of the camera pose estimation. The robust camera estimations parts are represented by yellow boxes while purple ellipses represent the lightweight estimation [45].

current frame. In the edge visualization, a linear blend of the edges of the reference image and the current scene after rotation stabilization is shown. In the flipping visualization, users can flip between the reference photo and the current frame from the camera. Lastly, in the third visualization, a 3D model of the reference camera as is projected as a visual target onto the current frame. In the extensive user interface evaluation, done by two pilot user studies and two final user studies, the edge visualization was chosen as a primary visualization for the prototype interface [45].

As a result, the use of a computational system navigating users with arrow visualization helps users take more accurate rephotographs in every test case than with a simple visualization. The average error of the method is 40% of the average error with the linear blend. It is important to note that the article states that if historical photographs are taken into account, it can take approximately 15–30 minutes to reach the desired viewpoint. Moreover, the use of the laptop with a connected camera with a tripod limits the portability. Researches envision their solution to run directly on the camera in the future [45]. Nowadays, it would make sense to run a similar application on a smartphone device instead of a professional DSLR camera.

2.4.2 Fast and Reliable Computational Rephotography on Mobile Device

Fast and Reliable Computational Rephotography on Mobile device presents a fast and reliable computational rephotography method with near real-time navigation on a mobile device, guiding the user through the rephotography process. The article introduces fast matching, which alternately uses features and optical flow to accomplish near real-time navigation

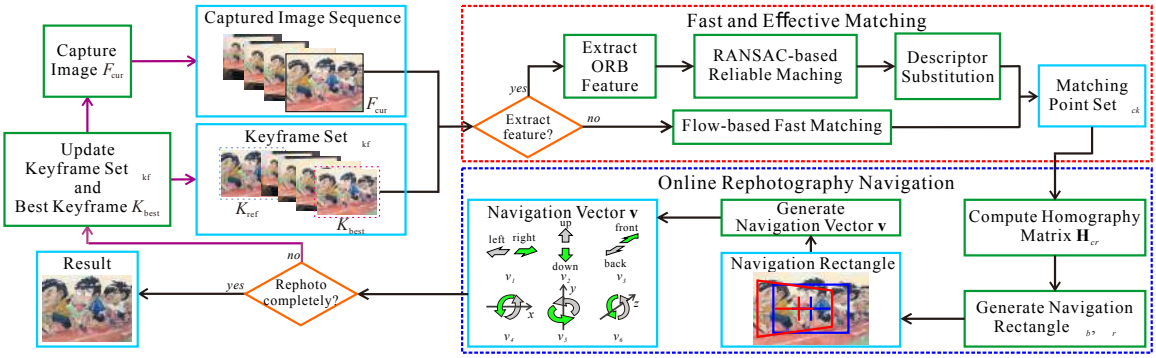

Figure 2.14: Computational process of the fast rephotography method on a mobile device (Fast and Reliable Computational Rephotography on Mobile Device, 2018 [69]). During the computation, a keyframe and best frame are retained. After a new frame is captured, the algorithm performs fast and effective matching between the current frame and best frame and outputs matching point set. Next step is a computation of homography transformation between current and reference frame and generating navigation rectangle and vector. Navigation rectangle and vector are used to guide users to the intended camera pose. Afterward, the keyframe set is updated, and unless the rephotography process is finished, computation continues with a new frame [69].

on a mobile device. Furthermore, the authors also present techniques for improving the robustness of the method on great illumination and target changes [69].

Instead of relying on robust but more computationally expensive methods such as SIFT and SURF, authors chose more lightweight yet less robust ORB feature descriptor. ORB feature matching is followed by a fast optical flow tracking in a few succeeding frames. After the extraction of ORB feature points, matching outliers are removed through the use of RANSAC to enhance the matching accuracy. As light conditions in reference and current frame can differ, the procedure makes use of substituting feature descriptors in the reference frame by corresponding feature points in the current frame. However, this substitution of corresponding feature descriptors happens only during specific conditions discussed more in detail in the original paper. In contrast with the system introduced in section 2.4.1, the *Fast and Reliable Computational Rephotography on Mobile Device* uses only the mobile device screen to guide the user. As a result, the navigation system had to be changed due to the limited screen size of the mobile devices, and the paper proposes the use of navigation rectangles. Navigation rectangles are computed from the homography matrix and corresponding points between the current and the reference frame. Based on the relation between the two navigation rectangles, a navigation vector is generated. The vector indicates the motion direction [69]. Figure 2.15 shows navigation rectangles and the appropriate navigation vector.

Results presented in the paper state that the authors' method makes the rephotography process more successful and less time-consuming. In summary, when the user is guided by the method from *Fast and Reliable Computational Rephotography on Mobile Device*, the AFD (Average Feature-point Displacement) is lower than other methods in nine cases from total eleven test cases. Time spent with the rephotography process is the lowest in all test cases. The amount of time saved is in tens of seconds [69].

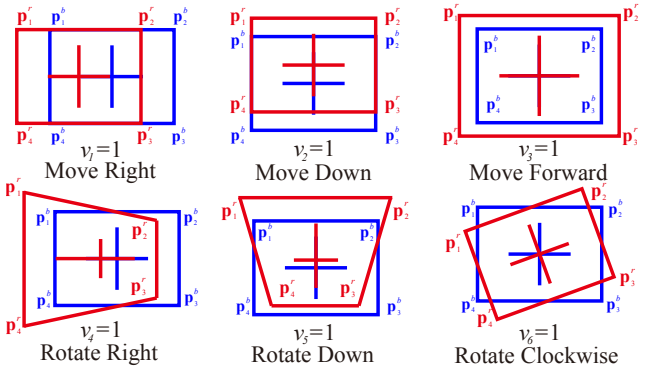

Figure 2.15: The interpretation of motion in six degrees of freedom using navigation rectangles with a corresponding navigation vector. (Fast and Reliable Computational Rephotography on Mobile Device, 2018) [69].

Figure 2.16: The user interface of the application (Collaborative Rephotography, 2013 [74]). In the first screen, the map shows various locations to rephotograph as pins. After arriving at the location, the user can verify the site to rephotograph. While capturing the site, a transparent overlay helps the user to align the camera correctly. Finally, the user has an option to upload their submission to the rephotography project [74].

2.4.3 Collaborative Rephotography

Collaborative Rephotography demonstrates a tool allowing multiple users to rephotograph various locations over time. The resulting tool is a mobile application in which the user can explore sites on a map, choose the desired site to rephotograph and rephotograph the location using a linear blend between the reference and current subject. The most valuable contribution in comparison with other rephotography projects is lowering the barrier to entry for volunteer users [74].

Screenshots from the application can be seen in Figure 2.16 and 2.17. The application shows a full-screen map on which pins represent locations for rephotography. When the user selects a specific location to rephotograph, they can verify the photographed subject. During the rephotography process, a transparent overlay on the screen helps the user align the current camera frame to capture a more accurate rephotograph. After the rephotograph is captured, the user has an option to upload the image.

The application introduced in *Collaborative Rephotography* paper helps to collect submissions to rephotography projects from a community. Furthermore, by creating a better user experience, the application makes it easier for volunteer users to participate in the

Figure 2.17: Screenshots from the application showing one of the participating rephotography projects (Collaborative Rephotography, 2013 [74]). In this case, the application helps to rephotograph and monitor the health, changes and local environment of street trees in New York City [74].

rephotography projects. The paper mentions national scenic overlooks and the Gowanus Conservancy in NYC as projects using the application. As a result of the use of the application, participating projects collect more various data more frequently [74].

2.4.4 Rephotography Using Image Collections

Rephotography Using Image Collections presents a novel technique to create rephotographs. Instead of providing some computational navigation system or blending reference and current image to help the user to capture a more accurate rephotograph, their solution relies on a collection of modern images around the target scene. Other papers in computational rephotography such as *Computational Rephotography* [45] and *Fast and Reliable Computational Rephotography on Mobile Device* [69] provide a near real-time solution to guide the user to find the accurate viewpoint of the reference photograph. However, in some cases, the users might not be able to go back to the original place to capture a rephotograph. Additionally, even with a real-time system, it can be hard to guide the user to the correct vantage point. The proposed technique generates a rephotograph of a historical photograph with some modern images. Using the historical photograph as a reference, it renders an image with the same view from the collected new photographs [60].

Figure 2.18 shows the rephotography process of the proposed method. The technique takes a historical reference photograph and a collection of modern images around the target scene as an input. At the beginning of the rephotography process, the system estimates camera parameters of the modern photographs through a camera calibration. Further, it constructs a 3D point cloud of the scene and then estimates depth maps. Camera calibration consists of estimating the intrinsic and extrinsic matrices of each camera. The 3D point cloud of the scene is generated using a structure from motion technique[2]. Next step is the estimation of camera parameters of the reference photograph. Instead of using SIFT or ORB features as in [45] and [69], the estimation of the camera parameters relies on the user-specified corresponding features between the reference photograph and 3D point cloud. This is due to the differences in photograph qualities, noise, and perspective, which contribute to a lot of incorrect feature correspondences when using SIFT. In this point, the system posses all camera parameters of each photograph and its depth map. The method

[2]Structure from motion is a method which aims to recover the camera motion and scene structure from image sequences [70].

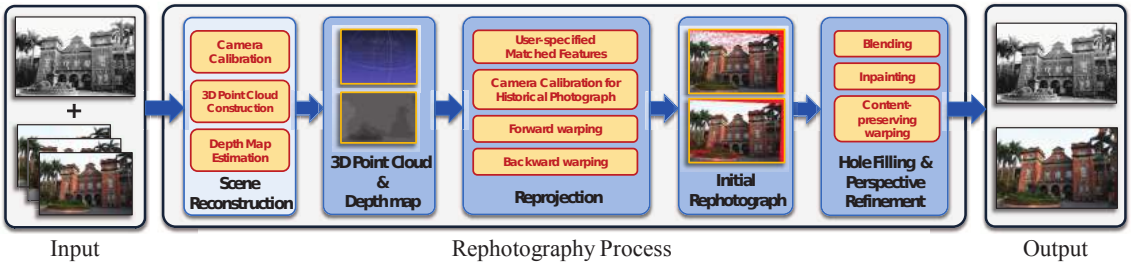

Figure 2.18: Overview of the rephotography process (Rephotography Using Image Collections [60]). The input of the system is a reference (historical) photograph, and a sequence of recent photographs captured close to the subject. First, the method estimates the camera parameters and constructs 3D point cloud of the scene by using structure from motion technique. Next, it asks the user to specify corresponding features between the reference image and 3D point cloud. After that, the system estimates camera parameters of the reference photograph and computes depth map of every photograph. Then, it computes forward and backward warping and generates two initial rephotographs. Finally, the system performs image inpainting and content-preserving warping and produces the resulting image.

then projects the 3D point cloud onto the historical viewpoint. This operation is called forward warping. As a result of the forward warping, a new view matching the viewpoint of the reference photograph is rendered, and its depth map is computed at the same time. Forward warping is followed by backward warping. In backward warping, pixels of the new view are reprojected back to each input photograph to render an initial rephotograph.

Before the final output is rendered, the system performs operations to handle possible mismatch of the scene structure. First, to fill some holes in the image after the forward and backward warping, the method blends the two rendered photographs. Furthermore, due to the occlusion, projection error or lack of scene structure information, the system performs inpainting to fill holes caused by one of the errors. The inpainting selects pixels in the holes individually in a best-first algorithm. After the selection of the pixel, it searches the entire image and finds a squared patch which is the most similar to the patch centered at the selected pixel. Finally, the system performs content-preserving warping to fix the mismatch of the scene structure. In content-preserving warping, a grid mesh constructed on the resulting image is warped in a way that the matching points between the result and reference image locate precisely at the position of the matching points of the historical image [60].

Rephotography result of the proposed technique can be seen in Figure 2.19. The leftmost image shows the reference photograph selected from the image collection shown in the second image. The second image shows the remaining image collection which is used to generate the result rephotograph. The third image displays the result of the rephotography process. The last image shows the difference between the rephotography result and the reference photograph. Darker shades of a red color signal the fact that the error is larger [60]. Table 2.1 shows computational times of the rephotography process and additional details about the selected photographs.

Figure 2.19: The result of the rephotography process and its comparison with the reference photograph (Rephotography Using Image Collections [60]). Leftmost image displays the reference photograph. The second image shows the collection of remaining photographs used in the system. The third image contains the generated result from the rephotography process. The last image displays the difference between the reference and result image.

Number of photos	16
Resolution	855 × 570
Depth map (sec.)	5.558
Ref. resolution	600 × 450
Ref. view (sec.)	10.93

Table 2.1: Computational times and details about the experimental data (Rephotography Using Image Collections [60]). The table shows computational times and additional details about the photographs in Figure 2.19.

Chapter 3

Mobile Application Development for iOS

3.1 Developing Applications for iOS

3.1.1 Overview of iOS Operating System

iOS is a mobile operating system for iPhone, iPad, and iPod Touch devices. It is developed by Apple, Inc. and the system was first introduced alongside the introduction of the first iPhone in 2007 [58]. iOS manages the device hardware and provides technologies required to implement native applications. The system is layered, and the set of layers is shown in Figure 3.1. Most of the system interfaces are delivered as a special package called framework. The framework is a directory that contains a dynamic shared library and the resources such as header files, images, and others [44]. The application-framework of iOS is called Cocoa Touch. The most critical framework for applications is the UIKit framework, which is the core Cocoa framework in iOS [44] [28]. Following frameworks can be found at each layer of the iOS system:

- **Core OS** level contains kernel, file system, networking infrastructure, security, power management, and device drivers [28].
- **Core Services** is a collection of frameworks providing the following services – string manipulation, collection management, networking, URL utilities, contact management, and preferences. Furthermore, it contains frameworks for hardware-based features such as GPS, compass, accelerometer, and gyroscope. Examples of frameworks in this layer are Core Location and Core Motion [28]. Additionally, this layer contains both Foundation and Core Foundation frameworks providing abstractions for common data types. Lastly, the Core Services layer contains the Core Data framework for object graph management and object persistence, which is discussed more in detail in section 3.1.2 [28].
- **Media** layer include frameworks supplying graphical and multimedia services to the Cocoa Touch layer. The layer includes frameworks such as Core Graphics, Core Text, Core Animation, AVFoundation, and Core Audio.
- **Cocoa Touch** layer consists of frameworks directly supporting iOS applications. An of frameworks in the Cocoa Touch layer is Map Kit [28].

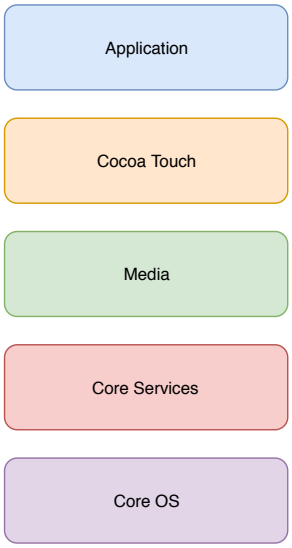

Figure 3.1: Cocoa in the architecture of iOS Operating System. The hierarchical diagram shows framework layers each giving the application access to its underlying technologies [28]. Figure based on *Cocoa in the architecture of iOS* [28].

Two particularly essential frameworks for developing applications for iOS are UIKit and Foundation.

UIKit is a framework providing objects to an application which are used to construct and manage its user interface [28]. Furthermore, the framework defines the structure for application behavior, including event handling and drawing [28].

Foundation defines a base layer of classes usable for any Cocoa application. It provides objects for primitive data types, collections, and operating-system services and serves as an object-oriented version of the Core Foundation framework. Furthermore, it introduces conventions for memory management, object mutability, and notifications [28] [44].

