VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

AUTOMATICKÝ OŘEZ SNÍMKŮ V PROSTŘEDÍ ANDROID

AUTOMATIC IMAGE CUT IN ANDROID

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PATRIK ONDRIGA**
AUTHOR

VEDOUCÍ PRÁCE **doc. Dr. Ing. DUŠAN KOLÁŘ**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Ondriga Patrik**
Program: Informační technologie
Název: **Automatický ořez snímků v prostředí Android**
Automatic Image Cut in Android
Kategorie: Počítačová grafika

Zadání:

1. Prostudujte aplikační prostředí OS Android. Zaměřte se na práci se snímky obrazovky, úpravu a editaci obrázků.
2. Nastudujte algoritmy pro identifikaci významných částí obrazu na základě podobnosti či dělicích linií a dále na automatickou retuši - odstranění nežádoucího textu.
3. Po dohodě s vedoucím navrhněte algoritmus, který provede automatický ořez snímku obrazovky v prostředí Android. Návrh aplikace musí počítat s víceznačným vstupem (konzultujte s vedoucím), možností ruční editace, apod. Dále vyberte vhodný algoritmus pro automatickou retuši textu ze snímku - text bude vhodným způsobem vybrán ručně.
4. Aplikaci implementujte dle návrhu z předchozího bodu a důkladně otestujte.
5. Zhodnoťte přínos aplikace, diskutujte možná další rozšíření.

Literatura:

- Documentation for app developers, dostupné z: https://developer.android.com/docs [cit. 2020-10-25]
- Allen B. Downey, Chris Mayfield: Think Java: How to Think Like a Computer Scientist, 2nd Edition, ISBN: 978-1492072508, 2020
- Daniel Lin: A Novel Method of Detecting Lines on a Noisy Image, 2015, dostupné z: https://ir.library.oregonstate.edu/xmlui/bitstream/handle/1957/56121/Dan_Senior_Thesis.pdf?sequence=1 [cit. 2020-10-25]
- Dle doporučení vedoucího

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Kolář Dušan, doc. Dr. Ing.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 18. října 2021

Abstrakt

Práca je zameraná na navrhovanie orezov obrázka. Myšlienkou bolo vytvoriť mobilnú aplikáciu, ktorá by zo snímky obrazovky niektorej sociálnej siete navrhla orez tak, aby zostal len samotný obrázok bez prostredia sociálnej siete. Na tento účel nebola využitá umelá inteligencia, ale hranové funkcie.

Abstract

Thesis is focused on suggest image cut. The idea was to create a mobile application that would suggest image cut from screenshot of a social network so after that, there was only image without social network environment. No AI was used for this purpose, but edge detection methods.

Kľúčové slová

hranové funkcie, Canny, Prewitt, Laplacian, Sobel, orezanie snímky, mobilná aplikácia, inpainting

Keywords

edge detection methods, Canny, Prewitt, Laplacian, Sobel, image cut, mobile application, inpainting

Citácia

ONDRIGA, Patrik. *Automatický ořez snímků v prostředí Android.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Dr. Ing. Dušan Kolář

Automatický ořez snímků v prostředí Android

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána doc. Dr. Ing. Dušana Koláře. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Patrik Ondriga
6. mája 2022

Poďakovanie

Rád by som poďakoval vedúcemu mojej práce doc. Dr. Ing. Dušanovi Kolářovi, ktorý ma viedol procesom tvorby a poskytoval mi odborné konzultácie.

Obsah

Kapitola 1

Úvod

Táto práca je zameraná na jednoduchšie orezávanie snímok obrazovky niektorej sociálnej siete. Myšlienkou je vytvorenie aplikácie pre mobilné zariadenia, ktorá svojmu užívateľovi poskytne automatický návrh pre orezanie obrázka. Užívateľ potom jedným kliknutím môže orez potvrdiť, alebo jednoducho prispôsobiť.

Druhou hlavnou funkcionalitou aplikácie je odstraňovanie menších oblastí z obrázka. Tie môžu obsahovať napríklad vodoznak. Oblasť, ktorú si užívateľ vyberie na odstránenie, je nahradená buď pomocou metódy Navier-Stokes, alebo algoritmom navrhnutým Alexandrom Telea.

V súčasnosti, ak pri prezeraní príspevkov na sociálnej sieti bez tejto aplikácie si chceme uložiť nejaký obrázok, musíme prejsť niekoľkými krokmi. Je potrebné urobiť snímku obrazovky a následne ručne posunúť okraje orezu všetkých štyroch strán. Tento postup je časovo zdĺhavejší a vyžaduje vyššiu mieru sústredenia. Táto možnosť je navyše dosť nekomfortná v prípade, ak sa nachádzame napr. v prostriedku mestskej hromadnej dopravy.

Aplikácia nevyužíva umelú inteligenciu, ale pracuje s hranovými funkciami. Pomocou nich sa zvýraznia okraje obrázka na snímke obrazovky zo sociálnej siete. Aplikácia je implementovaná pre zariadenia s operačným systémom Android.

Kapitola 2

Teoretická časť

2.1 Digitálny obraz

Obraz je definovaný ako dvojrozmerná funkcia $F(x, y)$, kde x a y sú priestorové súradnice. Amplitúda F na ľubovoľnom páre súradníc (x, y) sa nazýva intenzita tohto obrazu na danom bode. Keď sú hodnoty x, y a amplitúdy F konečné, nazývame to digitálny obraz.

Inými slovami, obrázok môže byť definovaný dvojrozmerným poľom špecificky usporiadaným do riadkov a stĺpcov.

Digitálny obraz sa skladá z konečného počtu prvkov, z ktorých každý má určitú hodnotu na určitom mieste. Tieto prvky sa označujú ako prvky obrazu, alebo ako pixely. Najčastejšie sa ale používa označenie pixel. [9]

2.2 Hranová detekcia

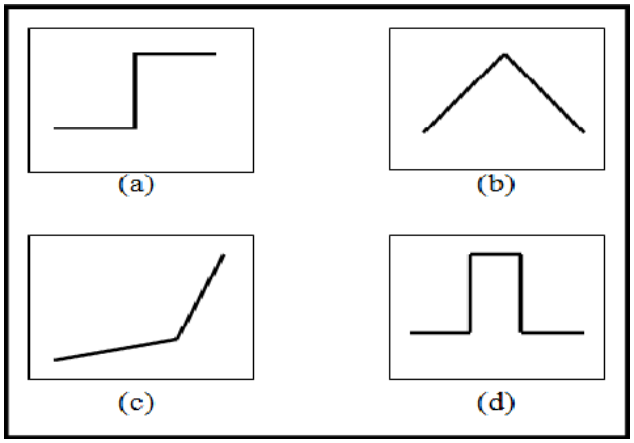
2.2.1 Hrany v digitálnych obrazoch

Detekcia hrán je základným a dôležitým nástrojom v hlavných oblastiach spracovávania obrazu, akým je detekcia objektov a extrakcia objektov z obrázka.

Hrany môžu byť spôsobené zmenami svetla, farieb, tieňov a textúrou. Tieto zmeny môžu byť využité na určenie hĺbky, veľkosti, orientácie a povrchových vlastností v digitálnom obraze. Digitálna analýza obrázka pomáha filtrovať nevyhnutné informácie na výber bodov hrán. Pri detekcii jemných zmien môže dôjsť k detegovaniu šumu ako hrany. Množstvo šumu, ktoré je detegované ako hrana a množstvo jemných zmien je závislé od nastaveného prahu pre pixely. Detekcia týchto hrán je veľmi náročná a zaberá veľa času, najmä ak je obraz poškodený šumom. [11]

Štandardné metódy detekcie hrán zahŕňajú konvolúciu obrázka s operátorom. Takýto operátor musí byť citlivý na veľké zmeny gradientu na obrázku, zatiaľ čo vracia nulové hodnoty pre jednotné oblasti. [10]

Na obrázku 2.1 sú znázornené základné druhy hrán.