3.1.2 Persisting data in Core Data

Core Data is an object graph management and persistence framework used for managing the model layer objects in iOS applications [37]. The framework is located in the Core Services layer. Core Data abstracts the details of the mapping application's model objects to store. Thus it makes it easier to save data without managing a database directly [30]. The framework is initialized in the application as a stack consisting of Model, Context, Store Coordinator, and Persistent Container. The stack, shown in Figure 3.2, has to be initialized before accessing application data [30]. The initialization process prepares Core Data for creation and data requests of the application data.

Model describes the data accessed by the Core Data stack [37]. The `NSManagedObjectModel` contains one or more objects representing the entities in the schema [30]. Furthermore, it maintains a mapping between each of its entity objects and an associated managed object class. The model is usually created by using a data modeling tool in Xcode. However, the model can be created in code when needed [30]. Finally, `NSManagedObjectModel` is loaded into memory as a first step of the creation of the Core Data stack [37].

Context is a group of related model objects representing an internally consistent view of one or more persistent stores [31]. Changes made to the managed objects are stored

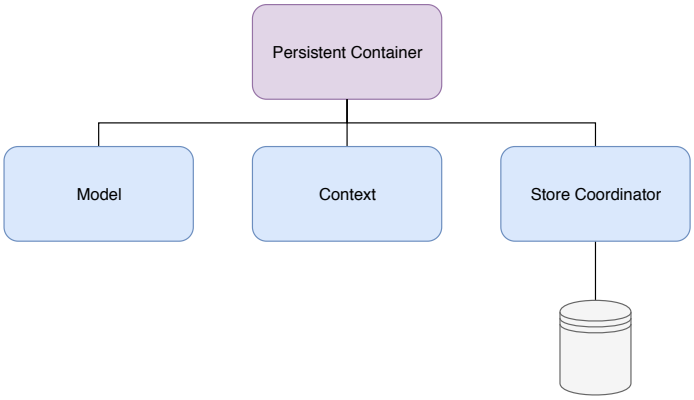

Figure 3.2: The diagram shows a Core Data stack which manages and persists the application's model layer. Store Coordinator is responsible for fetching and saving instances of the application types from stores. Context manipulates and tracks changes to instances of the managed object. The model contains one or more objects representing entities in the application's schema. Lastly, Persistent Container encapsulates Core Data stack in an iOS application and provides a simplified creation and management of the stack. Figure based on *Core Data Stack* [37].

in memory in the corresponding managed object context. Only after the managed object context is saved, the changes are persisted to one or more persistent stores [31].

Store Coordinator is used by instances of Context (`NSManagedObjectContext`) to save object graphs and retrieve model data from persistent storage. The context uses the coordinator to access the model. Moreover, the coordinator unifies and presents a group of persistent stores as an aggregate store to the managed object context [31]. There are several types of persistent stores – SQLite, Binary, and In-Memory. Fourth persistent store type XML is not available on iOS [30].

Persistent Container encapsulates Core Data stack in the application. It simplifies the creation and management of the set of classes that collectively supports the application's model layer. `NSPersistentContainer` is used to set up those classes all at once [37].

Figure 3.3 shows the data modeling tool in Xcode. The left side of the screen shows a list of entities. New entities are added using a button located at the bottom of the screen. The middle of the screen shows attributes and relationships for a selected entity. The right side accompanies an inspector where the corresponding class for the entity can be set. The data modeling tool supports two different editor styles – table and a graph style. Figure 3.3 shows the table style with the entities list, editor area, and data model inspector. Figure 3.4 shows the graph style. The graph style visually represents relationships between entities and shows their attributes. New relationships can be made between entities by control dragging from one entity to another.

3.1.3 Showing map and annotations of nearby places

The iOS operating system offers a framework in the Cocoa Touch layer to display maps in applications – MapKit. MapKit allows developers to embed maps into their applications, add annotations and overlays to the map and customize call outs. The framework also supports reverse geocoding and can be used to determine placemark information for map coordinates [29].

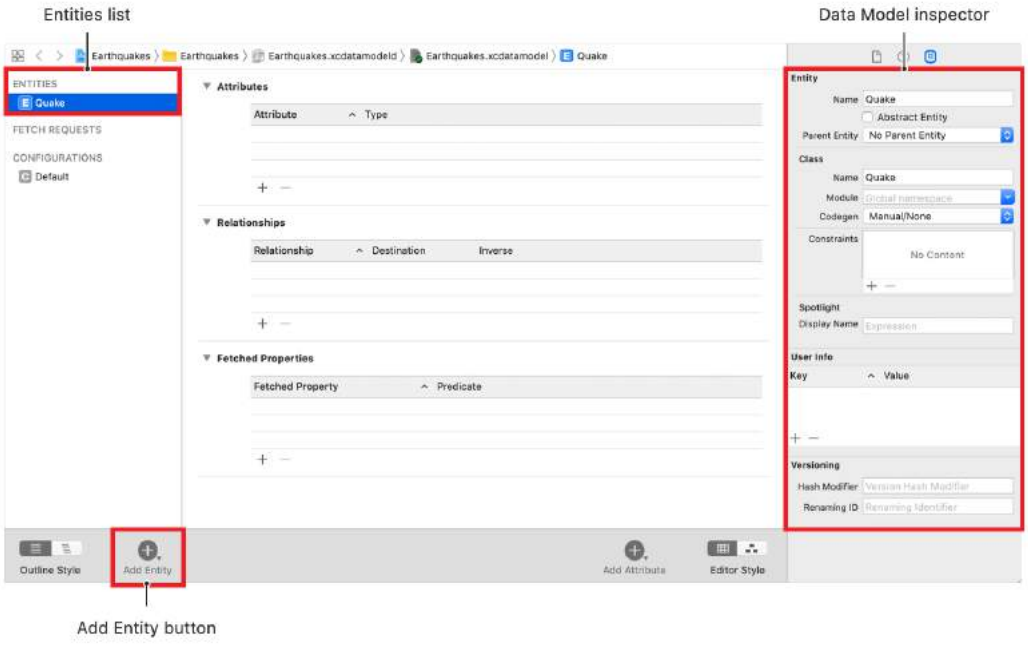

Figure 3.3: Data modeling tool in Xcode used for creating the Core Data model. The screenshots show the interface of the data modeling tool with entities list on the left side, editor area with attributes, relationships, and fetched properties list and data model inspector on the right. The bottom part of the screen contains a button for adding new entities, attributes, and a switch for changing the editor style between table and graph style. Figure source *Configuring Entities* [37].

MapKit supports following map types – standard, satellite, hybrid, satellite flyover, hybrid flyover, and muted standard [41]. Standard map type is a street map that shows the position of all roads and some road names [41]. Satellite type includes images of the area from a satellite while hybrid includes a satellite image of the area with roads and road names information on top. Both satellite and hybrid flyover map types include satellite images with flyover data when available. The flyover is a view from the air with photo-realistic and interactive 3D models. As the time of writing, two cities in the Czech Republic support flyover mode in Apple Maps – Brno and Prague. Lastly, map type muted standard is a street map where the additional application data are emphasized over the underlying map details [41].

Figure 3.5 contains images of four different map types – standard, satellite, hybrid, and satellite flyover. The images show the difference between different map types. The standard map type shows roads with road names and can be used for navigation applications or applications where the information about roads and streets is more important to the user than the view and layout of the area of interest. Satellite and hybrid map type shows a satellite image of the area while the hybrid map type has additional information about road names. The satellite and hybrid map type can give users a better understanding of the area, such as the structure of buildings, green areas, and others. The last image shows a flyover that displays photo-realistic 3D models. However, flyover map type can be only used in supported cities.

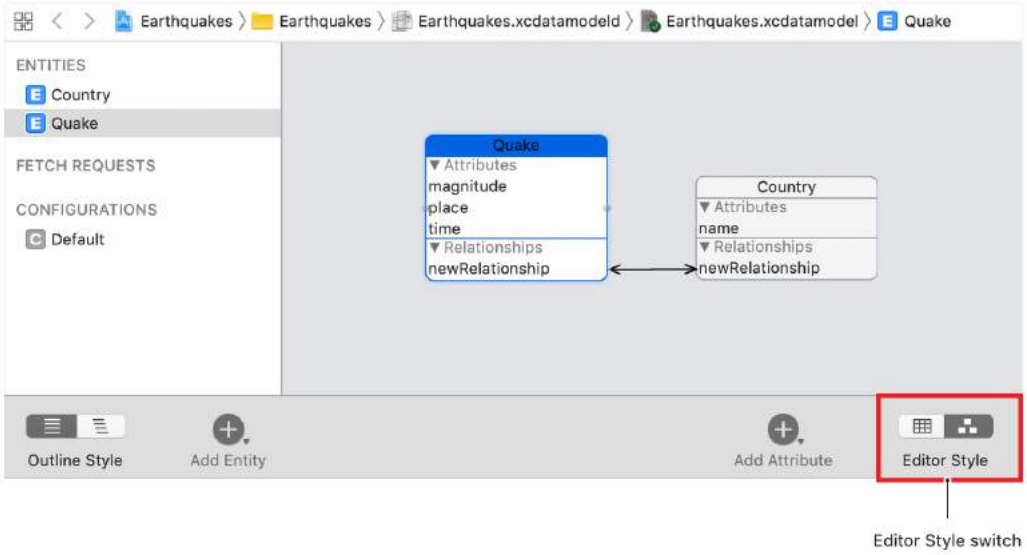

Figure 3.4: The screenshot shows a graph style of the editor with two attributes and a relationship between them. Apart from the relationship, the graph editor style also displays entity attributes. The editor style is selected by using the switch at the right bottom part of the data modeling tool. (*Configuring Relationships* [37])

The framework supports map annotations based on location-specific data with customized annotation views and callouts [29]. The developer provides annotation as an object that conforms to `MKAnnotation` protocol with required coordinate property and optional title and subtitle properties. MapKit offers default annotation views for point-based annotations – `MKPointAnnotation`, `MKMarkerAnnotationView` and `MKPinAnnotationView`. `MKPointAnnotation` is an object tied to the specified point on the map. `MKMarkerAnnotationView` displays a balloon-shaped marker at the designated location. Finally, `MKPinAnnotationView` displays a pin image on the map. Figure 3.6 shows a customized map annotation with an application provided shape and a callout with a custom image, title, and subtitle.

3.1.4 Network communication

Foundation framework includes an API for downloading and uploading data to URL endpoints. The API is provided by `URLSession` class and related classes. Additionally, it allows downloads in the background when the application is not running or when the application is suspended. Authentication and notifications about redirections and other network events are supported through delegate methods.

The URLSession API can be used to create one or more sessions, each managing a group of related data transfer tasks [42]. Individual tasks represent a request for a specific URL. For basic requests with no configuration object, the API has a singleton shared session. The shared session is not as customizable as created sessions. However, URLSession contains several session types with the following configurations:

- *Default* session similar to the shared session. The difference between the two is that the default session allows more configuration and supports an incremental download of the data through its delegate methods [42].

Figure 3.5: Different map types available in MapKit framework. The leftmost screenshots shows a standard map type with a street map and road names. The second image shows a satellite map type and the third screenshots displays hybrid map type. In hybrid map type satellite image contains road information. Last screenshot shows a satellite flyover of the area with 3D models. [41].

- *Ephemeral* session, in contrast with the shared session, does not persist caches, cookies, or credentials to disk [42].
- *Background* session allows performing upload and download tasks in the background while the application is not running [42].

Tasks used inside the mentioned sessions are responsible for uploading data to a server and then downloading response data into a file on disk or as objects in memory. URLSession provides three types of tasks:

- *Data* task is a task that sends and receives objects of type `Data`. This task is designated for a short request to a server [42].
- *Upload* task is similar to the data task but supports uploads in the background. Furthermore, upload tasks often send data in the form of file [42].
- *Download* task retrieves data from a server in the form of file and supports background upload and download of data [42].

Asynchronicity in URLSessions can be achieved by using completion handlers or calls of appropriate delegate methods. Completion handler blocks are called when the transfer finishes with success or with an error. With the second approach, a class conforming to respective protocols handles the delegate methods and implements custom logic. Figure 3.7 shows the flow when completion handler is used to handle success and error paths of *URLSessionDataTask*. On the contrary, Figure 3.8 displays the process of using appropriate delegates.

Figure 3.6: Screenshot from Apple's sample code application *Annotating a Map with Custom Data*. The image shows a custom map annotation with a callout that contains an image, title and subtitle.

Figure 3.7: The diagram shows the flow of receiving result from `URLSessionDataTasks` by using a completion handler to take care of error and success path of the transfer process. Figure based on *Fetching Website Data into Memory* [42].

Supported networking protocols in URLSession API are `HTTP/1.1`, `HTTP/2` and `SPDY`. The API supports by default `data`, `file`, `ftp`, `http` and `https` URL schemes. Developers can subclass `URLProtocol` and provide custom networking protocols and URL schemes. However, custom networking protocol or URL scheme is only usable for the application's private use [42]. After the release of iOS 9.0 and OS X 10.11, a new security feature *App Transport Security* (ATS) is enabled by default for all HTTP connections made with URLSession. The App Transport Security requires to use HTTPS over HTTP in all connections.

3.2 Programming camera-based applications

3.2.1 Capturing photos using AVFoundation

AVFoundation framework unites four major technologies for working with multimedia on Apple platforms. It supports capturing, processing, synthesizing, controlling, importing, and exporting multimedia content.

The AVFoundation Capture subsystem provides functionality for capturing photos and recording video. It supports configuration of built-in cameras, microphones, and external

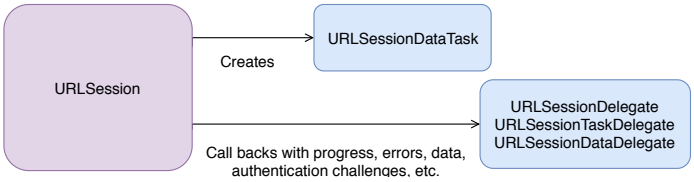

Figure 3.8: The implementation of delegate to receive results from a task. Figure based on *Fetching Website Data into Memory* [42].

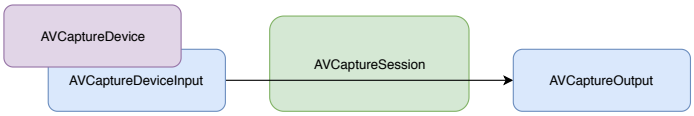

Figure 3.9: AVFoundation Capture subsystem architecture with its three main parts – capture sessions, inputs and outputs. Figure based on *Cameras and Media Capture* [36].

capture devices. Through its video, photo and audio capture services the subsystem can be used to build a custom camera UI with a combined photo or video shooting inside the application. Furthermore, it provides direct control over the photo and video captures, such as focus, exposure, and stabilization options [36]. Additionally, the subsystem can be used to produce photos in RAW format, depth map, or videos with custom timed metadata. Finally, it gives access to pixel or audio data streaming directly from a capture device [36].

The Capture subsystem is divided into three main parts – sessions, inputs, and outputs. Capture sessions connect one or more inputs to one or more outputs [36]. Inputs are sources of media, including capture devices built-in devices on iOS and Mac devices. Outputs obtain media from inputs and produce outputs such as raw pixel buffers or movie files [36]. Figure 3.9 displays the architecture of AVFoundation Capture subsystem.

It is recommended to interact with and configure `AVCaptureSession` on a dedicated serial dispatch queue to prevent blocking of the main queue on which the UI rendering and interaction with user happen [36]. After receiving inputs, the session sends the data to relevant outputs for processing. Furthermore, the application needs to provide `AVCapturePhotoSettings` with configuration parameters such as focus, flash, and resolution. In camera applications, generally, the process of capturing a photo is initiated by the user by pressing a button functioning as a shutter button on a real camera. When the user taps on the button the application initiates a photo capture with specified settings by calling `capturePhotoWithSettings:delegate:` method on a selected `AVCapturePhotoOutput` object. Once the `capturePhotoWithSettings:delegate:` method is called, the process for starting photography ends. Operations on that individual photo capture are handled in delegate callbacks [36]. Figure 3.10 summarizes the capture process and shows its timeline with delegate methods.