Obr. 2.1: Základné druhy hrán. Prevzaté z [19].

a. **schod**: intenzita v obrázku sa náhle zmení z jednej krajnej hodnoty na druhú krajnú hodnotu,

b. **strecha**: keď zmena intenzity nenastane náhle, ale trvá určitý čas,

c. **rampa**: keď zmena intenzity nenastane náhle, ale nastane na určitej vzdialenosti, takýmto spôsobom sa schodová hrana mení na rampovú,

d. **čiara**: intenzita v obrázku náhle zmení svoju hodnotu a vráti sa na pôvodnú na krátkej vzdialenosti. [19]

2.2.2 Detekcia hrán

Detekciu hrán je možné definovať ako zobrazovanie čiar, ktoré označujú hranice a rozdeľujú predmety od ostatného priestoru, alebo od iných predmetov na obrázku. [4]

Detekcia hrán využíva prístup, pri ktorom sa zmeny intenzity vyskytujúce sa v bodoch obrazu deklarujú ako hrana. Ide o sériu akcií, ktorá sa používa na identifikáciu týchto bodov v obraze, ktoré jasne definujú zmeny intenzity. Táto séria akcií je potrebná na extrahovanie informácií súvisiacich s obrázkom, napr. zaostrenie, vylepšenie a umiestnenie objektu na obrázku. [1]

2.3 Algoritmus detekcie hrán

Algoritmus detekcie hrán je definovaný týmto postupom:

1. vstupným parametrom je farebný obrázok,
2. vyhladzovanie - vyhladzovanie sa používa na odstránenie šumu tak, aby sa zachovali nepoškodené skutočné hrany na obrázku,
3. zvýraznenie, aplikovanie derivácie na zvýšenie kvality hrán,
4. nastavenie prahu rozsahu pre odlíšenie zašumených pixelov od pixelov hrany,
5. lokalizácia - niektoré aplikácie lokalizujú predmety pomocou hrán a medzier medzi pixelmi,
6. získanie obrázka po zvýraznení hrán. [1]

2.3.1 Detekcia hrán založená na výpočte gradientu

Na detegovanie hrán na obrázku existuje množstvo algoritmov a tie môžeme klasifikovať ako algoritmy založené na derivácii, alebo na gradiente. Algoritmus založený na derivácii berie prvú, alebo druhú deriváciu na každom pixele. Algoritmus založený na gradiente berie gradient po sebe idúcich pixelov v smere x a y. Na tento účel sa používa kernel operácia. Kernel je malá matica, kde jej stred leží na zvolenom pixele obrázka, vynásobená koeficientmi filtra zodpovedajúcimi pixelom matice obrázka pre špecifikovaný pixel umiestnený v strede matice. Ak je vypočítaná hodnota nad určeným prahom, potom pixel v strede sa klasifikuje ako hrana. Takýto výpočet sa opakuje pre každý pixel obrázka posúvaním po matici obrázka zľava doprava a zhora dole. Sobel, Prewitt a Canny sú príkladmi gradientových metód detekcie hrán. [16]

Prístup založený na výpočte gradientu sa v digitálnych obrázkoch nazýva aj maskovanie. Diferenciálne aproximácie v horizontálnom alebo vertikálnom smere obrazu sú vypočítané pomocou digitálnej masky. [4] Pod pojmom digitálna maska je myslený kernel.

2.3.2 Sobel

Sobel operátor je jeden z najčastejšie používaných hranových detekcií. [19] Počíta dvojrozmernú veľkosť gradientu nad obrázkom a používa sa na nájdenie približnej absolútnej veľkosti gradientu v každom bode na vstupnom obrázku v odtieňoch šedej. Sobel hranová detekcia používa dvojicu 3x3 konvolučných masiek. Jedna vyhodnocuje gradient v x-ovom smere (stĺpce) a druhá zase vyhodnocuje gradient v y-ovom smere (riadky). Konvolučná maska je všeobecne omnoho menšia ako skutočný obrázok. [15] Pomocou Sobel operátora je gradient obrázka I, označený ako M, počítaný nasledovne: [5]

$$M = \sqrt{S_x^2 + S_y^2}$$

S_x je gradient obrázka vo vertikálnom smere vypočítaný konvolúciou masky s obrázkom I nasledovne:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I$$

S_y je gradient obrázka v horizontálnom smere vypočítaný konvolúciou masky s obrázkom I nasledovne:

$$S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * I$$

2.3.3 Prewitt

Prewitt hranová detekcia je vhodná na odhadnutie veľkosti a orientácie hrán. [19] Je podobná operátoru Sobel a je používaná pre detegovanie vertikálnych a horizontálnych hrán v digitálnych obrazoch [15] v prípade, keď obraz má jasné prechody intenzity a obsahuje len malý šum. Je vhodná len pre dobre kontrastné nezašumené obrazy. Používa 3x3 masky pre nájdenie vrcholnej veľkosti gradientov obrázka I, označené ako M. Keď sa nájde maximálna veľkosť, tak detekcia pracuje s podobnou rovnicou, akú používa Sobel. [5]

$$M = \sqrt{G_x^2 + G_y^2}$$

G_x je gradient obrázka vo vertikálnom smere vypočítaný konvolúciou masky s obrázkom I nasledovne:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

G_y je gradient obrázka v horizontálnom smere vypočítaný konvolúciou masky s obrázkom I nasledovne:

$$G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

2.3.4 Laplacian

Laplacian metóda hľadá nulové prieniky pre druhú deriváciu obrázka na nájdenie hrán, kedy sa používa Laplacian operátor. [17] Táto metóda má fixnú charakteristiku vo všetkých smeroch a senzitivitu na šum. Hrany určuje krížením núl z početných uzavretých slučiek. Metódy nulového kríženia sú zaujímavé kvôli ich schopnostiam znižovať šum a majú potenciál pre robustný výkon. [20] Pre určenie hrán na obrázku sa nemusí použiť zložitá rovnica obsahujúca druhú deriváciu, ale kernel operácia. Pre tento účel môžeme použiť jednu z dvoch konvolučných masiek:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

2.3.5 Canny

Spomedzi doteraz navrhnutých metód detekcie hrán je Canny najdôkladnejšie definovaným operátorom a je široko používaný. Obľúbenosť detektora hrán Canny možno pripísať jeho optimálnosti z pohľadu troch kritérií: dobrej detekcie, dobrej lokalizácie a jedinej odozvy na hranu. Má tiež pomerne jednoduchú implementáciu. [2]

Zaisťuje dobrú odolnosť proti šumu a zároveň deteguje skutočné hrany s minimálnou chybovosťou. [20] Využíva viacúrovňový algoritmus na detekciu širokého rozsahu hrán.

Prvým krokom je odfiltrovanie šumu v pôvodnom obrázku predtým, ako sa pokúsi nájsť akékoľvek hrany. Nakoľko Gaussov filter je možné vypočítať pomocou jednoduchej masky, je využívaný v algoritme Canny. Po vypočítaní vhodnej masky je možné vykonať Gaussove vyhladenie pomocou štandardných konvolučných metód. Konvolučná maska je zvyčajne oveľa menšia ako skutočný obrázok. Výsledkom toho je, že maska sa posúva cez obrázok a súčasne manipuluje so štvorcom pixelov. Čím väčšia je šírka Gaussovej masky, tým nižšia je citlivosť detektora na šum.

Po vyhladení obrazu a odstránení šumu je ďalším krokom zistenie sily hrán pomocou gradientu obrazu. Pre tento účel sa používa Sobel operátor.

Ďalej sa počíta smer hrany. Ten sa vypočíta pomocou gradientu v smeroch x a y. Ak sa však súčet x rovná nule, vygeneruje sa chyba. Takže v kóde musí byť stanovené obmedzenie vždy, keď k tomu dôjde. Keď je gradient v smere x nulový, smer hrany sa musí rovnať 90 stupňom, alebo 0, v závislosti od toho, aká je hodnota gradientu v smere y. Ak má gradient v smere y hodnotu nula, smer hrany sa bude rovnať 0 stupňom. V opačnom prípade bude smer hrany rovný 90 stupňom.

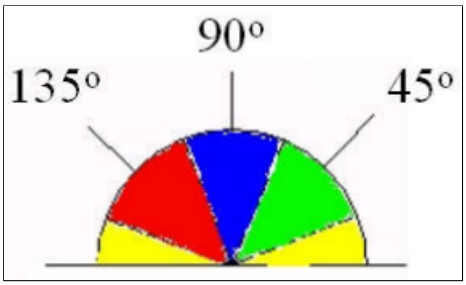
Keď je známy smer hrany, ďalším krokom je uvedenie smeru hrany do vzťahu so smerom, ktorý možno vysledovať na obrázku 2.2. Ak sú teda pixely obrázka 5x5 zarovnané nasledovne

$$\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & A & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{matrix}$$

je možné pri pohľade na pixel „A“ usúdiť, že pre okolité pixely existujú len štyri možné smery:

- 0 stupňov (v horizontálnom smere),
- 45 stupňov (pozdĺž kladnej uhlopriečky),
- 90 stupňov (vo vertikálnom smere),
- 135 stupňov (pozdĺž zápornej uhlopriečky).

Teraz sa orientácia hrán musí rozdeliť do jednej z týchto štyroch smerov v závislosti od toho, ku ktorému smeru je najbližšie.



Obr. 2.2: Farebné znázornenie smerov hrán. Prevzaté z [10].

Akýkoľvek smer patriaci do žltého rozsahu (0 až 22,5 a 157,5 až 180) je nastavený na 0 stupňov. Akýkoľvek smer hrany ležiaci v zelenom rozsahu (22,5 až 67,5) je nastavený na 45 stupňov. Akýkoľvek smer patriaci do modrého rozsahu (67,5 až 112,5) je nastavený na 90 stupňov. A nakoniec akýkoľvek smer hrany patriaci do červeného rozsahu (112,5 až 157,5) je nastavený na 135 stupňov.

Keď už sú známe smery hrán, je potrebné použiť „nie maximálne“ potlačenie. „Nie maximálne“ potlačenie sa používa na pozdĺžne sledovanie okraja v smere hrany a potláča akékoľvek hodnoty pixela, ktorý sa nepovažuje za okraj. Na výstupnom obrázku tak vznikne tenká čiara.

Nakoniec sa použije hysterézia na odstránenie pruhovania. Pruhovanie je rozbitie obrysu hrany spôsobené kolísaním výkonu operátora nad a pod prahovou hodnotou. Ak sa na obrázok aplikuje jeden prah $T1$ a hrana má priemernú silu rovnajúcu sa $T1$, potom v dôsledku šumu nastanú prípady, keď hrana klesne pod prah. Rovnako niekedy bude presahovať aj prah, takže hrana bude vyzerať ako prerušovaná čiara. Aby sa tomu zabránilo, hysterézia používa 2 prahové hodnoty, vysokú a nízku. Každý pixel v obraze, ktorý má hodnotu väčšiu ako $T1$, sa považuje za hranový pixel. Potom sa všetky pixely, ktoré sú pripojené k tomuto hranovému pixelu a majú hodnotu väčšiu ako $T2$, tiež považujú za hranové pixely. [10]

2.4 Šum v obrázkoch

Šum je náhodná zmena intenzity v obraze a je viditeľná ako zrná v obraze. Môže sa objaviť na obrázku ako efekt základných fyzikálnych efektov ako fotónová povaha svetla, alebo energia tepla vo vnútri senzora fotoaparátu. Tieto narušenia obrazu môžu vzniknúť v čase snímania, alebo pri prenose obrazu. Šum znamená, že pixely v obraze zobrazujú rôzne hodnoty intenzity namiesto skutočných hodnôt pixelov.

Algoritmus odstránenia šumu je proces odstránenia, alebo zníženia šumu z obrazu. Algoritmy odstránenia šumu znižujú, alebo odstraňujú viditeľnosť šumu vyhladzovaním celého obrazu a ponechávajú oblasť blízko hraníc kontrastu nezmenenú. Tieto algoritmy však môžu zakryť jemné detaily s nízkym kontrastom. [18]

Rozdielne šumy majú svoje vlastné charakteristiky, vďaka ktorým sú rozlíšiteľné od ostatných. Bežné typy šumu, ktoré vznikajú v obraze sú tieto: [8]

1. impulzný šum,
2. aditívny šum,
3. multiplikačný šum.

2.5 Inpainting

Hoci *inpainting* obrazov, alebo umenie opravy starých a poškodených obrazov existuje už mnoho rokov, v poslednej dobe si získalo ešte väčšiu popularitu kvôli nedávnemu vývoju techník spracovania obrazu. So zlepšením nástrojov na spracovanie obrazov flexibilitou úprav digitálneho obrazu našiel automatický *inpainting* obrazu dôležité aplikovanie v počítačovom videní a stal sa tiež dôležitou a náročnou témou výskumu v oblasti spracovania obrazu.

Zdieľané obrázky v sociálnych sieťach môžu obsahovať mnoho objektov pridaných k týmto obrázkom vrátane podpisu, vodoznaku alebo emotikonov. Pridanie týchto objektov môže zmeniť sémantiku obrázkov. Ich odstránenie z obrázkov je všeobecne uznávaným problémom a aktuálnou témou vo výskume počítačového videnia.

Inpainting obrazu je proces dokončenia, alebo obnovenia chýbajúcej oblasti v obraze, alebo odstránenia niektorých pridaných objektov. Operácia *inpainting* závisí od typu poškodenia na obrázku a od aplikácie, ktorá toto poškodenie spôsobila. Napríklad pri obnove obrázkov hovoríme o odstránení škrabancov alebo textu, ktoré sa nachádzajú na obrázkoch, zatiaľ čo v aplikácii na úpravu fotografií nás zaujíma odstránenie objektov v kódovaných obrazoch a prenosových aplikáciách. Operácia súvisiaca s *inpainting* je obnovovanie chýbajúcich blokov. Pri obnove virtuálnych malieb je súvisiacou operáciou odstránenie škrabancov. [21]

2.5.1 Telea

Telea algoritmus je založený na technike *Fast marching method*. Vytvoril ho Alexandru Telea v roku 2004. V oblasti, ktorá bola vybraná na opravu, algoritmus začína z jej okrajov a pokračuje dovnútra. Takýmto spôsobom postupne vypĺňa zvolenú oblasť. Pre nahrádzaný pixel sa zoberú hodnoty susediacich pixelov. Normalizovaný váhový súčet týchto hodnôt sa stáva novou hodnotou pre daný pixel. Výber váhy je dôležitý. Väčšiu váhu majú tie pixely, ktoré ležia blízko bodu, blízko normály hranice a tie, ktoré ležia na hraničných obrysoch.