3.2.2 Persisting user captured photographs

There are multiple ways to save multimedia data on the iOS platform. The first method to analyze is saving photographs as a file on the device. Foundation framework includes `NSFileManager` class that supports saving and loading files from and into the file system. This method does not require authorization from the user, and the application can create a custom folder structure inside the application's bundle. When the most common workflow

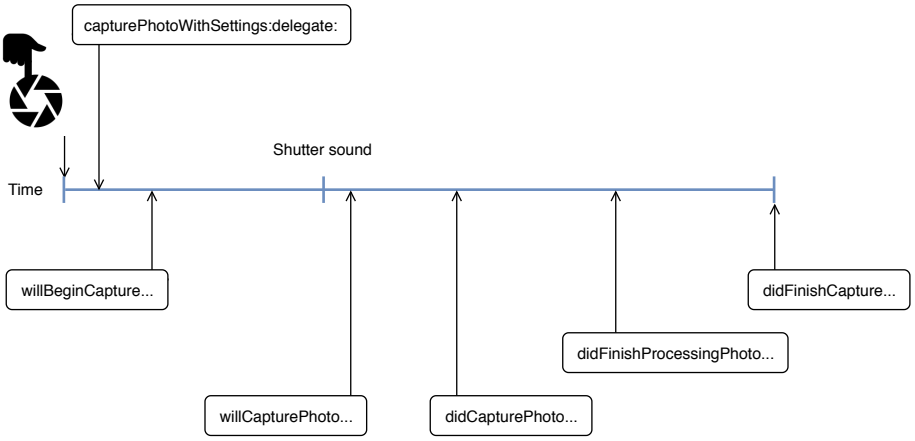

Figure 3.10: The process of capturing a still photo using AVFoundation framework. After the users taps on a shutter button `capturePhoto:WithSettings:delegate:` method on `AVCapturePhotoOutput` is called. From this point, the process of the photo capture gets handled in the delegate methods. Figure based on *AVCam: Building a Camera App* [36].

of camera-based application is taken into account, the application creates folders based on albums and stores photographs based on their belonging to the album. However, when the application model layer is managed in the database, the developer needs to track file paths of all saved photographs and their corresponding album. Therefore, manipulating with data becomes more troublesome and error-prone.

The second approach would be to save image data directly into the SQL database. In this scenario, the photo model would have an attribute containing image data. The relationship between the photo and its respective album is achieved by a one-to-many (as long as one photo can be only in one album) between an album and photos. Moreover, the application model is now united, and the metadata are saved in the same way as image data. The model can be extended with relationships between its entities and common CRUD[1] operations are easier to do then in the first method. However, saving large binary data into SQL databases is an anti-pattern, and it can negatively influence the performance of SQL operations on affected entities. Even if the model is optimized with an entity holding only the down-sized thumbnail data, the database is bloated with large binary data and retrieving hundreds or thousands of entities can become slow.

Last discussed method is using Core Data to save and retrieve user captured photographs. This approach combines the positives of the two previously mentioned approaches while suppressing their negatives. Core Data allows to save extensive binary data as an attribute of an entity but saves the data outside of the SQL database. At the same time, the framework automatically manages the data and performs appropriate operations on the device's file system. From a developer perspective, objects of the specified managed class get fetched from the database, and their attributes contain the image data.

Furthermore, if the developer provides a value transformer class, the framework can automatically transform raw data from a file system into an UIKit class such as `UIImage`. It is important to note, that becomes of the nature of Core Data being an object-graph, the objects from the database are being held in the memory. It is the developer's responsibility to design the application in such a way that it does not fetch hundreds or thousands of

[1] CRUD stands for **C**reate **R**ead **U**pdate **D**elete.

photographs into the memory. This scenario results in the application getting killed by the system to free the device's memory. A viable architecture with Core Data as a persistence layer would consist of a class incrementally fetching tens of photographs visible in the user interface while maintaining a low memory footprint. Furthermore, by levering Core Data's *faulting*[2] mechanism and incorporating a thumbnail entity with a reasonably sized image data, the application can perform CRUD operations on such data effortlessly.

3.3 Using location and motion services in mobile applications

3.3.1 Core Location

Core Location framework provides services for resolving device's geographic location, altitude, orientation or position relative to a nearby iBeacon[3] [38]. Core Location uses all available on device hardware to collect data. The hardware includes Wi-Fi, GPS, Bluetooth, magnetometer, barometer, and cellular hardware [38]. The use of Core Location services in the application has to be authorized by the user. When the request for permission to use location services is granted by the user, and the framework determines if the required services are available, the services can be started using a `CLLocationManager` object. Results from location services are handled in the associated delegate object within its delegate callbacks [38].

The framework provides three unique services for retrieving the user's location. The services differ in authorization and power requirements, each providing different advantages. The developer can use one or more services at different times, depending on the application's needs [38]. Table 3.1 summarizes available location services.

Visits location service is the most power-efficient method to retrieve location data. It delivers location updates when the user has spent time in one location and then moves on. Every update includes both the location and the amount of time spent at that location [38]. This service is used for detecting patterns in the user's behavior, but it is not intended for navigation or real-time activity tracking. Visits location services require `Always` authorization.

Significant-change location service is a less power-demanding alternative to track the user's location without the need for frequent updates or the precision provided by GPS [38]. It depends on more power-conscious alternatives to determine the user's location and sends updates only when significant changes to the user's location occur. The service requires `Always` authorization.

Standard location service is a configurable and general-purpose method for fetching the user's location in real time. It uses considerably more power than its alternatives. However, it delivers the most accurate and immediate location data. The service is intended for real-time navigation or recording a user's path. The service requires `In-Use` or `Always` authorization.

[2] Faulting reduces the application's memory usage by keeping placeholder objects (faults) in the persistent store [30].

[3] Devices with iBeacon technology can be used to establish a region around an object. The technology allows the iOS device to determine when it has entered or left the region. Furthermore, the device provides an estimation of closeness to the beacon. The technology itself leverages Bluetooth Low Energy (BLE) [40].

Service	**Power-efficiency**	**Location precision**	**Authorization**
Visits location service	The most power-efficient	Delivers location data when the user stays on the same location for some time. The service also sends the amount of time spent at the given location.	Always
Significant-change location service	Power-friendly	Tracks user's location with lower-power services and sends location updates only when important changes to that location occur.	Always
Standard location service	Uses significantly more power than the alternatives	Delivers the most accurate and immediate location data.	In-Use / Always

Table 3.1: Overview of different location services with their corresponding power-efficiency, location precision and required authorization information [38].

3.3.2 Core Motion

Core Motion collects motion and environment related data from the on-device hardware of iOS devices. Devices are equipped with accelerometers, gyroscopes, pedometer, magnetometer, and barometer. Most of the services in the Core Motion framework provides both the raw values recorded by the hardware and a processed version of those values [39]. Processed values do not include sorts of bias that might negatively affect how the data are used [39].

Accelerometer measures changes in velocity along one axis. Every iOS device has a three-axis accelerometer. This accelerometer collects data in each of the three axes. The values delivered by the accelerometers are measured in increments of gravitational acceleration. Acceleration value *1.0* represents an acceleration of 9.8 meters per second in the corresponding direction. Based on the direction of the acceleration, the values may be positive or negative [39]. `CMMotionManager` class provides two techniques of accessing the accelerometer data. The first method is to pull accelerometer data only when the application needs the data. The second approach is to push accelerometer updates to the application at consistent intervals. The maximum frequency at which the application can request updates is dependent on the hardware and is generally at least 100 Hz. When the application requests a frequency that is greater than the frequency supported by the hardware, the framework uses the supported maximum instead [39].

The gyroscope measures the rate at which the device rotates around a spatial axis [39]. Most of the iOS devices have a three-axis gyroscope that collects data in all three axes. The gyroscope measures values in radians per second around the particular axis. The values may be positive or negative, depending on the direction of rotation [39]. `CMMotionManager` provides the same interface for gyroscope as for the accelerometer. Therefore, there are interfaces for pulling and pushing the gyroscope data. The unprocessed data collected from the gyroscope interfaces may be biased by other factors such as device acceleration [39]. In the case the application needs gyroscope values without bias, the documentation recommends to use device-motion interfaces of `CMMotionManager` class instead.

The device-motion service provides an interface to access unbiased motion-related data. The raw values delivered by accelerometer and gyroscope has to be processed to remove bias from other factors such as gravity or device acceleration. The service makes use of all associated hardware and generates a `CMDeviceMotion` object. The `CMDeviceMotion` object contains information about the device's orientation in three-dimensional space relative to the reference frame, unbiased rotation rate, and current gravity vector. Moreover, it includes a vector representing user-generated acceleration without gravity and current magnetic-field vector [39]. The techniques for accessing the device-motion data is similar to the ones of accessing accelerometer and gyroscope data.

The device attitude delivered by the device-motion service is always defined relative to a fixed reference frame. The used reference frame is based on the interfaces used to start the service. Core Motion offers four reference frames – `xArbitraryZVertical`, `xArbitraryCorrectedZVertical`, `xMagneticNorthZVertical` and `xTrueNorthZVertical`. The `xArbitraryZVertical` reference frame describes a frame in which the Z-axis is vertical, and the X-axis points in an arbitrary direction in the horizontal plane. Furthermore, `xArbitraryCorrectedZVertical` describes the same reference frame as `xArbitraryZVertical`. However, in contrast with `xArbitraryZVertical`, when the magnetometer is available and calibrated, delivered values from the magnetometer are used to improve long-term yaw accuracy. As a result, using `xArbitraryCorrectedZVertical` reference frame

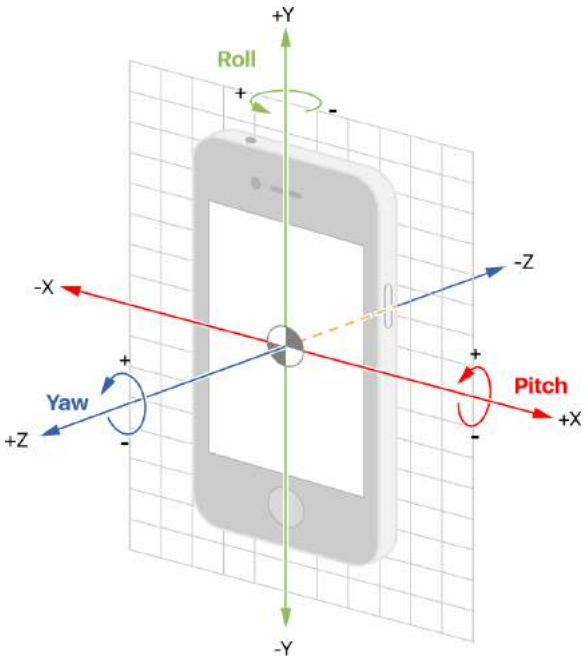

Figure 3.11: The figure shows the direction of the x, y, z-axis and the corresponding Euler angles reported from the Core Motion framework. Source: *Understanding Reference Frames and Device Attitude* [39].

leads to an increased CPU usage. The `xMagneticNorthZVertical` reference frame describes a frame in which the Z-axis is vertical, and the X-axis points toward magnetic north. When using this specific reference frame, the magnetometer may need to be calibrated. Finally, the `xTrueNorthZVertical` reference frame describes a frame in which the Z-axis is vertical, and the X-axis points toward true north. This reference frame may require the magnetometer to be calibrated. Also, the reference frame requires the location to be available in order to calculate the difference between magnetic and true north [39]. Every reference frame assumes a device that is lying on a flat surface and is rotated in a specific direction [39].

The `CMDeviceMotion` has `attitude` property, which contains pitch, roll, and yaw values for the device that are relative to the reference frame. The values correspond to the device's attitude in three-dimensional space. In the case when all three values are equal to 0, the device's attitude matches the orientation of the reference frame. Rotation values are in the range $-\pi$ to π and reflect the amount of rotation in radians around the specified axis. Figure 3.11 shows how pitch, roll, and yaw values are delivered in each of the three axes. Pitch is a rotation around the device's x-axis, the roll is a rotation about the y-axis and yaw corresponds to a rotation along the z-axis.

Chapter 4

Design of the Mobile Application

4.1 Requirements and Design Goals

4.1.1 User Interface

The process of designing a user interface for application for rephotography had to take several constraints into account. First, the main content of the application is user captured photographs. Therefore, the user interface should make the photographs the core part of the interface and should not distract the user with unnecessary colors or UI elements. Second, the photographs are connected to scenes. The scenes are captured at a specific geographical location, and the location must be presented to the user.

Furthermore, the user interface needs to communicate which captured image is the reference and the distance from the user's location to the scene's location. The capture process should be seamless and should guide the user to input all necessary information before the scene gets saved. The user should always have the option to go back and change the input, and the application should show an overview of all inputted data before the object is saved. Lastly, the application is planned to be available on the iOS platform only. Therefore, the user interface should use default system UI elements and follow Apple's Human Interface Guidelines[1].

Figure 4.1 shows two wireframes. The wireframes show the first screen in the application that the user sees after logging in. The *Discover* screen has two modes – Nearby and Map. *Nearby* screen shows a gallery of scenes nearby the user's location. The data are sorted by the distance from the user's location to the scene's location. The second mode (*Map*) shows a map with annotations of nearby scenes. On both screens, when the user taps on a scene, the application will show detail for it. The detail screen is shown in Figure 4.2.

The detail should contain the most crucial information, such as the reference image, scene's location, and an option to rephotograph the same scene. Furthermore, the screen should provide a human-readable representation of the scene's coordinates and a way to navigate to its location. Additionally, the detail screen should show all captured photographs of the scene.

The exploration capability of nearby scenes is the first part of the application core features. The second is capturing new and rephotographing existing scenes. The functionality of capturing a new scene should be accessible to the user at most of the times when using the application. A common solution to this on iOS platform is to use *tab bar* element. The

[1] https://developer.apple.com/design/human-interface-guidelines/ios/overview/interface-essentials/

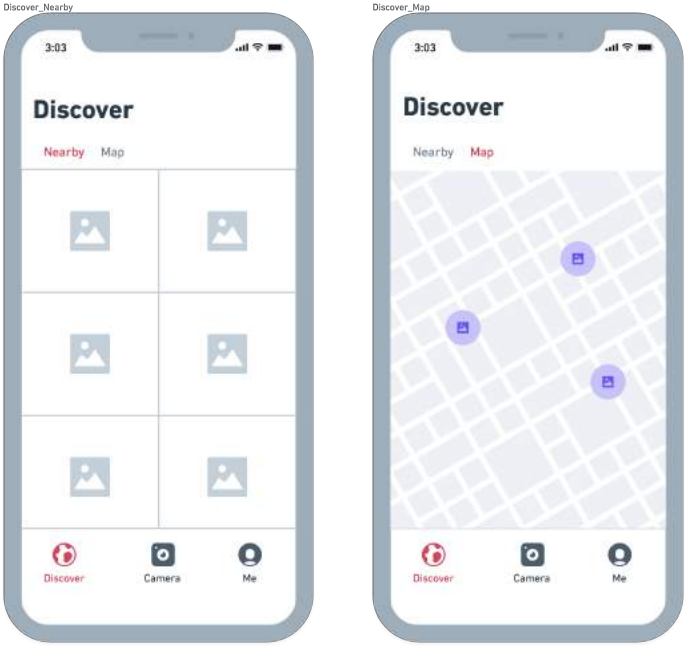

Figure 4.1: The *Discover* screen displays data of nearby scenes in two modes – as a grid of images (*Nearby*) or on a map (*Map*).

tab bar is a view at the bottom of the screen serving as an access point to different parts of the application. The application's tab bar in 4.1 consists of *Discover*, *Camera* and *Me* icons. Each of them navigates the user to the corresponding screen. The *Camera* screen starts a new capture session and immediately shows the direct feed from the device's camera. The interface of the camera is designed as a modal – it is presented over the whole screen. To dismiss the presented modal screen, the user can tap on a cancel button in the top left corner. The photograph is captured by tapping on a virtual shutter button. After the user captures the image, the application displays the captured photograph to the user for a review. After the user confirms the new scene's image, the application asks for a feet shot. When the user captures the place where they stand and confirms the image, the application shows a map with a pin to specify the location on the map. By using a drag and drop the user specifies the precise location using the pin. Finally, after the location is confirmed, the application shows an overview containing the captured image, the location, feet shot, and the user can optionally provide a descriptive name for the scene. Figure 4.3 shows the complete flow of capturing a new scene.

4.1.2 Mobile Application Architecture

The mobile application for iOS is based on *Model-View-Controller* (*MVC*) architecture. The MVC is a design pattern that divides objects into three roles based on their responsibilities. The roles are *model*, *view* and *controller*. Furthermore, the pattern defines the way how those objects communicate with each other. Model objects represent the data specific to an application and provide the business logic that manipulates and process that data. A view is an object that the user can see; it knows how to draw itself on the screen and can respond to user actions. Controllers are objects that act as a mediator between one or more

of an application's view objects and one or more of its model objects [32]. Figure 4.4 shows the communication between the respective roles in MVC.

In the case that a model object changes, the object notifies a controller which updates the corresponding view objects and the controller interprets user actions in the view objects that create or modify data and updates the model or creates a new model object.

Moreover, the application uses the *Target-Action* pattern to handle manipulations with control objects such as buttons and sliders[2]. Furthermore, one of the most used design patterns in the application is *Delegation* as most of the APIs used for the core functionality of the application using this pattern. Additionally, the delegation is used to make one object act on behalf of, or in coordination with, another object[3]. The application leverages *Dependency Injection* design pattern to pass dependencies to other objects through segues or its initializers. Finally, the application makes use of the *Singleton* pattern in a few cases. Even when the Singleton pattern is used in *Cocoa* framework, the application uses this pattern in sporadic cases where it is desired to have a single point of control, e.g., `DeviceLocationService` class that creates `CLLocationManager` and handles its delegate methods.

The application uses container view controllers to implement some parts of the user interface. Furthermore, this concept is used to prevent the *Massive View Controller* problem where one view controller has too many responsibilities. The *Discover* screen (Figure 4.1) displays grid of nearby scenes or shows them on map. The user can select the *Nearby* and *Map* mode through a `UISegmentedControl` inside a `UIToolbar` which is attached to the top of the screen. Figure 4.5 illustrates how a container view controller is used to distribute responsibilities between several view controllers. The `CameraViewController` manages the camera and handles user interactions such as tapping on a shutter button. In its view, the `CameraViewController` embeds a `CameraOverlayViewController` inside the container view. The `CameraOverlayViewController` is responsible for drawing overlay content over the camera feed such as the grid, navigation rectangle, or displaying the overlay of the reference image. This way, each view controller has its responsibilities. The communication between those separate view controllers can be accomplished by delegation.

4.1.3 Designing API Service

The application allows users to browse scenes captured nearby their location and upload new scenes and photographs. The server with the application interface has to provide the functionality to handle requests with multimedia data and build responses based on specified parameters. The chosen server's architecture style is *REST* (*Representational State Transfer*). The application's REST web service leverages and is build on top of HTTP methods to perform CRUD operations on a resource. The resource in the context of the application is a Scene, User, Photo, and others. The service uses `GET`, `POST`, `PUT`, `DELETE` and `PATCH` HTTP methods to perform corresponding action with the resource. The API service responds to the client with an HTTP header and a response data in JSON format [4].

Moreover, the client serializes the data for upload into JSON and sends them in a body of the request. When designing a REST API with data in JSON format, there are two

[2] https://developer.apple.com/library/archive/documentation/General/Conceptual/Devpedia-CocoaApp/TargetAction.html

[3] https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/Delegation.html

conventional approaches in regards to transferring images or other binary data. The first approach is to encode the data into the JSON amongst the other transferred data. The transfer of image data inside a JSON serialized data can be accomplished by encoding binary data and representing them in `Base64`. The result of the encoding process is a sequence of ASCII characters which can be sent as a string in JSON. However, the output bytes to input bytes ratio is 4:3 yielding a total of 33% overhead on the transferred data [34]. The overhead can be mitigated in some cases with a compression algorithm, but the use of `Base64` encoding to send images from a server to the mobile application rises additional concerns[4]. The second approach is to build HTTP request with a `multipart` content-type [47]. In this case, a specific *boundary* string divides the whole body of the request into one or more body parts [47]. In the scenario of uploading a new scene, the application would send three distinctive body parts – JSON formatted metadata and binary data for the captured image and feet shot. The application uses the second approach to mitigate the problems of using `Base64` encoding for sending images between a server and a mobile client.

The API should provide public API endpoints for user registration and login or log out. However, operations such as creating a new scene or adding a photograph should require user authentication. The REST architecture style is stateless, and thus the authentication cannot be implemented by traditional stateful authentication such as by using sessions or cookies. A standard solution for a stateless authentication is a token authentication. Whenever the user registers a new account or logs in, the server responds with an authentication token which the client safely saves. Furthermore, the client's application embeds the authentication token into the HTTP header, and the server can authenticate the user with each request.

The API service should save the model data into a database, but the binary image data should not be saved into the SQL database. The reasons behind this are discussed more in-depth in section 3.2.2. Moreover, the saving process to the several data storages should be atomic, and when a part of the transaction fails, the transaction should rollback all modified data and revert the database to the previous state.

4.2 Possible Solutions

4.2.1 Various methods of camera pose estimation

Relative camera pose estimation is a process of accurately estimating the location and orientation of the camera concerning another's camera's reference system. The task of camera pose estimation in the papers discussed in sections 2.4.1 and 2.14 is accomplished by using a SIFT or ORB feature-detector for extracting key-points, computing correspondences between those key-points and then estimating the essential or homography matrix. In the case of estimating the essential matrix, the solution uses a 5-points algorithm. In both cases, the RANSAC algorithm is used to detect inliers for robust matching. Finally, the discussed systems compute a translation or navigation vector and visualize navigation elements guiding the user to the desired viewpoint. The drawbacks of this feature-based camera pose estimation are that the estimation depends in no small extent on the correspondence assignment. In a case of textureless objects there are too few correspondences. On the other hand, in the case of objects with repetitive texture or considerable viewpoint change, there are too many noisy correspondences [55].

[4] https://medium.com/snapp-mobile/dont-use-base64-encoded-images-on-mobile-13ddeac89d7c

Lately, researches apply convolutional neural networks (CNNs) to solve the task of camera pose estimation [55, 59, 63]. The [59] proposes an end-to-end CNN-based method for absolute 6-DoF[5] camera pose estimation from a single RGB image. The proposed network is robust to difficult lighting, motion blur, and different camera intrinsics [59]. Unlike the [59], the [63] focuses on the problem of relative camera pose estimation. The relative pose estimation provides methods for relation and representation learning for previously unseen scenes and objects [63]. The method produces a translation vector up to scale. The comparison of the best model from [63] with feature-based methods – SURF and ORB, confirms that the proposed model performs better than the baseline feature based methods [63]. Lastly, the [55] in comparison with [63] recovers a full translation vector. As for the comparison, the [55] compares the model with SURF. The results are compared with two implementations of SURF – *SURFSmall* and *SURFFull*. The versions differ in the resolution of the image. SURFSmall uses scaled images of 256×455 pixels, followed by a center-crop to 224×224 pixels. The SURFFull uses the original images without downsampling. In the case of the SURFSmall, the RPNet family reduces the error from 5% to 70% on both translation and rotation in most of the cases. SURFSmall slightly outperforms the RPNet-based methods on one dataset (KingsCollege). SURFFull significantly boosts the performance of traditional key-point based methods. However, the RPNetFC outperforms the traditional approach on OldHospital and ShopFacade. The SURFFull performs a little better on KingsCollege and StMarysChurch datasets. Further, The RPNetFC significantly outperforms SURFFull in cases when the images contain large viewpoint changes [55]. In conclusion, the system described in [55] produces competitive or better results over the traditional feature-based methods [55].

4.2.2 Types of device attitude representation

Core Motion framework, discussed in section 3.3.2, collects data from on-device the hardware and generates objects of type `CMDeviceMotion`. . The `CMDeviceMotion` objects contain `attitude` property. The property is of type `CMDeviceAttitude` and represents the device's orientation to a known frame of reference at a point in time [39].

The `CMDeviceAttitude` offers three different mathematical representations of the attitude - a rotation matrix, a quaternion, and Euler angles.

Euler angles representation of the device's attitude consists of the roll, pitch, and yaw values. The roll, pitch, and yaw values describe rotation in radians about a corresponding axis. Specifically, the pitch is rotation around x-axis, the roll is rotation around y-axis and yaw is rotation around the z-axis. Figure 3.11 shows the projection of roll, pitch, and yaw onto the axes.

Any rotation can be described by a sequence of three coordinate rotations [52]. Let the first rotation is an angle ψ about the k-axis, the second rotation is an angle θ about the j-axis, and the third rotation is an angle ϕ about the i-axis. Euler angle vector that arranges the angles in a three-dimensional vector is defined in [52] by:

$$\mathbf{u} := [\phi, \theta, \psi]^T. \tag{4.1}$$

The function that maps a Euler angle vector to its corresponding matrix $R_{ijk} : \mathbb{R} \rightarrow SO(3)$ is defined in [52] as:

$$R_{ijk}(\phi, \theta, \psi) := R_i(\phi) R_j(\theta) R_k(\psi) \tag{4.2}$$

[5] Six degrees of freedom

There are 27 possible rotation sequences of three integers in $\{1, 2, 3\}$ but only 12 satisfy the constraint that no two consecutive numbers in a correct sequence may be equal [52]. These Euler sequences are:

$$(i, j, k) \in \left\{ \begin{array}{cccc} (1,2,1), & (1,2,3), & (1,3,1), & (1,3,2), \\ (2,1,2), & (2,1,3), & (2,3,1), & (2,3,2), \\ (3,1,2), & (3,1,3), & (3,2,1), & (3,2,3) \end{array} \right\} \tag{4.3}$$

Let us assume a Euler Angle sequence $(2, 1, 3)$, then:

$$R_{213}(\phi, \theta, \psi) = R_2(\phi) R_1(\theta) R_3(\psi). \tag{4.4}$$

Rotational matrix is an orthogonal matrix whose multiplication with a vector rotates the vector while preserving its length [52]. The special group of 3×3 rotation matrices is denoted by $SO(3)$. Furthermore, if $R \in SO(3)$, then:

$$det\, R = \pm 1,\ R^{-1} = R^T. \tag{4.5}$$

The elements of a rotation matrix are referenced in [52] as:

$$R = \left[\begin{array}{ccc} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 \end{array} \right] \tag{4.6}$$

$$= \left[\begin{array}{ccc} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{array} \right] \tag{4.7}$$

Rotation about a single coordinate axis is known as a *coordinate rotation* [52]. The corresponding coordinate rotations for the x, y, z-axes are:

$$R_x(\alpha) = \left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin(\alpha) & \cos(\alpha) \end{array} \right] \tag{4.8}$$

$$R_y(\alpha) = \left[\begin{array}{ccc} \cos(\alpha) & 0 & -\sin(\alpha) \\ 0 & 1 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) \end{array} \right] \tag{4.9}$$

$$R_z(\alpha) = \left[\begin{array}{ccc} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{array} \right]. \tag{4.10}$$

Core Motion outputs a direction cosine matrix (DCM) [39]. The elements of the direction cosine matrix are cosines of the unsigned angles between the body-fixed axes and the world axes [52].

Let us consider the Euler Angle sequence $(2, 1, 3)$. The corresponding rotation matrix is defined in [52] as

$$R_{213}(\phi, \theta, \psi) = \left[\begin{array}{ccc} \cos(\phi)\cos(\psi) - \sin(\phi)\sin(\theta)\sin(\psi) & \cos(\phi)\sin(\psi) + \sin(\phi)\sin(\theta)\cos(\psi) & -\cos(\theta)\sin(\phi) \\ -\cos(\theta)\sin(\psi) & \cos(\theta)\cos(\psi) & \sin(\theta) \\ \sin(\phi)\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\psi) & \sin(\phi)\sin(\psi) - \cos(\phi)\sin(\theta)\cos(\psi) & \cos(\phi)\cos(\theta) \end{array} \right]. \tag{4.11}$$

Quaternions were discovered on 16 October 1843 by Irish mathematician William Rowan Hamilton. A quaternion can be expressed as:

$$q = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}. \tag{4.12}$$

The quaternion described in 4.12 has one real dimension and three imaginary dimensions [24]. The a in the equation 4.12 is the real (scalar) part of the quaternion while the $b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$ is the vector (imaginary) part. Imaginary parts i, j, k have a unit value of the square root of -1. However, the parts are all mutually perpendicular to each other and have different square roots of -1. Furthermore, a quaternion $\mathbf{q}$ can be represented as a vector:

$$\mathbf{q} = [q_0, q_1, q_2, q_3]^T = \begin{bmatrix} q_0 \\ \mathbf{q}_{1:3} \end{bmatrix}. \tag{4.13}$$

The norm of a quaternion is defined in [52] as:

$$||\mathbf{q}|| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}. \tag{4.14}$$

A unit quaternion is a quaternion with unity norm [52]:

$$||q|| = 1, \tag{4.15}$$

and is produced by dividing a non-zero quaternion q by its norm [24],

$$\mathbf{u} = \frac{q}{||q||}. \tag{4.16}$$

Let us consider the Euler Angle sequence $(2, 1, 3)$ again, the conversion formula from quaternion to Euler angles for this sequence is defined in [52] as

$$\mathbf{u}_{213}(R_q(\mathbf{q})) = \begin{bmatrix} \text{atan2}(-2q_1q_3 + 2q_0q_2, \\ \qquad q_3^2 - q_2^2 - q_1^2 + q_0^2) \\ \text{asin}(2q_2q_3 + 2q_0q_1) \\ \text{atan2}(-2q_1q_2 + 2q_0q_3, \\ \qquad q_2^2 - q_3^2 + q_0^2 - q_1^2) \end{bmatrix}. \tag{4.17}$$

Quaternions offer some advantages over the matrices. They are easier to normalize than matrices. Normalization cancels out a build-up of small rounding errors. Interpolation between quaternions is more straightforward by using *SLERP*[6] than interpolating between rotation matrices. Lastly, the quaternion representation is more compact as it requires only four scalars in comparison with a matrix which requires nine scalars.

However, matrices offer a more natural way to transform a point by multiplying a vector by a matrix instead of a sandwich form needed for quaternions. Moreover, there are more available libraries for matrices than quaternions [35].

All of the mentioned representations – Euler angles, rotation matrices, quaternions, are convertible to one another. However, Euler angles have singularities which originate from *gimbal lock*. Gimbal lock results from the indistinguishability of changes in the first and third Euler angles when the second angle is at some critical value [52]. Mathematically, the gimbal lock is represented as a singularity at which specific expressions are undefined [52].

[6] Spherical linear interpolation

As the place of singularity depends on the type of Euler angles, it is common to change the representation when an object approaches a singularity to avoid the gimbal lock. Moreover, the problem of gimbal lock can be entirely avoided by using unit quaternions to represent the device's attitude [52].

4.3 Proposed Solution

4.3.1 Using on device accelerometer and gyroscope sensors for rephotography

The application prefers a different approach than the rephotography systems described in chapter 2. Rather than estimating a camera pose through the feature-based methods, the application leverages the use of the on-device hardware sensors to enhance the rephotography process. The application uses the Core Motion framework to get device attitude data. The device attitude is collected from the `CMDeviceMotion` object. Core Motion merges the data from different sensors. Figure 4.7 shows the process of *sensor fusion*.