Keď je pixel nahradený, algoritmus sa presunie na ďalší najbližší pomocou *Fast marching method*. Pomocou tejto metódy sa zabezpečí to, že sa nová hodnota vypočíta najprv pre tie, ktoré susedia so známymi pixelmi. V podstate to teda funguje ako manuálna heuristická operácia. [14]

2.5.2 Navier-Stokes

Navier-Stokes algoritmus vytvorili Marcelo Bertalmio, Andrea L. Bertozzi, a Guillermo Sapiro v roku 2001. Tento algoritmus je založený na dynamike tekutín a využíva parciálne diferenciálne rovnice. Základným princípom je heuristika. Najprv sa pohybuje pozdĺž okrajov zo známych regiónov do neznámych (pretože hrany majú byť súvislé). Pokračuje po čiarach spájajúcich body s rovnakou intenzitou jasu, pričom sa porovnávajú gradientové vektory na hranici oblasti maľby. Na tento účel sa používajú niektoré metódy z dynamiky tekutín. Po ich získaní sa farba vyplní, aby sa znížil minimálny rozptyl v tejto oblasti. [14]

2.6 OpenCV

OpenCV (Open Source Computer Vision Library) je open source softvérová knižnica počítačového videnia a strojového učenia. OpenCV bol vytvorený s cieľom poskytnúť spoločnú infraštruktúru pre aplikácie počítačového videnia a urýchliť využitie strojového vnímania v komerčných produktoch. Keďže ide o produkt s licenciou BSD, OpenCV uľahčuje firmám používanie a úpravu kódu.

Knižnica má viac ako 2500 optimalizovaných algoritmov, ktoré zahŕňajú komplexnú sadu klasických aj najmodernejších algoritmov počítačového videnia a strojového učenia. Tieto algoritmy možno použiť na detekciu a rozpoznávanie tvárí, identifikáciu objektov, klasifikáciu ľudských činností vo videách, sledovanie pohybov kamery atď. OpenCV má viac ako 47 tisíc používateľov komunity a odhadovaný počet stiahnutí presahuje 18 miliónov. Knižnica je široko používaná v spoločnostiach, výskumných skupinách a vládnych organizáciách.

Má rozhrania pre C++, Python, Java a MATLAB a podporuje Windows, Linux, Android a Mac OS. OpenCV sa používa hlavne v aplikáciách so spracúvaním obrazu v reálnom čase a využíva inštrukcie MMX a SSE, ak sú k dispozícii. [13]

2.7 Android prístup k obrázkom

Mnohé aplikácie umožňujú používateľom vytvárať a pristupovať k médiám, ktoré sú k dispozícii na externom úložisku. *MediaStore framework* poskytuje optimalizovaný index do kolekcií médií. Umožňuje jednoduchšie získavanie a aktualizáciu týchto súborov. Aj po odinštalovaní aplikácie zostanú tieto súbory v zariadení používateľa.

Systém automaticky prehľadáva externé úložisko a pridáva mediálne súbory do nasledujúcich dobre definovaných kolekcií:

- **Obrázky** - obsahuje fotografie a snímky obrazovky, ktoré sú uložené v priečinkoch *DCIM* a *Pictures*,
- **Videá** - obsahuje videá, ktoré sú uložené v priečinkoch *DCIM*, *Movies* a *Pictures*,
- **Zvukové súbory** - obsahuje zvukové súbory, ktoré sú uložené v priečinkoch *Alarms*, *Audiobooks*, *Music*, *Notifications*, *Podcasts*, *Ringtones*, *Recordings* a *Movies*,

- **Stiahnuté súbory** - obsahuje súbory, ktoré sú uložené v priečinku *Download*.

Ak aplikácia zobrazuje niekoľko mediálnych súborov a požaduje, aby užívateľ zvolil jeden z nich, je oveľa efektívnejšie načítať iba náhľad takýchto súborov namiesto samotných súborov v plnej kvalite. Pre načítanie náhľadov má *framework* vytvorenú metódu, kde je možné definovať jeho veľkosť. [3]

Práca s *MediaStore* je podobná ako s SQL databázou. Pre prácu s obrázkami sa používajú podobné dotazy ako práve v spomínanom SQL jazyku.

2.8 Glide

Glide je rýchly a efektívny *framework* s otvoreným zdrojovým kódom pre správu médií a načítavanie obrázkov pre Android. Zahŕňa dekódovanie médií, ukladanie operačnej pamäte a disku do vyrovnávacej pamäte a združovanie zdrojov do jednoduchého a ľahko použiteľného rozhrania. Glide podporuje načítavanie, dekódovanie a zobrazenie statických záberov videa, obrázkov a animovaných obrázkov *gif*. Glide obsahuje flexibilné API, ktoré umožňuje vývojárom pripojiť sa k takmer akémukoľvek sieťovému zásobníku.

Glide sa primárne zameriava na to, aby rolovanie akéhokoľvek druhu zoznamu obrázkov bolo čo najplynulejšie a najrýchlejšie. Glide je ale tiež účinný takmer v každom prípade, keď je potrebné načítať, zmeniť veľkosť a zobraziť vzdialený obrázok.

Minimum Android SDK: Glide vyžaduje minimálnu úroveň API 14.

Compile Android SDK: Glide vyžaduje kompiláciu s rozhraním API 26 alebo novším. [6]

2.9 Subsampling Scale Image View

Subsampling Scale Image View je komponent pre zobrazovanie obrázkov pre Android navrhnutý pre fotografie a zobrazovanie obrovských obrázkov (napr. mapy a plány budov) bez chýb *OutOfMemoryErrors*. Umožňuje približovanie, posúvanie, otáčanie a zobrazovanie animácií. Podporuje jednoduché rozšírenie, takže je možné pridať vlastné prekrytie a detekciu udalostí dotyku.

Zobrazovanie voliteľne využíva podvzorkovanie a dlaždice na podporu zobrazenia veľmi veľkých obrázkov. Na začiatku je načítaná základná vrstva s nízkym rozlíšením a pri priblížení sa prekryje menšími dlaždicami s vysokým rozlíšením pre viditeľnú oblasť. Tým sa zabráni uchovávaniu príliš veľkého množstva dát v pamäti. Tento komponent je ideálny na zobrazovanie veľkých obrázkov a zároveň umožňuje priblíženie na detaily s vysokým rozlíšením. Pre menšie obrázky a pri zobrazovaní bitmapového objektu je možné zakázať dlaždicové usporiadanie. [12]

Kapitola 3

Dáta

Pre správne nastavenie algoritmu a následne jeho otestovanie bolo potrebné zozbierať testovacie obrázky.

Najprv som si určil prípady, pri ktorých by algoritmy určovania hrán mohli dôjsť k nesprávnym hodnotám. Všetky testovacie obrázky som zozbieral na dvoch sociálnych sieťach osobne. V prvom kroku som hľadal prípady, kedy okraje obrázka splývali s okolitým prostredím. Ďalší problém by mohol nastať, ak obrázok obsahuje veľké množstvo striedajúcich sa farieb. Taktiež som pridal aj snímky s malým množstvom farby. Snímky som vytvoril pri zapnutom tmavom a aj svetlom režime.