The `CMDeviceMotion` encapsulates measurements of the attitude, rotation rate and acceleration of a device. The `CMAttitude` provides three mathematical representations of the attitude – Euler angles (roll, pitch and yaw values), a rotation matrix and a quaternion. The application saves all the available information about the device's attitude. The data are saved during the capture process and thus represent the device orientation at the time of capture.

The application saves all the data from `CMDeviceMotion`. However, the most important properties of the *Motion* is the quaternion. The values `quaternionW`, `quaternionX`, `quaternionY`, `quaternionZ` are the corresponding parts of a `CMQuaternion` structure collected from the `CMAttitude` property of `CMDeviceMotion` object. The `CMQuaternion` mathematically represents a unit quaternion defined as

$$q_x * i + q_y * j + q_z * k + q_w,$$

where q_x, q_y, q_z equals to b, c and d in equation 4.12 [39]. The q_w stands for the scalar part and equals to a in the equation 4.12. The i, j and k are imaginary parts as in 4.12.

The unit quaternion q corresponds to a rotation of θ radians about the unit vector $\{x, y, z\}$. The quaternion q satisfies

$$q_x = x * \sin\left(\frac{\theta}{2}\right) \tag{4.18}$$

$$q_y = y * \sin\left(\frac{\theta}{2}\right) \tag{4.19}$$

$$q_z = z * \sin\left(\frac{\theta}{2}\right) \tag{4.20}$$

$$q_w = \cos\left(\frac{\theta}{2}\right). \tag{4.21}$$

The equations 4.18–4.21 are derived from *CMQuaternion* [39]. Furthermore, the quaternion q is used to represent the primary camera (device) attitude. Let q_0 be the quaternion saved at the capture time. Further, let q_1 be the current quaternion collected from the

sensors while recapturing a previously captured scene, then

$$q_{\Delta} = q_0^{-1} * q_1, \tag{4.22}$$

is a relative rotation between the reference and the current device attitude.

At last, the application persists the location data from the Core Location framework. The application uses the scene's coordinates to navigate the user to the correct place. Additionally, the feet shot (image of the place where the user has been standing) helps to locate the exact capture spot.

4.3.2 Navigating user to the desired camera pose with navigation rectangle

Section 4.3.1 describes how the application uses the sensors available on the device to collect the motion data representing the device attitude while the user captures a new scene. When the same scene is being recaptured, the application needs to guide the user to rotate the device to the same position as the reference. The equation 4.22 calculates a quaternion q_{Δ} representing a relative rotation between the reference and current orientation. The relative rotation denotes the rotation that needs to be performed to get from a current to the reference orientation. The application uses a system of a navigation rectangle with a grid to visualize the device pose.

First, let us consider the scenario that the user wants to capture a new scene. The user opens a camera screen in the application and the screen stars showing the feed of data from the camera. Furthermore, the camera shows a grid and a yellow rectangle representing an alignment level. Figure 4.8 shows the scenario in which the device is tilted to the right. The rectangle guides the user to align the device with the static grid. As mentioned before, after the user aligns the device and captures the scene with the shutter button, the application saves the motion data representing the device attitude.

Later, when the user wants to recapture the scene and taps on a *Retake* button on the scene's detail, the application shows a similar screen to the one when capturing a new scene. However, instead of using the rectangle only as a representation of an alignment level, the rectangle now represents the relative rotation. In order to move the rectangle and use it as a visualization of a rotation represented in a unit quaternion, the application has to convert the quaternion to a point on a screen. The point has an x and y coordinates in a range of $[0, \text{WIDTH}]$ for x coordinate and $[0, \text{HEIGHT}]$ for y coordinate, where WIDTH and HEIGHT are constants equal to the maximum width and height of the camera's view. The coordinates are computed as

$$\begin{aligned} x &= \frac{\text{ROLL} + \pi}{2\pi} * \text{WIDTH} \\ y &= \frac{\text{PITCH} + \frac{\pi}{2}}{\pi} * \text{HEIGHT}, \end{aligned}$$

where ROLL and PITCH are Euler angles derived from the quaternion q_{Δ}. The conversion between the quaternion and Euler angles is done by using the equation 4.17. Specifically, the ROLL and PITCH values are calculated as

$$\text{ROLL} = \text{atan2}\left(-2 * q_x * q_z + 2 * q_w * q_y, q_z^2 - q_y^2 - q_x^2 + q_w^2\right) \tag{4.23}$$

$$\text{PITCH} = \text{asin}\left(2 * q_y * q_z + 2 * q_w * q_x\right). \tag{4.24}$$

The rectangle's position is then set to the point (x, y). Whenever the device rotates around x-axis (pitch), the rectangle moves up and down. At the same time, when the device rotates around y-axis (roll), the rectangle moves left and right. Figure 5.3 shows the navigation rectangle guiding the user when retaking an existing scene.

4.3.3 Designing user interface for rephotography applications

The section 4.1.1 contains description of the essential application's functions and Figures 4.1, 4.2 and 4.3 shows the wireframes of several application's screens. The principles discussed in section 4.1.1 apply to other types of applications with images as their primary content, such as social networks and photo editing applications. This section focuses solely on features designed to help users with the rephotography process.

The figure 4.2 shows an image of the scene's detail screen. The *View feet shot* buttons shows an image of the place from which the reference image was taken. The feet shot's purpose is to guide the user to the correct location better and capture the most accurate rephotograph of the reference image. If the environment contains distinctive features such as marks on the ground, the feet shot image can precisely pinpoint the location from which the original photograph has been photographed. Additionally, *View timeline* navigates the user to a screen showing images ordered by their capture date thus creating a view of how the scene looks at different times. Lastly, *View on map* shows a reference image and the scene's location at the same time in order to help the user navigate in an environment and spot the object of interest more easily.

The first wireframe of the camera process, shown in Figure 4.3, displays a camera screen with a shutter button and two additional buttons. The *Flip* button animates a flip transition between the feed from the device's camera and a scene's reference image. Flip allows the user to have a quick look at the reference image and find the object of interest in the environment. Furthermore, the *Overlay* button shows a reference image as an overlay on the camera feed. Additionally, the user can change its opacity through a slider. The overlay helps the user to align the device better to match the reference photograph.

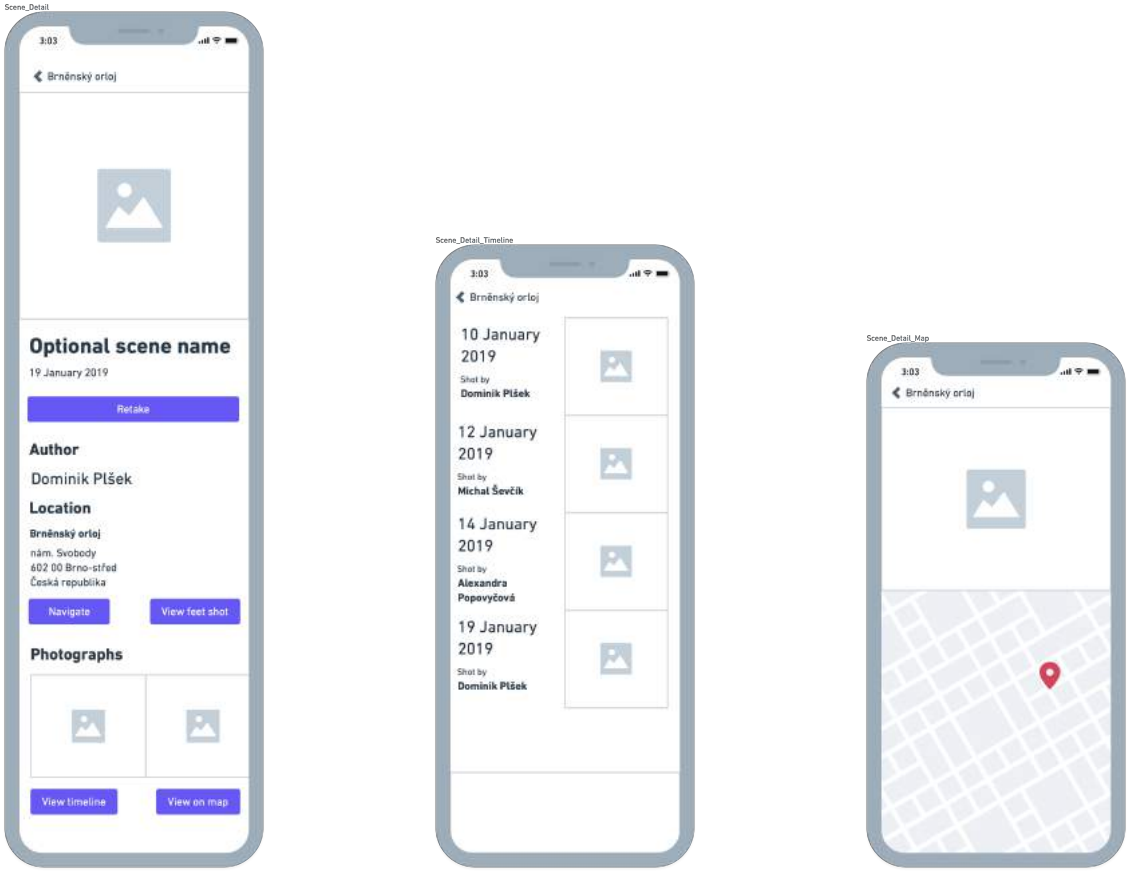

Figure 4.2: The detail screen of a scene. The detail screen shows a reference image in its header with the scene's optional name and capture date. The *Retake* button starts a camera with a navigation rectangle guide system. The location information is obtained from reverse geocoding of the scene's GPS coordinates. Below the location, the *Navigation* button opens the Apple Maps application with the scene's location set as a destination. The *View feet shot* button shows the photo from the exact place where the scene was captured. The horizontal grid of images shows the rephotographs of the scene. Further, *View timeline* shows the timeline of photographs with the corresponding date. Lastly, *View on map* shows a reference image on the top and a map view on the bottom to help the user orientate nearby the scene's location.

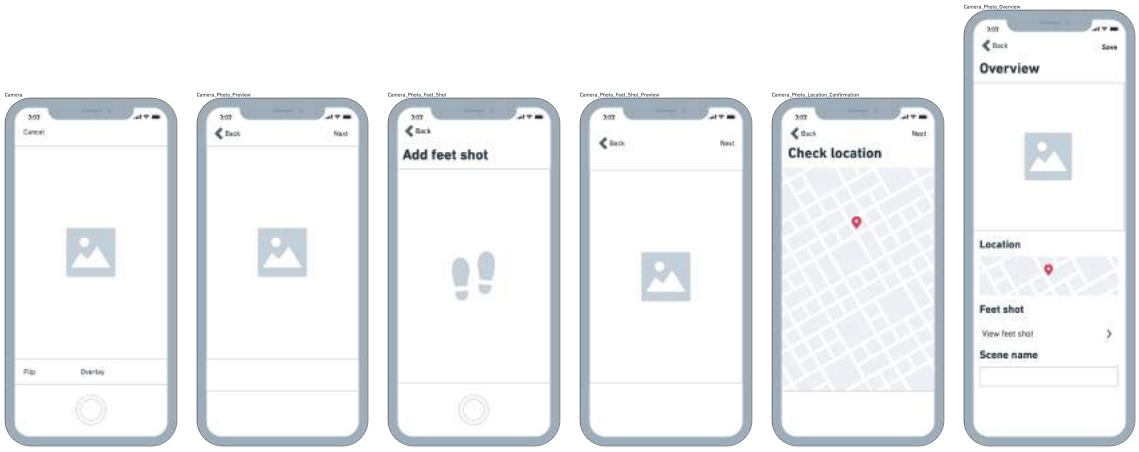

Figure 4.3: The figure shows the flow of capturing a new scene. The first screen is a camera through which the user captures a photograph of a new scene. Furthermore, the application shows a preview of the captured photo for the user to confirm. When the user confirms the photograph, the application lets the user capture a photo of its feet and thus mark the place from which the scene has been captured. Then, the application shows a preview of the feet shot for the user to confirm. After that, the application allows the user to refine the location of the scene on the map through a pin. Lastly, the application shows the user overview of the captured scene.

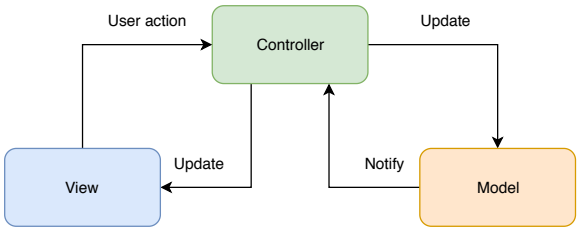

Figure 4.4: The communication between the roles in the Model-View-Architecture on iOS. Figure based on *Model-View-Controller* [32].

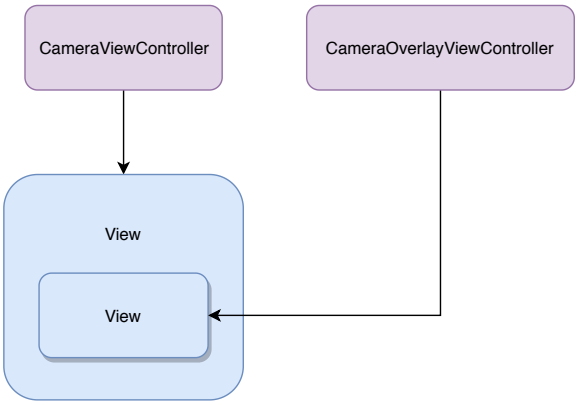

Figure 4.5: The view controller containment is a technique of embedding a view controller inside another view controller's view. The figure shows the application of the view controller containment on one case from the application where `CameraOverlayViewController` is embedded inside a container view in `CameraViewController`'s view. Figure based on *View Controller Containment* [56].

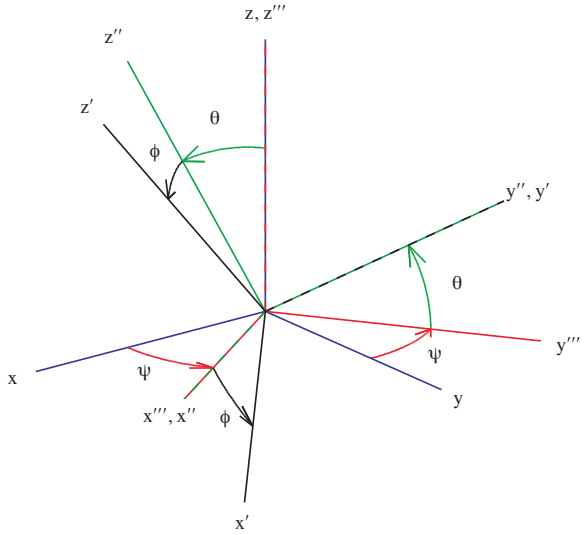

Figure 4.6: Euler Angle Sequence (2, 1, 3). (Source: Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors) [52]

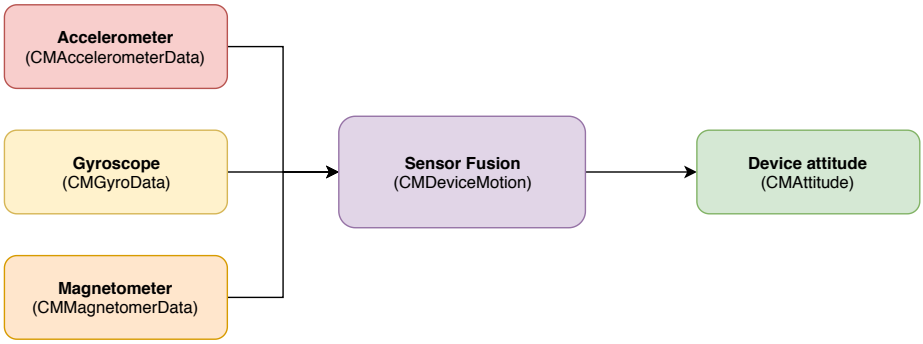

Figure 4.7: The figure shows a sensor fusion of data from several on-device sensors to one representation of the device attitude. Figure based on *Your Phone has Attitude!* [43].

Figure 4.8: Screenshot from the application showing a camera screen with a grid and a rectangle representing the device's alignment level. The rectangle helps the user align the device with the static grid.

Chapter 5

Implementation

5.1 Implementing User Interface

The chapter 4 contains a description and wireframes of the envisioned application. The application uses several technologies from the Foundation framework. Namely, it uses *AVFoundation* for a custom camera implementation, *Core Location* and *Core Motion* for collecting location and motion data, and *MapKit* to show map with annotations of nearby scenes. While *Core Location* and *Core Motion* deliver only data, *MapKit* and *AVFoundation* require the use of dedicated views and layers to integrate into an application.

Most of the application's user interface is built by using Interface Builder. The *Interface Builder* tool is integrated into Xcode and offers functionality to build the scaffolds of the application's user interface visually. As the application makes the most use of user interface elements from the standard library, the tool helps to visualize the interface of the application before building and running the application in the simulator or on the device. The individual screens are represented as view controllers with views inside a *Storyboard*. A storyboard is an encapsulation of the design-time view controller graph represented in an Interface Builder storyboard resource file[1]. The individual storyboard scenes are connected through *segues*. *Segue* is an object that prepares and performs the visual transition from one view controller to another[2]. The scene with a view controller can be located in a Storyboard file with other view controller scenes or refactored to a dedicated Storyboard file. Furthermore, the segue can connect a scene and a reference to a scene. Figure 5.1 displays a document outline with a view controller's view hierarchy and the corresponding view controller scene.

The fundamental building blocks of the application's user interface are `UITableView` and `UICollectionView`. The `UICollectionView` is a class that offers a customizable way to present ordered collection of data. The property `collectionViewLayout` on `UICollectionView` provides a customization point to provide the layout used to organize the items. Both the Discover screen in its *Nearby* mode (Figure 4.1, left) and *Profile* screen use the `UICollectionView` to display a grid layout of images. The grid is accomplished by using the `UICollectionViewFlowLayout` with the cell's width and height equal to the half of the view's width. The `UITableView` is used to build the *Scene Detail* (Figure 4.2, left), *Timeline* (Figure 4.2, middle), and *Scene Overview* (Figure 4.3, last) screens. Furthermore, the *Scene Detail* and *Scene Overview* support drill-down navigation. The *Scene Detail* screen's `UITableView` contains a `UICollectionView` in

[1] https://developer.apple.com/documentation/uikit/uistoryboard
[2] https://developer.apple.com/documentation/uikit/uistoryboardsegue

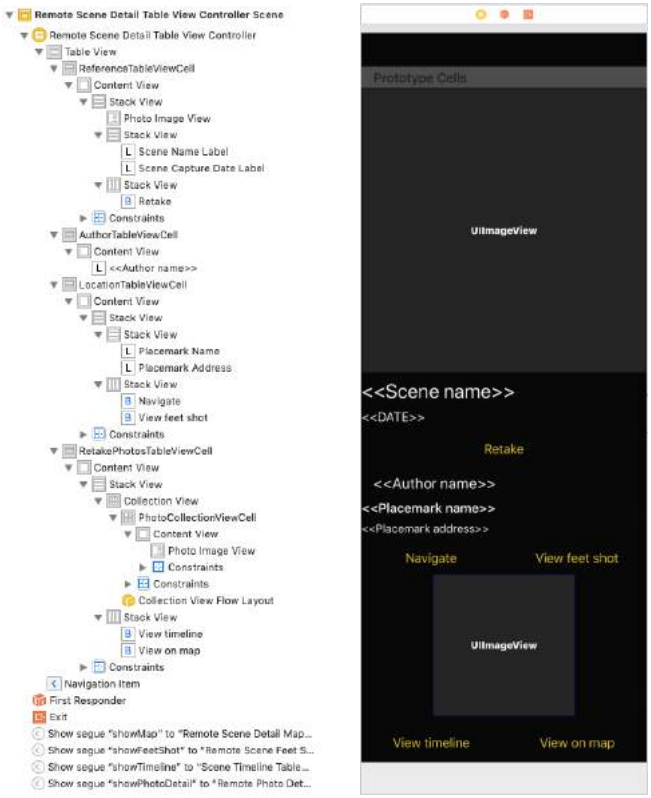

Figure 5.1: The Figure contains screenshots from Xcode that shows view hierarchy (left) and Storyboard scene (right) of the *Scene Detail*.

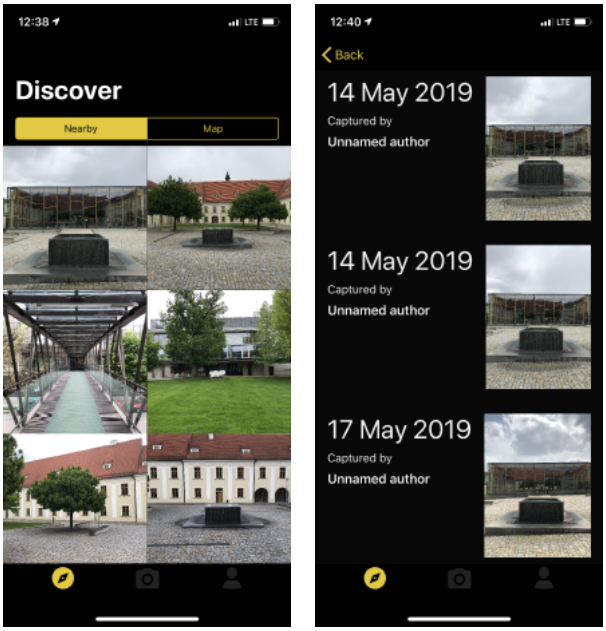

Figure 5.2: The first image in the Figure depicts a *Nearby* mode that shows scenes around the user's location. The second image shows a timeline of a scene.

its cell. The collection view is added as a subview of one of the `UITableview`'s cells and configured in `tableView:willDisplayCell:forRowAtIndexPath:` delegate method of `UITableViewDelegate`. Furthermore, the scroll direction is horizontal instead of vertical as it is in the case of *Nearby* and *Profile* `UICollectionViews`.

The camera scene contains a `CameraViewController` responsible for controlling the device's camera and two container view controllers dedicated for managing the camera overlay and control panel. The `AVCaptureVideoPreviewLayer` added as a sublayer to the camera view controller's view displays the feed from the device's camera as it is being captured.

The camera overlay is responsible for drawing a grid, an overlay reference image, a slider for changing the opacity of the reference image overlay and the navigation rectangle. The grid view is drawn as lines between pairs of `CGPoints`. The class `GridView` uses the `@IBDesignable`, and `@IBInspectable` attributes and thus the grid gets rendered in the Interface Builder and the grid color is configurable in Interface Builder's Attributes Inspector. The overlay of a reference image on top of the camera view is done by changing the opacity of a `UIImageView` through the `UISlider`. Additionally, to make the `UISlider` vertical and not horizontal, a transformation matrix with a rotation angle $-\frac{\pi}{2}$ is applied. Lastly, the navigation rectangle is drawn as `CALayer`. The process of moving the rectangle based on the device motion is described in section 4.3.2. In the case of capturing a new scene, the rectangle rotates based on the yaw angle of the device. The yaw is computed from the quaternion collected from *Core Motion*. Finally, a rotation matrix applied to the `CALayer`'s transform matrix rotates the rectangle around the z-axis by the angle equal to the yaw.

The control panel shows the *Flip* and *Overlay* button when recapturing an existing scene. The overlay image with the slider and the navigation rectangle is drawn only in the mode when the user is recapturing an existing scene. Furthermore, the flip animation is

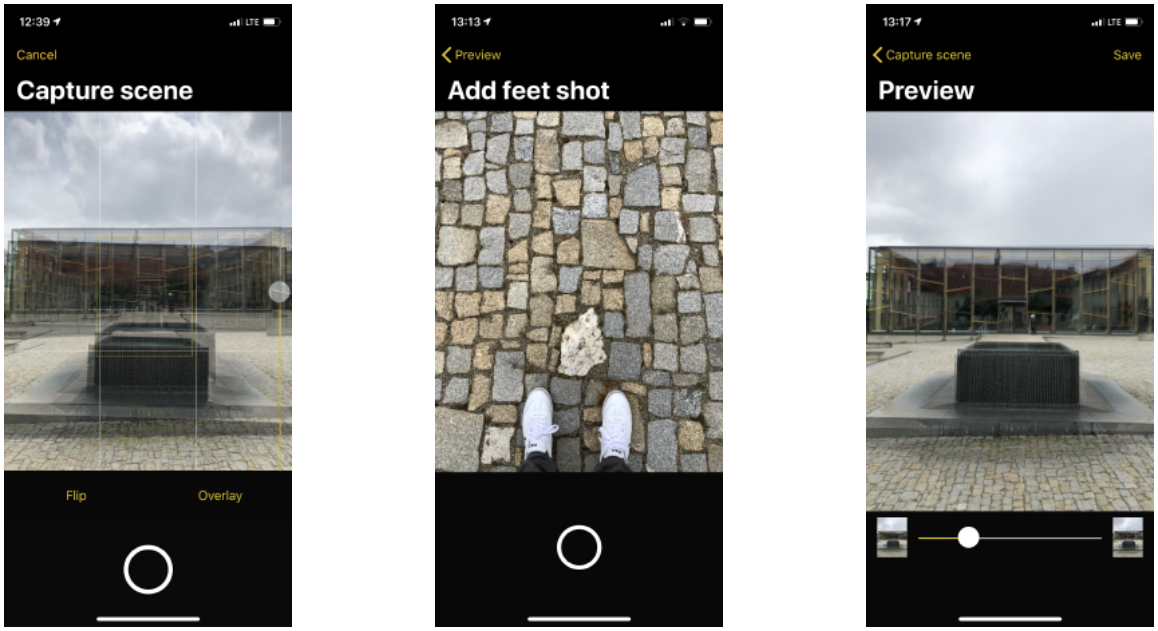

Figure 5.3: The first image displays the *Retake* mode with an activated overlay and navigation rectangle. The middle image shows the feet shot capture in process of creating a new scene. The last image depicts a *Preview* view that is available after recapturing an existing scene. The slider changes the opacity of the corresponding image and the screen shows the differences between the reference and newly captured photograph.

done by using transition animation on *UIView* class. The animation effectively hides one view while setting the other view's changing its `hidden` property to true and opacity to value 1.0, making the second view visible on a screen.

The map scene contains a view controller with a `MKMapView` as its subview. The map type is set to `Standard` and the difference between the other map types is discussed more in detail in section 3.1.3. The annotations are customized `MKMarkerAnnotationViews` with the scene's reference image and a camera icon set as its glyph image. The `MapKit` does not allow to change the appearance of the `MKMapView` yet on iOS. It is possible to show a `MKMapView` with a dark appearance on iOS by using Apple's private API, which is the case of this application. As a result, the application would not meet requirements to be accepted to the Apple's App Store. Other solution would be to use a different framework for showing maps such as *Mapbox* framework which offers a lot more customization than MapKit. However, as MapKit is a native framework, the application does not need to have third-party dependencies, and the integration of MapKit into an application is seamless.

5.2 Developing Application Features

The section 4.1.1 defines the functionality and user interface of the application. The application shows the users scenes nearby their geographical location in two modes – grid view and map view. Furthermore, the user can capture a new scene through the application, and the capturing flow is illustrated in Figure 4.3. The application connects to the web service that serves as API. The application uses the API to upload new scenes, download scenes nearby the user, and create new user accounts. As a result of this, the application uses

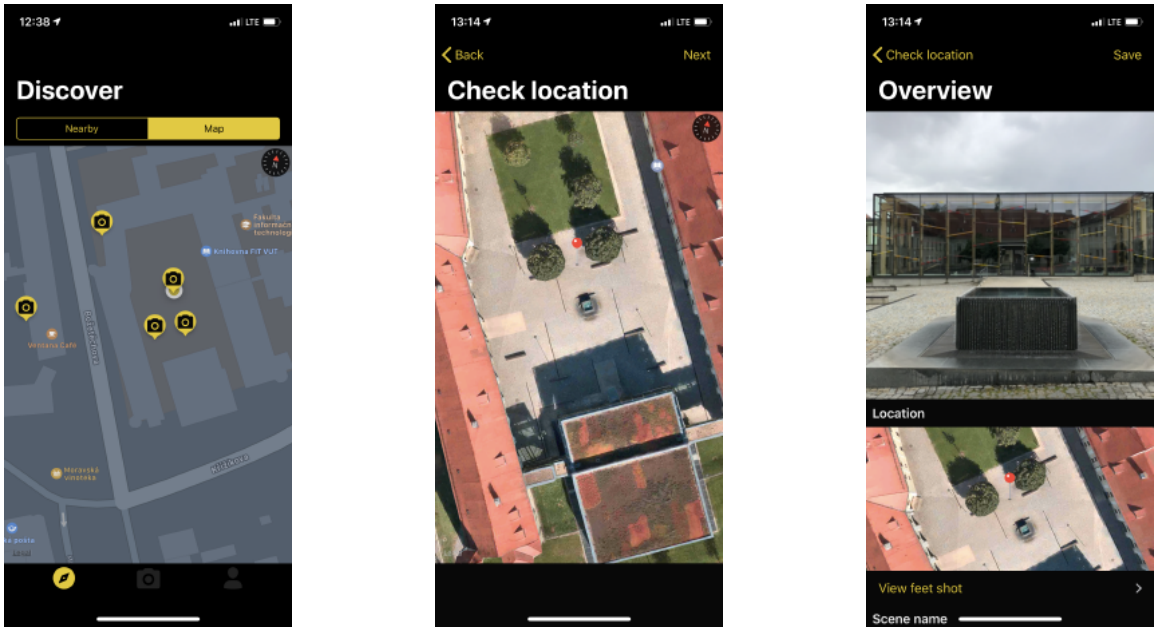

Figure 5.4: The Figure illustrates the *Map* mode (left) that displays nearby scenes on the map. Furthermore, the middle image shows the location confirmation screen in the process of capturing a new scene. The user can specify the exact location by drag and dropping the pin. Lastly, the *Scene Overview* contains the captured image, snapshot of the map and additional scene data such as feet shot.

many technologies available on iOS such as Core Location, Core Motion, AVFoundation, and Core Data.

The `CameraViewController` is responsible for managing the interaction with the device camera. It holds a strong reference to `CaptureSessionController` which sets up and controls `AVCaptureSession` from `AVFoundation` framework. The `CaptureSessionController` is class that sets up which on-device camera will be used and configures the capture device. Moreover, it sets up the `AVCaptureVideoPreviewLayer` to show the feed from the camera in a view on screen. Additionally, it exposes `capturePhoto:withSettings:delegate` method that the `CameraViewController` uses to capture a photo. The `CameraViewController` retrieves the `videoPreviewLayer` from the `CaptureSessionController` object, sets its `frame` to the camera view's `bounds` and adds it as a `sublayer` to the view. Furthermore, the `CameraViewController` calls `configureCaptureSession` method on the `CaptureSessionController` and updates the control panel based on the `capturingState` which can be equal to `newSceneCapture` in the case that the user taps on the camera button from the tab bar or `retakeCapture` when the user taps on *Retake* button on existing scene's detail screen. This configuration is done in `viewDidLoad` method of the controller. In the `viewWillAppear` method the controller calls the `startCaptureSession` method on `CaptureSessionController` object which starts the flow of data from the configured inputs to the outputs connected to the `AVCaptureSession` instance. The view controller's `viewDidAppear` method contains a code that sets up the motion services and starts accelerometer updates. These updates are then used to determine the orientation of the device, e.g., `landscape` or `portrait` orientation. When the user taps on the shutter button, through the *Target-Action* pattern, the controller's `didTapOnShutterButton`

method is called. This method calls controller's `capturePhoto` function that creates `AVCapturePhotoSettings` object. The method creates a new instance of `PhotoCaptureProcessor` and through *Dependency Injection* pattern it passes the `photoSettings` through the initializer. The controller adds this object to an array of `PhotoCaptureProcessor` objects to keep a strong reference and avoid deinitialization of this object. Finally, the `capturePhoto:withSettings:delegate` function on `CaptureSessionController` instance is called.