Jeden z prípadov, ktorý by mohol spôsobiť nesprávne určenie obrázka v snímke, je využitie nesprávnych hrán. Keďže určujem okraje orezu v tvare obdĺžnika, akékoľvek vodorovné alebo horizontálne čiary môžu spôsobiť nesprávne určenie hraníc obrázka. Dobrým príkladom je napríklad tabuľka, alebo obdĺžnikový objekt. Pridal som aj obrázky, pri ktorých by určenie orezu malo prebehnúť bez problémov.

Tieto snímky som urobil v troch rozličných zariadeniach s rozličnými veľkosťami obrazoviek. Prvé zariadenie má 5-palcový displej s rozlíšením 1080x1920 pixelov, druhé zariadenie má 6,4-palcový displej s rozlíšením 1440x2960 pixelov a tretie zariadenie má 6,21-palcový displej s rozlíšením 1080x2248 pixelov. Táto sada obsahovala 33 snímok a slúžila prevažne na vývoj a odladenie.

Vytvoril som aj druhú sadu snímok za pomoci týchto troch zariadení. Táto slúžila výlučne na otestovanie a vyhodnotenie správnosti navrhovaných orezov. Sada obsahuje celkovo 100 snímok. Nachádzajú sa tu aj obrázky zo sady používanej pri vývoji. Všetkých 100 snímok som rozdelil do dvoch skupín. Prvá skupina obsahovala obrázky, pri ktorých som predpokladal, že by algoritmus mal byť schopný minimálne rozpoznať hrany, kde by sa mal nachádzať orez. V tejto skupine je 78 snímok. V druhej skupine vždy minimálne jedna hrana obrázka vo vnútri snímky splývala s prostredím sociálnej siete. Tu bola teda vysoká pravdepodobnosť, že orez bude algoritmom určený nesprávne. Tu sa nachádza 22 snímok.

Pre účely zistenia rýchlosti nájdenia orezu vzhľadom na veľkosť snímky som vytvoril tretiu sadu obrázkov. Tá obsahuje 40 snímok. Táto sada je taktiež rozdelená do dvoch skupín. V prvej skupine je 20 unikátnych snímok obrázkov vyhotovených pri rozlíšení displeja 720x1480 pixelov. V druhej skupine sa nachádza taktiež 20 unikátnych snímok, ale v rozlíšení displeja 1440x2960 pixelov. V prvej aj druhej skupine sa nachádzajú snímky obrazovky rovnakých obrázkov, len s rozdielnym rozlíšením displeja. Táto sada snímok bola vyhotovená tým istým mobilným zariadením s veľkosťou displeja 5 palcov, na ktorom je možné meniť jeho rozlíšenie.

Kapitola 4

Algoritmus pre určovanie hraníc obrázka a inpainting

4.1 Určovanie hraníc obrázka

Aby algoritmus určil správne orez, je potrebné lokalizovať obrázok v snímke. Tento obrázok je vždy v tvare obdĺžnika. Problém ale nastáva v tom, že jeho výška nie je vždy rovnaká. Okrem toho jeho poloha nie je fixná a teda sa skoro nikdy nenachádza na rovnakom mieste. Po preskúmaní mojich testovacích údajov som si spísal trojbodový zoznam informácií, podľa ktorých bude algoritmus hľadať okraje obrázka.

1. Obrázok je v tvare obdĺžnika a preto je potrebné nájsť všetky vodorovné a zvislé čiary.
2. Obrázok väčšinou zaberá celú šírku obrazovky.
3. Obrázok vždy začína na vrchnej polovici obrazovky a končí na spodnej a to bez ohľadu na jeho veľkosť alebo umiestnenie.

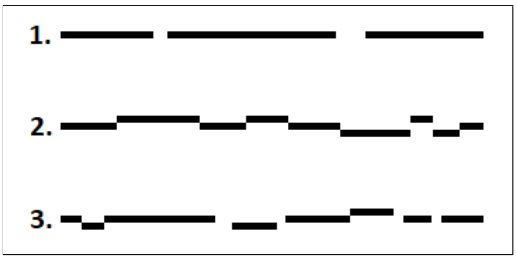
4.1.1 Hranové funkcie

Pre lokalizáciu obrázka je potrebné nájsť jeho okraje. Pre tento účel je ideálne použiť hranové funkcie. Pomocou nich nájde algoritmus všetky hrany, ktoré sa v snímke nachádzajú. Ako už bolo spomenuté, pre účel programu sú podstatné len vodorovné a zvislé hrany. Tieto hrany ďalej označujem ako čiary.

Pre nájdenie vodorovných a zvislých čiar som použil štyri hranové detekcie. Používam Canny, Laplacian, Sobel a Prewitt algoritmy. Pôvodný obrázok som teda previedol do odtieňov šedej, nakoľko tieto detekcie na vstup neočakávajú farebné obrázky, ale práve v odtieni šedej. Ďalším krokom by malo byť použitie algoritmu na vyhladenie snímky a tým odstránenie šumu. V mojom prípade sa analyzuje snímka obrazovky a nie fotky vyhotovenej fotoaparátom. Takže hrany, ktoré sú pre účely nájdenia okrajov obrázka dôležité, nie sú znehodnotené šumom. Preto som tento krok mohol preskočiť, čím som ušetril výpočtový čas. Pre prah rozsahu, podľa ktorého sa určujú body hrany od okolia, som sa rozhodol nastaviť fixnú hodnotu. Je to z toho dôvodu, aby som ušetril ďalší čas. Tento prah som nastavoval pozorovaním správania programu na testovacích snímkach tak, aby som zachytil čo najviac reálnych bodov hrán bez tých falošných.

Po použití hranových funkcií sa často stáva, že sú čiary čiastočne poškodené (obr. 4.1). Nedostatky, s ktorými sa program musí vysporiadať, sú nasledovné:

1. roztrhnutie čiary na krátkom úseku,
2. poskočenie čiary v určitých úsekoch o pixel vyššie alebo nižšie,
3. kombinácia roztrhnutia a posunu o pixel.



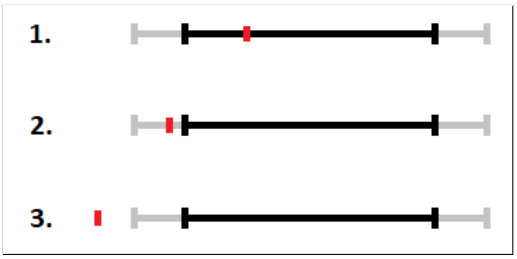
Obr. 4.1: Poškodenia hrán

Sčasti sú tieto poškodenia hrán opravené tým, že algoritmus používa 4 hranové detekcie. Vďaka tomu, ak niekde vznikne poškodenie po jednej hranovej funkcii, tak tento defekt je prekrytý výsledkom ďalších 3 detekcií.

4.1.2 Vodorovné a zvislé čiary

Ako ďalší krok je určenie vodorovných a zvislých čiar pomocou bodov hrán. Z každej detekcie hrán sa prahovaním určia pixely, ktoré sa považujú za body hrán. Z týchto bodov sa teda vytvárajú čiary, ktoré sa ukladajú do hashmap. Ako kľúč sa používa Y hodnota pre vodorovné a X hodnota pre zvislé čiary.

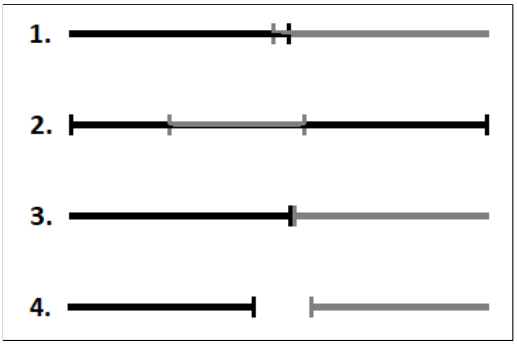
Algoritmus funguje nasledovne. Pre aktuálne analyzovaný bod sa podľa hodnoty jeho kľúča zoberú všetky už predtým vytvorené čiary z hashmapy. Následne sa porovnáva vzdialenosť bodu od týchto čiar. Ak je menšia ako určitá hodnota, ktorú som určil na základe testovania s mojimi dátami, tak sa upraví začiatok, alebo koniec takejto čiary na bod, ktorý aktuálne analyzuje algoritmus. V prípade, že sa bod nedá spojiť so žiadnou čiarou, ten sa uloží do hashmapy ako nová čiara. Následne sa zoberie ďalší bod a tento proces sa opakuje. Táto časť algoritmu zabezpečuje opravu čiar v prípade, že je na niektorom mieste roztrhnutá na krátkej vzdialenosti.



Obr. 4.2: Vzájomná poloha bodu a čiary

Na obrázku 4.2 môžeme vidieť 3 polohy bodu, v akých sa môže nachádzať bod vzhľadom na polohu čiary. Červenou farbou je znázornený bod, čiernou je zobrazená čiara a šedou farbou je znázornená čiara po natiahnutí jej začiatku a konca o povolenú veľkosť medzery medzi bodom a čiarou. Prípady 1 a 2 znázorňujú vzájomnú polohu, kedy algoritmus prehlási, že bod leží na čiare. V prípade 3 algoritmus prehlási, že bod na čiare neleží.

Po spracovaní výstupov všetkých hranových funkcií a naplnení hashmap sa vykonáva zlučovanie čiar. Zoberie sa čiara z hashmapy, ktorú sa algoritmus následne snaží spojiť s niektorou čiarou na rovnakej, o jednu vyššej a o jednu nižšej úrovni. Začiatok a koniec prvej čiary sa predĺži o veľkosť povolenej medzery medzi čiarami. Pre takto predĺženú čiaru a druhú čiaru z hashmapy sa kontroluje, či sa v niektorom bode prekrývajú. V takom prípade sa spoja a vznikne nová čiara. Tú sa algoritmus snaží spojiť s ďalšími čiarami v hashmape. V prípade, že nie je možné zlúčenie s ďalšou čiarou, je vrátená naspäť do hashmapy. Algoritmus následne zoberie ďalšiu čiaru v poradí a proces sa opakuje, až kým nepríde na koniec hashmapy. Táto časť algoritmu je rovnaká pre vodorovné a aj zvislé čiary. Vďaka tejto časti algoritmu sa vyrieši problém, kedy sa čiara na niektorom úseku posunie o pixel vyššie alebo nižšie. Zároveň sa týmto spôsobom odstráni problém, keď vznikne medzera v čiare. V prípade zlučovania čiar, ktoré neležia na rovnakej úrovni sa Y hodnota pre vodorovné a X hodnota pre zvislé čiary určí porovnaním ich dĺžok. Táto hodnota je teda určená podľa toho, ktorá z týchto dvoch čiar je dlhšia.



Obr. 4.3: Vzájomná poloha dvoch čiar

Na obrázku 4.3 sú znázornené všetky vzájomné polohy dvoch čiar. Sú tu znázornené prípady dvoch čiar, ktoré ležia na rovnakej úrovni. Ale rovnaké vzájomné polohy nastanú aj v prípade, kedy sa spájajú čiary na rozdielnych úrovniach. Algoritmus pred ich porovnaním predĺži konce jednej z nich o určitú vzdialenosť. Ide o povolenú medzeru medzi čiarami. Na obrázku sú teda čiary znázornené už po tomto predĺžení. Veľkosť povolenej medzery medzi bodom a čiarou a medzery medzi dvomi čiarami je rovnaká. Prvé 3 prípady znázorňujú polohu čiar, kedy ich je možné zlúčiť do jednej. Jedine vo štvrtom prípade sa čiary nedajú spojiť.

Takto zlúčené čiary sa ďalej ukladajú do poľa horizontálnych čiar a poľa vertikálnych čiar. Pred uložením sa vyradia tie čiary, ktoré nedosahujú dostatočnú dĺžku. Pre horizontálne čiary bola testovaním určená veľkosť 1/3 celkovej šírky snímky a pre vertikálne čiary zase 1/5 celkovej výšky snímky. Všetky kratšie čiary nie sú dostatočne dlhé na to, aby mohli byť prehlásené za potenciálne okraje hľadaného obrázka.

4.1.3 Obdĺžnik orezu

Poslednou fázou je z potenciálnych okrajov odhadnúť tie, ktoré by mohli tvoriť obdĺžnik, v ktorom by sa mal nachádzať hľadaný obrázok na snímke obrazovky. Keďže sa predpokladá, že obrázok zaberá celú šírku obrazovky, v prvom kole sa zoberú všetky čiary, ktoré sa aspoň jedným koncom dotýkajú okraja snímky. Zoznam takýchto čiar je usporiadaný podľa hodnoty Y pre horizontálne čiary a X pre vertikálne čiary. Ďalej sa tento zoznam rozdelí

na dva. V prvom zozname sa nachádzajú čiary, pre ktoré ich hodnota Y, respektíve X, je menšia ako polovica celkovej výšky snímky, respektíve jej šírky. V druhom zozname zostanú všetky čiary, ktoré túto podmienku nespĺňajú. Pozícia pixelu sa v tomto algoritme počíta od ľavého horného rohu. Prvý zoznam teda obsahuje potenciálny horný okraj obrázka a druhý zoznam zase potenciálny dolný okraj. Prvý zoznam sa začne prechádzať od konca a hľadá sa prvá čiara, ktorej oba konce sa dotýkajú okrajov obrazovky. Takáto čiara sa teda prehlási za horný okraj obrázka. V prípade, ak sa takáto čiara nenájde, zoberie sa prvá čiara od konca (čiže najbližšia čiara ku stredu snímky). Podobne sa hľadá aj dolný okraj obrázka, len s tým rozdielom, že sa zoznam prechádza od začiatku.

V prípade, že sa takýmto spôsobom nenájde horný okraj alebo dolný okraj obrázka, algoritmus prejde na hľadanie obdĺžnika vo vnútri snímky. Tu sa najprv hľadajú dve zvislé čiary, ktorých začiatočné body majú rovnakú hodnotu X a koncové body majú tiež rovnakú hodnotu X. Ak sa takéto čiary nájdu, hľadajú sa k nim vodorovné čiary, ktoré by prepájali ich začiatky a ich konce. Pomocou tejto časti algoritmu je zabezpečené nájdenie obrázka aj v prípade, ak by nezaberal celú šírku snímky.

Ak by sa ani takýmto spôsobom nepodarilo nájsť okraje obrázka, algoritmus orezové hrany nastaví na okraje snímky.

4.2 Inpainting

Aby bolo možné odstrániť text pomocou metódy Telea alebo algoritmu Navier-Stokes, je potrebné určiť masku oblasti, ktorá sa má odstrániť. Ideálne by bolo vymaskovať len oblasť textu. Aby ale bolo ovládanie pre užívateľa čo najjednoduchšie, v aplikácii si užívateľ vyberie obdĺžnikovú oblasť a tá sa následne nahradí. Výber metódy nie je automatický, ale je ponechané na užívateľovi, aby si vybral.