The `PhotoCaptureProcessor` implements `AVCapturePhotoDelegate` methods. Figure 3.10 shows the process of capturing a photo through *AVFoundation* framework and the corresponding delegate methods. In the `willBeginCapture...` function the processor fetches current location and motion data. In the `didFinishProcessing...` function, the processor retrieves the photo and adds GPS metadata to its file data representation. Moreover, it keeps reference to the image data, metadata, and preview pixel buffer. Finally, the `didFinishCapture...` method creates a custom `CaptureSessionData` and passes it through a `captureDidFinish` delegate method back to the `CameraViewController`.

The `CameraOverlayViewController` is a view controller embedded inside a container view in `CameraViewController`'s view. The process of drawing a navigation rectangle, grid and animation of the flip and overlay buttons is described in sections 4.3.2 and 5.1.

The `DiscoverViewController` manages two embedded view controllers – `NearbyCollectionViewController` and `MapViewController`. Based on the state of the `UISegmentedControl`, the appropriate view controller is shown on the screen. The controller listens to notifications from the `DeviceLocationService` class, and when user location gets updated, it fetches the location's coordinates. Furthermore, it loads scenes from the remote web server through `APIClient` class. The model objects are initialized from the decoded data from the API. The JSON from the web service contains URLs for the photos. Therefore, for each scene, a feet shot image has to be downloaded, and for all photos that belong to the scene, the image data has to be downloaded. The application uses `DispatchGroup` API to synchronize these asynchronous network tasks. After the network calls finish, the data are persisted in-memory effectively, creating a cached object graph backed by Core Data.

The `MapViewController` uses the `NSManagedObjectContext` that is responsible for manipulating and tracking the view of the in-memory persistent store. The controller fetches scenes from the context and creates annotation views for each scene. Further, the controller implements `NSFetchedResultsControllerDelegate`; thus, the annotations are dynamically added, updated, or removed when the model changes.

5.3 Programming and Deploying API

The application allows a user to upload captured scenes and display scenes captured by other users. The requirements for this sharing and viewing functionality are described in section 4.1.3. The API is designed as a REST API that communicates with a database and serializes the data into a JSON. The data are then sent to the client (mobile application) with a corresponding status code in an HTTP header.

The application server is programmed in Ruby on Rails framework. Rails is a web application development framework written in Ruby language[3]. It is a *Model-View-Controller* (*MVC*) framework, and Rails enforces a structure for the application. In *MVC* architec-

[3] https://guides.rubyonrails.org/getting_started.html

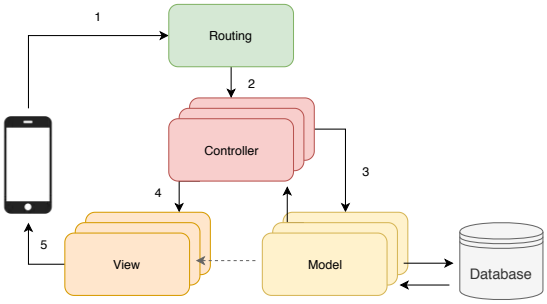

Figure 5.5: The Model-View-Controller architecture of Ruby On Rails application. Source: *Agile Web Development with Rails 4* [67].

ture, a model is responsible for maintaining the state of the application. In some cases the state is temporary in others it is persisted outside of the application, e.g., in database.

Furthermore, the view is responsible for generating a user interface. Most often, the user interface is based on data in the model. However, the view does need to be a user interface per se, but in specific cases such as in an API service, the view is, in fact, the model data serialized in JSON. Moreover, even when the view may present various inputs for the user, it never handles incoming data. The view's responsibilities are finished at the moment the data is displayed. The controller is in charge of the application. Controllers receive events from the clients, interact with the model, and display views [67].

Figure 5.5 shows the process of handling an incoming request to Rails application. Let us assume that the client posts new scene from the device to `http://localhost:3000/api/v1/scenes/`. The routing component receives the incoming request and parses it. The request contains a HTTP method `POST` and a path `/api/v1/scenes`. The path `/api/v1/` denotes a nested namespace consisting of `api` and `v1` namespaces. The reason behind using a namespace is to organize group of controllers[4]. Additionally, the `v1` part indicates the version of the API. As a result, the API service can change but still maintain a backwards compatibility with its clients. The part `scenes` corresponds to controller class `ScenesController` and by convention, `POST` methods are associated with `create()` actions [67].

The `create()` method in `ScenesController` handles the request. First, it parses the parameters sent with the request, checks if the request's header contains `multipart/form-data` as its content type, parses JSON metadata and the data for scene's image and feet shot. Additionally, the actions on `ScenesController` require the user to be logged-in. The authentication is handled in `ApiController`. The controller uses Rails function `authenticate_with_http_token` to retrieve the token from the HTTP header, the function finds the user in the database based on the token and securely compares the `SHA256` digest of the sent token and user's token. The `ScenesController` prepares a hash of attributes containing the parsed parameters from the request and current user id. Furthermore, it creates and calls a new scene `Create` transaction and passes it the attributes and a block which gets executed later in the transaction. The Rails application uses `dry-transaction` gem. The gem provides a way to define complex business transactions that includes processing over many steps and by many different objects. It takes a `Railway Oriented Programming`[5]

[4] `https://guides.rubyonrails.org/routing.html`
[5] `http://fsharpforfunandprofit.com/rop/`

approach to capturing and returning errors from any step in the transaction[6]. The transaction uses several steps to create all necessary objects before saving the scene. The steps are wrapped around in a `db_transaction` step which creates `ActiveRecord` transaction that rollbacks all the changes in the database when an error occurs in any part of the transaction. The image data are stored on the disk in the development environment and uploaded into Amazon S3 Bucket[7] in production environment through the `ActiveStorage` API.

In a successful case, the transaction returns an object, which is the newly created scene. Whenever an error occurs, the transaction fails and returns all errors that occurred during the transaction. The `ScenesController` then returns a response with JSON serialization of the object and HTTP status code `201` (`Created`) in a successful case or JSON containing error messages and HTTP status code `422` (`Unprocessable Entity`) in a failure case.

The process of fetching scenes nearby a given location is handled in `index()` method in `ScenesController`. The controller's action corresponds to `GET` `http://localhost:3000/api/v1/scenes/?longitude=X&latitude=Y` request where `X` and `Y` are the appropriate GPS coordinates. The request's URL can have one optional query string `radius`. The radius sets the distance from the user's location to search within. The default value is 2500 meters. The `index()` method handling the request calls `nearby` method with the provided longitude, latitude and radius as its parameters. The `nearby` method computes a square region around the user location using the equation 5.4. The equation computes the distance between two points. Each point has a pair of latitude, and longitude coordinates denoted (ϕ_i, λ_i). Differences in latitude and longitude are identified as $\Delta\phi$ and $\Delta\lambda$ and can be calculated by using the following formulas from [23]:

$$\Delta\phi = \phi_2 - \phi_1 \tag{5.1}$$

$$\Delta\lambda = \lambda_2 - \lambda_1. \tag{5.2}$$

The values of $\Delta\phi$ and $\Delta\lambda$ are in radians. The mean latitude is represented as ϕ_m and computed in [23] as follows:

$$\phi_m = \frac{\phi_1 + \phi_2}{2}, \tag{5.3}$$

where ϕ_1 and ϕ_2 are latitude coordinates for point P_1 and P_2 respectively. Additionally, let us assume that the Earth's radius is denoted as R and the radius is equal:

$$R = 6371009\,\text{kilometers}.$$

Finally, the equation to compute the distance between the two points is calculated in [23] as follows:

$$D = R\sqrt{(\Delta\phi)^2 + (\cos(\phi_m)\Delta\lambda)^2}. \tag{5.4}$$

The method uses the value of the `radius` parameter, or the default value 2500, as the distance D. Additionally, it computes the ϕ_m, $\Delta\phi$ and $\Delta\lambda$ values. Furthermore, the method applies the computed values to calculate `minLatitude`, `maxLatitude` and `minLongitude`, `maxLongitude`. Finally, the method queries the database and retrieves identifiers of scenes which are in the computed radius. The query that performs `INNER JOIN` and selects the scenes matching the criterion is shown in listing 5.1.

[6] `https://dry-rb.org/gems/dry-transaction/`

[7] Amazon Simple Storage Service is a cloud web storage for storing and retrieving data.

```
Scene.joins(:location).where("(? <= longitude) AND (longitude <= ?) AND (? <= ↩
    latitude) AND (latitude <= ?)",
                             minLongitude, maxLongitude,
                             minLatitude, maxLatitude)
```

Listing 5.1: The query performs an `INNER JOIN` with a single associated model – `location`. The query's where clause contains the condition that the scene's coordinates needs to be in a specified radius around the user location.

The API is deployed to Heroku cloud application platform. The application uses the *PostgreSQL* database, which is provided as an add-on on Heroku. Furthermore, the images are stored in Amazon S3 Bucket. The keys needed to connect and access the bucket are configured in Heroku's config vars. The deployment method is set to Github, and the application is connected to a private repository on Github from which a selected branch can be deployed to the platform.

Chapter 6

Results

6.1 Evaluating User Interface

The application's user interface is evaluated through the observations of testers interactions. The testers were handed over a device with the application already installed. The users were asked to explore the application, try to find a scene, display the information in the detail screen, and capture a new scene. Moreover, the application was tested with pairs of testers – one tester was responsible for capturing a new scene while the second had to recapture the same scene.

The first impressions from the application were generally positive. The users testing the application were mostly people acquainted with the iOS system, but one respondent had problems navigating the application because he was not familiar with the system. Apart from that, the users were pleasantly surprised by the fast navigation, simplicity, and three testers explicitly said that they like the color theme of the application. However, the purpose of the application had to be described to the testers before the testing. Otherwise, the testers were confused with the fact that they will be recapturing existing scenes. The confusion may be caused by the fact that most of the testers were not the actual target audience of the application, which is photographers with interest in rephotography. The best results were accomplished when the testers worked in a pair. One tester was tasked to find a scene, capture it, and follow through the process in the application. The second tester was then asked to go and capture the same scene. In this scenario, the first tester knew that the second tester depends on the input from the application, such as the feet shot and adequately placed the pin on the map. This way, the first tester tried to find a place which has distinctive features on the ground and captured a scene which would make sense to rephotograph such as a building or a tree.

There are some points from the testing that shows which functionality is indeed helpful from the point of the user and which not. One issue is that almost no one used the *Flip* button without explicitly said to do. The *Overlay* button was used almost in every case. However, the testers found out the functionality mainly after they were introduced to it. The *Navigate* functionality was also not used as the testers were in the walking distance from the scene, and they preferred the *View on map* functionality. The navigation system was understood without many trials and errors, but the users had first to try navigating the phone to different sides to understand it fully. After that, it was found helpful. In most cases, the user used the navigation system to align the device approximately and then proceed to find more subtle differences with the reference photo using the *Overlay*.

During the testing, the testers were asked to provide opinions about the functionality of the application and the overall review of the application. As for the user interface, one tester would prefer to have access to feet shot in the camera. The application has an easily extendable control panel, and the button could also be placed next to the shutter button on either side. In one case, the tester would prefer to capture the feet shot first and then proceed with capturing the scene. However, this could be the problem of a missing tutorial and a product page where the application would have more time to explain the purpose and flow of the capture process. One tester proposed unique *QR code* stickers that would be placed on the ground nearby some objects, and when the user scans the QR code, the application will start in a retake mode for the corresponding scene. However, this proposal is hard to implement as distributing the stickers would be troublesome and may cause problems. Nevertheless, in some use-cases where the owner of the place or the authority over the specified area would be interested in rephotographing its scenes or objects inside the area, then this proposal could be useful.

Overall, the application was reviewed as a well-designed application. The best reviews were given when the testers were using the application in pairs, or the tester had prior experience with photography and some interest in capturing the same scene at different times. The application found interest in the testers, and it even sparked creativity as they started to share more use-cases in which the application could be used and proposed some changes and functionality. The testers agreed that the navigation system with the proposed interface and functionality helped to capture a more accurate rephotograph.

6.2 Accuracy Analysis of Rephotographed Photos

The accuracy analysis of rephotographed photos is done on a selected scene that was rephotographed by the testers. The scene (see Figure 6.1) was rephotographed four times by different testers in order to capture the scene at different times and by various users. This models the envisioned common use-case of the application. In order to compare the rephotographs against the reference photograph, the selected scene has to include several significant points and edges in order to make the comparison. The scene contains a building and fountain, and both these objects can be used to align the reference photo onto the rephotograph as they contain straight lines and create edges. The testers used the navigation rectangle and *Overlay* feature before capturing the rephotograph. The rephotographs captured by the testers are displayed in Figure 6.2.

The accuracy of the rephotographed photos itself is determined based on Euclidean distance between selected points in the reference photo and corresponding points in the rephotographs. Figure 6.1 shows the key points used for the computation. Furthermore, Table 6.1 lists coordinates of the key points in the corresponding images. The coordinates are normalized to a range $\langle 0, 100 \rangle$ to make the results comparable amongst images with different resolution.

The Euclidean distance between two points in 2D is defined in [22] as

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}, \tag{6.1}$$

where $\mathbf{p} = (p_1, p_2)$ and $\mathbf{q} = (q_1, q_2)$. Furthermore, for each key point P, the distance is computed in pairs where point $\mathbf{p}$ is the point in the reference image and $\mathbf{q}_i, i \in 0 \dots 4$ is the corresponding point in rephotographs. The table 6.2 contains the computed Euclidean distances between the appropriate pairs for all key points.

Figure 6.1: The reference photograph with four key points used to determine the accuracy of the rephotographs.

Image	P_1	P_2	P_3	P_4
IMG_3419	$(31.51, 60.74)$	$(2.65, 33.85)$	$(96.46, 33.16)$	$(86.38, 76.93)$
IMG_3420	$(32.57, 59.28)$	$(2.98, 33.80)$	$(96.33, 34.03)$	$(85.48, 75.17)$
IMG_3432	$(33.04, 60.96)$	$(3.84, 34.47)$	$(97.22, 33.73)$	$(87.17, 76.96)$
IMG_3433	$(30.59, 63.17)$	$(1.92, 36.88)$	$(94.91, 36.53)$	$(84.69, 79.24)$
IMG_3434	$(31.51, 63.59)$	$(3.01, 37.00)$	$(95.87, 36.36)$	$(85.91, 79.74)$

Table 6.1: The key points coordinate normalized to a 100×100 image.

Image	$d_{P1}(p_0, q_i)$	$d_{P2}(p_0, q_i)$	$d_{P3}(p_0, q_i)$	$d_{P4}(p_0, q_i)$
IMG_3419	0	0	0	0
IMG_3420	1.81	0.33	4.09	1.97
IMG_3432	1.54	1.34	23.01	0.79
IMG_3433	2.60	3.11	47.12	2.86
IMG_3434	2.85	3.17	18.28	2.84

Table 6.2: The Euclidean distances computed for the corresponding key point and image against the reference photo IMG_3419.

$\mathbf{d_{P_x}(p_0, q_i)}$	$\bar{x}$	σ
$d_{P1}(p_0, q_i)$	2.20	0.53
$d_{P2}(p_0, q_i)$	1.99	1.15
$d_{P3}(p_0, q_i)$	23.13	12.00
$d_{P4}(p_0, q_i)$	2.12	0.73

Table 6.3: The arithmetic mean and standard deviation for the appropriate distances.