Kapitola 5

Implementácia a testovanie

Samotná aplikácia je implementovaná v jazyku Java. Pre implementovanie hranových funkcií a metódy Telea s Navier-Stokes pre inpainting som použil knižnicu OpenCV. Za účelom jednoduchšieho testovania a odladovania som vytvoril demo aplikáciu. Tá je spúšťaná pomocou buildovacieho programu ant.

Táto demo aplikácia slúži na zobrazenie jednotlivých krokov pri procese analýzy snímky. Ako je možné vidieť na obrázku 5.1, hlavná časť aplikácie obsahuje tri obrázky. Na prvej je vždy zobrazená originálna nezmenená snímka, ktorú program analyzuje. To, čo sa zobrazí na ostatných dvoch obrázkoch záleží od toho, aká časť algoritmu sa práve sleduje. V dolnej časti sa nachádzajú tlačidlá na preskakovanie a analyzovanie snímok, ktoré sa nachádzajú v priečinku „screenshots“. Vedľa tlačidiel „ďalšia snímka“ a „predchádzajúca snímka“ sa nachádza tlačidlo pre prepínanie zobrazovania výsledkov jednotlivých častí algoritmu.

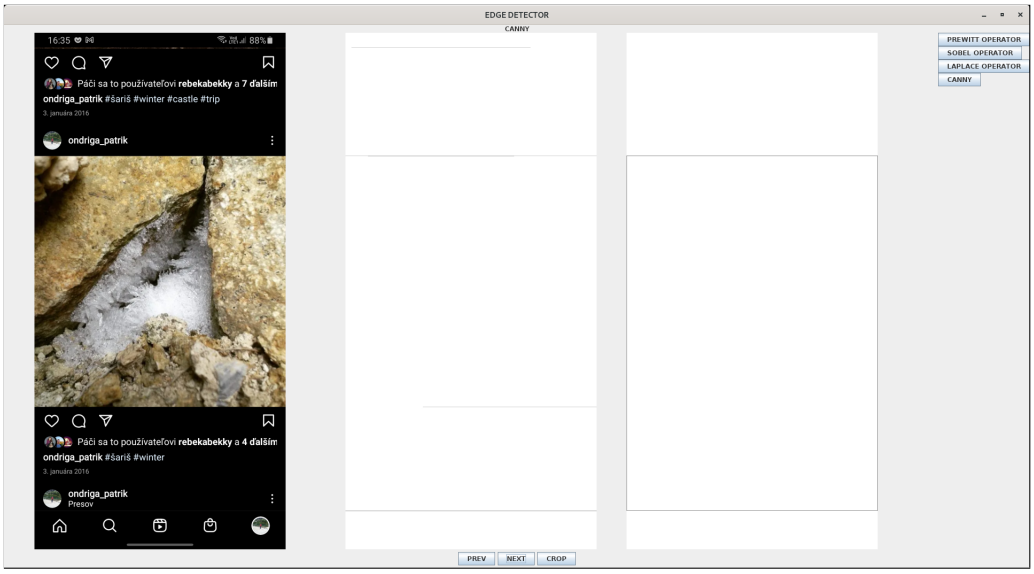
Demo aplikácia pri spustení zobrazuje výsledok niektorej zo štyroch hranových funkcií. Na druhom obrázku je zobrazená originálna snímka po spracovaní hranovej detekcie. Na treťom sa zobrazujú čiary, ktoré algoritmus návrhu orezu našiel po spracovaní takto upravenej snímky. Zároveň sa hrubou čiarou zobrazuje aj výsledný navrhovaný orez. Prepínanie medzi hranovými funkciami sa nachádza na pravej strane obrazovky.

Druhou sledovanou funkcionalitou je kombinovanie výsledkov hranových funkcií. Čiže to, ako algoritmus funguje ako celok. Na druhom obrázku sa vykreslia všetky čiary, ktoré program našiel pri kombinácii výstupov štyroch hranových detekcií. Na treťom obrázku je zobrazený iba samotný navrhovaný orez.

Pri ďalšom prepnutí sa taktiež sleduje správanie algoritmu ako celku, ale tentokrát sa nezobrazuje iba navrhovaný orez, ale už aj pôvodný obrázok orezaný podľa tohto návrhu.

Poslednou sledovanou funkcionalitou je inpainting. Na druhom obrázku sa zobrazuje maska, podľa ktorej sa bude nahrádzať oblasť v snímke. Táto maska je v demo aplikácii pevne daná a nie je možné ju bez zásahu do kódu meniť. Na treťom obrázku sa ďalej zobrazuje pôvodná snímka s nahradenou oblasťou pomocou metódy Telea alebo Navier-Stokes.

Na obrázku 5.1 je zobrazené spracovávanie screenshotu mojej fotografie zverejnenej na sociálnej sieti Instagram. Je na ňom možné vidieť, že finálny orez nebol určený správne. Je to spôsobené nenájdením celej dolnej hrany obrázka. V mobilnej aplikácií by ale užívateľovi stačilo posunúť spodnú hranu orezu a funkcia „pomocník posunu“ by mu pomohla s presným umiestnením spodnej strany orezu.



Obr. 5.1: Grafické rozhranie demo aplikácie

Okrem vizuálneho zobrazovania výsledkov algoritmu je možné pre demo aplikáciu spustiť jednotkové testy. Tie otestujú správnosť implementácie programu. Kontrolujú správnosť jednotlivých tried, ako je uvedené v sekcii testovanie 5.2. V konzole sa počas testovania priebežne zobrazujú výsledky jednotlivých testovacích sád. Po dokončení sa zobrazí počet neúspešných testov. V prípade neúspechu sa vypíše chybová hláška testu, ktorý neprešiel.

Demo aplikácia je schopná ďalej vygenerovať *jar* súbor. Ide o skompilovaný program, ktorý obsahuje iba triedy potrebné pre automatický orez a inpainting. Čiže sa v ňom nenachádza grafické prostredie demo aplikácie. Taktiež sa v ňom nenachádza ani knižnica OpenCV, nakoľko táto knižnica má druhú verziu optimalizovanú pre mobilné aplikácie pre operačný systém Android. Tento *jar* súbor je vkladaný ako knižnica do programu pre mobilné užívateľské prostredie.

Pri implementácii demo aplikácie som použil kód, ktorý prevádza formát *Mat* z knižnice OpenCV do formátu *BufferedImage*. Toto prevádzanie sa využíva výhradne iba v demo aplikácii pri zobrazovaní. *Jar* súbor túto časť kódu neobsahuje. Autorom týchto riadkov je Krishna Kasyap [7].

Užívateľské prostredie bolo implementované v prostredí Android Studio. Boli použité komponenty, ktoré ponúka toto vývojové prostredie. Rámček, ktorý sa používa pre vymedzenie oblasti orezu a zároveň sa používa pre vymedzenie oblasti odstránenia, je jediným komponentom, ktorý bolo potrebné vytvoriť od základu.

5.1 Odladovanie algoritmu

Pre potreby odladenia programu som použil program VisualVM. Pri kontrolovaní rýchlosti spúšťam demo program v režime profiling. Ten sa hneď po spustení pozastaví, až kým nie je stlačená klávesnica enter. Je to z toho dôvodu, aby som mohol vo ViasualVM nájsť proces, na ktorom beží program a mohol nastaviť jeho sledovanie. Potom sa spustí proces hľadania orezu snímok v priečinku, pričom sa priebežne zobrazuje počet spracovaných obrázkov a celkový počet, ktorý sa nachádza v priečinku.