Table 6.3 shows the arithmetic mean and standard deviation for individual distances between the points. The overall arithmetic mean $\bar{x}$ based on all computed distances for each key point is **7.36**%, and the standard deviation is **3.60**%. The task of photographing the same scene at different times is difficult, in particular without the use of a tripod or other photographic equipment. The resulting accuracy achieved only with software guidance, based on a motion and location data, can be considered as an impressive result. Even the rotation of the rephotograph was not significant. Furthermore, by achieving this result, captured photographs can be used for the original purpose of tracking the scene changes over time.

However, this type of result is not achievable at locations with the non-distinctive ground, which is often the case of indoor photography. Moreover, the navigation system is dependent on several device sensors, and even with the sensor fusion, the accuracy and reliability of those sensors are not guaranteed.

Figure 6.2: Photographs of the fountain scene.

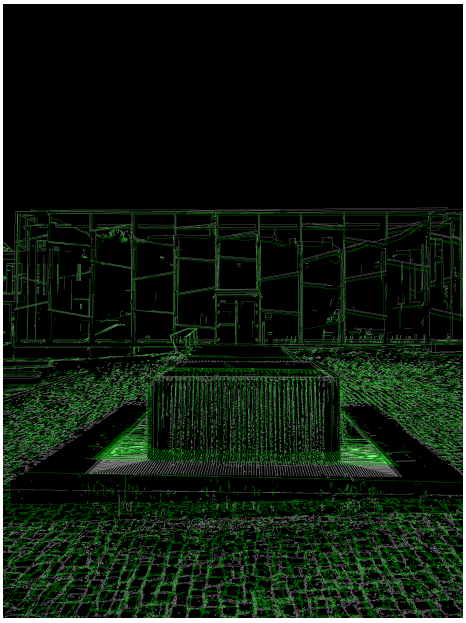

Figure 6.3: The Figure shows difference between the reference photograph (white color) and one of the rephotographs (green color). The edges were detected by using Canny edge detector.

Chapter 7

Conclusion

This thesis presents the problem of rephotography and introduces its application in several fields. Furthermore, the work explains the concept of computational rephotography and discusses related papers. Moreover, the project shows numerous rephotography projects and focuses on distinctive features. The analysis of camera applications for iOS devices shows their capabilities and concentrates on technologies applied in such applications. Further, the part about applications for viewing photographs presents widely known applications focused on sharing and viewing photographs and multimedia content. Additionally, the text discusses distinguishing features and user interface elements of the mentioned applications. Furthermore, the work contains a detailed description of several rephotography methods in section 2.4. The section presents algorithms, modules, and use-cases of the designed systems in their respective subsections.

The work describes the process of designing the mobile application, its user interface, and the server side and API. The text contains an overview of the iOS operating system and discusses several techniques of persisting data on iOS devices, showing map and annotations and network communication. Moreover, the work describes the process of capturing photos and using location and motion services on iOS devices.

The goal of the thesis is to devise a prototype of a mobile application focused on collaborative rephotography. In order to support sharing and viewing scenes and photographs amongst the users, a server with API for communication between the server and mobile application had to be implemented and deployed. The user interface is designed specifically for the task of rephotography with emphasis on user experience and optimized for the iOS platform.

The results of testing performed on a group of testers show that the application helps users capture rephotographs usable for tracking the scene changes over time. Moreover, the application was well-received during the testing, and the testers appreciated its simplicity and ease of use. Although the testers did not use some features, the evaluation of the user interactions shows that the application has to present and educate the users about rephotography and support requested controls in the camera screen such as the function to view a feet shot directly in the camera screen in recapture mode.

The mobile application provides a solid foundation for a platform that could be used for rephotography on a larger scale. The application would have to primarily focus on outdoor rephotography where the current system for navigating the user works best.

Bibliography

[1] Announcing Instagram Profiles on the Web!
Retrieved from:
https://www.tumblr.com/dashboard/blog/instagram/35068144047

[2] Camera Finder.
Retrieved from: https://www.flickr.com/cameras

[3] Halide Press Kit.
Retrieved from: https://www.dropbox.com/sh/pc0owz863y4jqdi/AABS08x6CaRdr-GZExLidcX3a?dl=0

[4] HTTP Methods.
Retrieved from: https://restfulapi.net/http-methods/

[5] Instagram for Android.
Retrieved from: https://instagram-press.com/blog/2012/04/03/instagram-for-android-available-now/

[6] Instagram Launches.
Retrieved from:
https://instagram-press.com/blog/2010/10/06/instagram-launches-2/

[7] Introducing Boomerang from Instagram.
Retrieved from: https://instagram-press.com/blog/2015/10/22/introducing-boomerang-from-instagram/

[8] Introducing Hyperlapse from Instagram.
Retrieved from: https://instagram-press.com/blog/2014/08/26/introducing-hyperlapse-from-instagram/

[9] Introducing Instagram Direct.
Retrieved from: https://instagram-press.com/blog/2013/12/12/introducing-instagram-direct/

[10] Introducing Instagram for Windows Phone.
Retrieved from: https://instagram-press.com/blog/2013/11/20/introducing-instagram-for-windows-phone/

[11] Introducing Instagram Stories.
Retrieved from: https://instagram-press.com/blog/2016/08/02/introducing-instagram-stories/

[12] Introducing Layout from Instagram.
Retrieved from: https://instagram-press.com/blog/2015/03/23/introducing-layout-from-instagram/

[13] Introducing Video on Instagram.
Retrieved from: https://instagram-press.com/blog/2013/06/20/introducing-video-on-instagram/

[14] ProCamera Quick Start Guide.
Retrieved from:
https://www.procamera-app.com/procamera_manual/QuickStartGuide_en.pdf

[15] ProCamera User Manual.
Retrieved from: https://manual.procamera-app.com/en/Manual_en.pdf

[16] Snapchat on the App Store.
Retrieved from: https://itunes.apple.com/app/snapchat/id447188370

[17] Snapchat's History: Evolution Of Snapchat And Timeline (2018).
Retrieved from: https://www.buycustomgeofilters.com/blog/snapchat-history-and-updated-timeline

[18] Thinking Outside the Square: Support for Landscape and Portrait Formats on Instagram.
Retrieved from:
https://instagram-press.com/blog/2015/08/27/thinking-outside-the-square-support-for-landscape-and-portrait-formats-on-instagram/

[19] VR.
Retrieved from: https://www.flickr.com/vr

[20] Welcome to IGTV.
Retrieved from:
https://instagram-press.com/blog/2018/06/20/welcome-to-igtv/

[21] What are the video upload requirements for IGTV?
Retrieved from: https://help.instagram.com/1038071743007909

[22] Euclidean distance. 2001-.
Retrieved from: https://en.wikipedia.org/wiki/Euclidean_distance

[23] Geographical distance. 2001-.
Retrieved from: https://en.wikipedia.org/wiki/Geographical_distance

[24] Quaternion. 2001-.
Retrieved from: https://en.wikipedia.org/wiki/Quaternion

[25] Historypin. 2010.
Retrieved from: https://www.historypin.org

[26] WhatWasThere. 2012.
Retrieved from: http://www.whatwasthere.com

[27] rePhoto. 2013.
Retrieved from: http://projectrephoto.com

[28] What Is Cocoa? 2013.
Retrieved from: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/CocoaFundamentals/WhatIsCocoa/WhatIsCocoa.html#//apple_ref/doc/uid/TP40002974-CH3-SW16

[29] Location and Maps Programming Guide. 2016.
Retrieved from: https://developer.apple.com/library/archive/documentation/UserExperience/Conceptual/LocationAwarenessPG/MapKit/MapKit.html#//apple_ref/doc/uid/TP40009497-CH3-SW1

[30] Core Data Programming Guide. 2017.
Retrieved from: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/CoreData/index.html#//apple_ref/doc/uid/TP40001075

[31] Core Data Core Competencies. 2018.
Retrieved from: https://developer.apple.com/library/archive/documentation/DataManagement/Devpedia-CoreData/persistentStore.html#//apple_ref/doc/uid/TP40010398-CH29-SW1

[32] Model-View-Controller. 2018.
Retrieved from: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

[33] Time After Time. 2018.
Retrieved from: https://www.markhersch.com

[34] Base64 encoding and decoding. 2019.
Retrieved from: https://developer.mozilla.org/en-US/docs/Web/API/WindowBase64/Base64_encoding_and_decoding

[35] Maths - Transformations using Quaternions. c1998-2017.
Retrieved from: http://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternions/transforms/index.htm

[36] Cameras and Media Capture. c2019.
Retrieved from: https://developer.apple.com/documentation/avfoundation/cameras_and_media_capture

[37] Core Data. c2019.
Retrieved from: https://developer.apple.com/documentation/coredata

[38] Core Location. c2019.
Retrieved from: https://developer.apple.com/documentation/corelocation

[39] Core Motion. c2019.
Retrieved from: https://developer.apple.com/documentation/coremotion

[40] Getting Started with iBeacon. c2019.
Retrieved from:
https://developer.apple.com/ibeacon/Getting-Started-with-iBeacon.pdf

[41] MapKit. c2019.
Retrieved from: https://developer.apple.com/documentation/mapkit

[42] URL Loading System. c2019.
Retrieved from:
https://developer.apple.com/documentation/foundation/url_loading_system

[43] Your Phone has Attitude! c2019.
Retrieved from:
http://trueviewvisuals.com/2015/02/20/your-phone-has-attitude/

[44] Apple, I.: IOS Technology Overview. 2012.

[45] Bae, S.; Agarwala, A.; Durand, F.: Computational Rephotography. *ACM Trans. Graph.*. vol. 29, no. 3. July 2010: pp. 24:1–24:15. ISSN 0730-0301. doi:10.1145/1805964.1805968.
Retrieved from: http://doi.acm.org/10.1145/1805964.1805968

[46] Bernazzani, S.: A Brief History of Snapchat.
Retrieved from: https://blog.hubspot.com/marketing/history-of-snapchat

[47] Borenstein, N.; Freed, N.: MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies. RFC 1521. RFC Editor. September 1993.

[48] Burke, P.: *Eyewitnessing*. Ithaca, USA: Cornell University Press. first edition. 2008. ISBN 978-0801473180.

[49] Crook, J.: Snapchat Launches v5.0 With Revamped UI, Swipe Navigation, And In-App Profiles. 2013.
Retrieved from:
https://techcrunch.com/2013/06/05/snapchat-launches-v5-0-banquo-with-revamped-ui-address-book-friend-finder-and-in-app-profiles/

[50] DeAmicis, C.: Did Snapchat succeed because of its controversial UI?
Retrieved from: https://www.figma.com/blog/did-snapchat-succeed-because-of-its-controversial-ui/

[51] Dempsey, J.: Portrait Mode.
Retrieved from: https://iphonephotographyschool.com/portrait-mode/

[52] Diebel, J.: Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors. *Matrix*. vol. 58. 01 2006.

[53] Dunsford, R.: iPhone Time Lapse.
Retrieved from: https://iphonephotographyschool.com/iphone-time-lapse/

[54] Editor, C.: Understanding How Slow-Motion Video Works.
Retrieved from: https://www.creativeplanetnetwork.com/news/understanding-how-slow-motion-video-works-608921

[55] En, S.; Lechervy, A.; Jurie, F.: RPNet: an End-to-End Network for Relative Camera Pose Estimation. *CoRR*. vol. abs/1809.08402. 2018. 1809.08402.
Retrieved from: http://arxiv.org/abs/1809.08402

[56] Gregersen, R.: View Controller Containment. 2013.
Retrieved from: https:
//www.objc.io/issues/1-view-controllers/containment-view-controller/

[57] Hemmings, M.: How To Use ProCamera App To Shoot Stunning iPhone Photos.
Retrieved from: https://iphonephotographyschool.com/procamera/

[58] Honan, M.: Apple unveils iPhone. 2004.
Retrieved from: https://www.macworld.com/article/1054769/iphone.html

[59] Kendall, A.; Grimes, M.; Cipolla, R.: Convolutional networks for real-time 6-DOF camera relocalization. *CoRR*. vol. abs/1505.07427. 2015. 1505.07427.
Retrieved from: http://arxiv.org/abs/1505.07427

[60] Lee, K.-T.; Luo, S.-J.; Chen, B.-Y.: Rephotography Using Image Collections. *Computer Graphics Forum*. vol. 30, no. 7. 2011: pp. 1895–1901. (Pacific Graphics 2011 Conference Proceedings).

[61] Levere, D.; Yochelson, B.; Abbott, B.: *New York changing*. New York: Museum of the City of New York. first edition. c2005. ISBN 15-689-8473-1.

[62] McArdle, J.: Retake Melbourne : mobile application for re-photography and comparative imaging research. 2013.
Retrieved from: http://hdl.handle.net/10536/DRO/DU:30056177

[63] Melekhov, I.; Kannala, J.; Rahtu, E.: Relative Camera Pose Estimation Using Convolutional Neural Networks. *CoRR*. vol. abs/1702.01381. 2017. 1702.01381.
Retrieved from: http://arxiv.org/abs/1702.01381

[64] Nijland, W.; Coops, N.; Coogan, S.; et al.: Vegetation phenology can be captured with digital repeat photography and linked to variability of root nutrition in Hedysarum alpinum. *Applied Vegetation Science*. vol. 16, no. 2. 2013: pp. 317–324. doi:10.1111/avsc.12000.
https://onlinelibrary.wiley.com/doi/pdf/10.1111/avsc.12000.
Retrieved from: https://onlinelibrary.wiley.com/doi/abs/10.1111/avsc.12000

[65] ProCamera, T.: What's New in ProCamera 8 v6.3.
Retrieved from:
https://www.procamera-app.com/en/blog/whats-new-in-procamera-8-v6-3/

[66] Raghu, S.: TimeLens. 2018.
Retrieved from: https://itunes.apple.com/us/app/timelens/id1240390371?mt=8

[67] Ruby, S.; Thomas, D.; Hansson, D. H.: *Agile Web Development with Rails 4*. Pragmatic Bookshelf. fourth edition. 2013. ISBN 1937785564, 9781937785567.

[68] Sandofsky, B.: Halide 1.7: In Depth.
Retrieved from: https://blog.halide.cam/halide-1-7-in-depth-17105d3ff740

[69] Shi, Y.-B.; Tian, F.; Miao, D.; et al.: Fast and Reliable Computational Rephotography on Mobile Device. 07 2018. doi:10.1109/ICME.2018.8486559.

[70] Snavely, N.; M. Seitz, S.; Szeliski, R.: Photo tourism: exploring photo collections in 3D. ACM Trans Graph 25(3):835-846. *ACM Trans. Graph.*. vol. 25. 07 2006: pp. 835–846. doi:10.1145/1141911.1141964.

[71] Steiner, S. C.: What is Focus Peaking?
Retrieved from: https://www.bhphotovideo.com/explora/photography/tips-and-solutions/what-focus-peaking

[72] Wesson, K.: Best Camera App For iPhone.
Retrieved from:
https://iphonephotographyschool.com/best-camera-app-for-iphone/

[73] Wesson, K.: Live Photos.
Retrieved from: https://iphonephotographyschool.com/live-photos/

[74] West, R.; Halley, A.; Gordon, D.; et al.: Collaborative Rephotography. In *ACM SIGGRAPH 2013 Studio Talks*. SIGGRAPH '13. New York, NY, USA: ACM. 2013. ISBN 978-1-4503-2343-7. pp. 20:1–20:1. doi:10.1145/2503673.2503693.
Retrieved from: http://doi.acm.org/10.1145/2503673.2503693

[75] Zappa, D.: How To Use iPhone Burst Mode For Incredible Action Photos.
Retrieved from: https://iphonephotographyschool.com/iphone-burst-mode/