Rýchlosť výpočtu ovplyvňujú vlastnosti obrázka. Tým je napríklad rozlíšenie. Ale aj rôznorodosť farieb a intenzita ich striedaní v snímke. Preto som vybral 33 rôznych snímok z troch rozličných mobilných zariadení s inou veľkosťou obrazovky. Rýchlosť ovplyvňuje aj zariadenie, na ktorom je testovanie spúšťané. Preto som sa rozhodol vykonať tento test vždy päťkrát. Všetky testy som spúšťal na rovnakom zariadení, zatiaľ čo na ňom bežali v pozadí rovnaké aplikácie. Zariadenie, na ktorom bol test vykonaný, je Asus x555l. Obsahuje procesor Intel Core i3-5010U s frekvenciou 2.1GHz. Operačná pamäť má veľkosť 8GB. Program bežal na operačnom systéme Linux. Obrázky boli načítavané z SSD disku.

Tabuľka 5.1 zobrazuje hodnoty, ktoré som nameral v prvotnej verzii programu, následne v druhej a posledný stĺpec obsahuje aktuálne hodnoty rýchlosti programu.

Tabuľka 5.1: Čas potrebný na výpočet orezových hrán

	1. verzia		2. verzia		finálna verzia	
počet fotiek	33	1	33	1	33	1
priemer	140,68s	4,263s	72,876s	2,208s	66s	2s
smerodajná odchýlka	1,722s	0,052s	0,79s	0,024s	1,903s	0,058s

Prvá verzia programu potrebovala priemerne 4 sekundy na výpočet jedného obrázka. To je priveľmi dlhý čas na to, aby sa dala aplikácia prehlásiť za rýchlu. Preto bolo potrebné lokalizovať a optimalizovať miesta v kóde, na ktorých sa stráca najviac času. Po dôkladnej analýze v programe VisualVM som našiel metódu, ktorá vykonávala prechod cez všetky body na obrázku. Tento prechod sa ale vykonával dvakrát. Najprv riadok po riadku pre nájdenie vodorovných čiar a následne stĺpec po stĺpci na nájdenie zvislých. Prechádzanie obrázkom je operácia lineárnej zložitosti, týmto kusom kódu som túto zložitosť zdvojnásobil.

V druhej verzii som tento problém vyriešil odstránením druhého prechodu a zavedením hashmap pre vodorovné a zvislé čiary. Táto úprava zrýchlila program približne o polovicu.

Posledným výrazným vylepšením bolo zavedenie kontroly, či bod, ktorý sa bude najbližšie kontrolovať, neleží na niektorej čiare, ktorá sa našla v predchádzajúcej hranovej funkcii. Metóda, ktorá zaberá najviac času je metóda knižnice OpenCV. Táto metóda zabezpečuje prístup k jednotlivým pixelom na koordinátach, ktoré sú jej predané. Preto sa kontroluje, či koordináty, pre ktoré by sa kontroloval pixel, neležia na predtým nájdenej čiare. Ak áno, program sa posunie na koniec takejto čiary a pokračuje sa v kontrolovaní pixelov. Toto vylepšenie obmedzuje počet volaní spomínanej knižnej metódy.

Finálna verzia programu potrebuje priemerne 2 sekundy na odhad orezových čiar. Podľa môjho názoru je táto rýchlosť dostačujúca, ale aj napriek tomu som sa rozhodol dať možnosť užívateľovi zvoliť si rýchlosť na úkor presnosti. Pridal som preto možnosť zvoliť si, koľko hranových funkcií sa použije, nakoľko pre každý výstup takejto funkcie musí program vždy prechádzať celým obrázkom a hľadať v ňom vodorovné a zvislé čiary.

5.2 Testovanie

Pre zabezpečenie správneho fungovania algoritmu som vytvoril sadu jednotkových testov jednotlivých častí. Počas pridávania ďalších funkcionalít, s ktorými som pôvodne nerátal, som medzi testy pridal aj regresné testy. Tieto testy sú rozdelené do týchto kategórií:

- testy triedy reprezentujúcej bod na snímke,
- testy triedy reprezentujúcej čiaru,

- testy triedy reprezentujúcej obdĺžnik.

Pre overenie celkovej funkcionality som spolu s regresnými testami vytvoril aj integračné testy. Tie sa zase nachádzajú v týchto kategóriách:

- testy algoritmu pre hľadanie čiar na obrázku a opravovanie poškodení spôsobených hranovými funkciami,
- testy algoritmu pre poskladanie obdĺžnika, ktorý reprezentuje okraje obrázka v snímke,
- testy pre vykonávanie návrhu orezových hrán a následného orezania snímky.

Vďaka priebežnému testovaniu sa mi podarilo vyhnúť sa množstvu malých chýb vo finálnej verzii programu. Najväčší problém, ktorý mi pomohli regresné testy odhaliť bol ten, keď som prekonvertoval maticu obrázka z typu *Mat* na *BufferedImage*. Počas optimalizovania algoritmu som si všimol, že získanie hodnoty pixelu z matice vo formáte *Mat* je časovo zdĺhavejšie, ako získanie tejto hodnoty z matice vo formáte *BufferedImage*. Po implementovaní konvertora sa algoritmus javil, akoby tento zásah nemal vplyv na počet nájdených hrán. Až po spustení testov som si všimol, že niektoré neprešli. Tie odhalili, že táto úprava spôsobila nájdenie iných hrán, ako tomu bolo pred úpravou. Pri bližšom preskúmaní som zistil, že pri konvertovaní matice obrázka dochádzalo k miernej zmene hodnôt pixelov. Nanešťastie som neprišiel na dôvod týchto zmien a musel som túto časť kódu odstrániť.

Kapitola 6

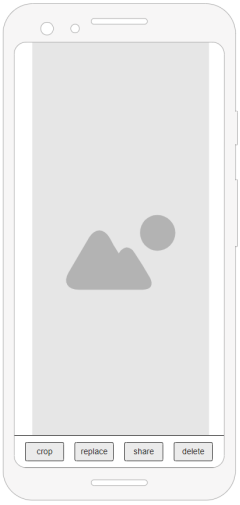
Užívateľské rozhranie

Užívateľské prostredie je cielené pre mobilné zariadenia s operačným systémom Android. Pri programovaní je potrebné zvoliť si vhodnú verziu. Pri prieskume v čase začatia tvorenia užívateľského prostredia som zvolil verziu 8.0 Oreo. Výberom tejto verzie som pokryl viac ako 80% všetkých aktívnych zariadení.

Dôležitým cieľom projektu je, aby sa dala aplikácia jednoducho ovládať. Preto je užívateľské prostredie rozdelené na štyri obrazovky. Všetky obrazovky ako aj rozloženie komponentov na nich je zobrazené v prílohe A.

Pre prístup k obrázkom uloženým v mobilnom zariadení, ale aj pre prístup k ich zmenšeným verziám bol použitý *MediaStore framework* spomínaný v sekcii 2.7. Aby bolo možné pristupovať k obrázkom a upravovať ich, je potrebné od užívateľa získať oprávnenie k prístupu do úložiska zariadenia. Pri prvom spustení aplikácie je užívateľ vyzvaný na udelenie tohto oprávnenia. V prípade, že by ho zamietol, aplikácia sa zavrie s oznámením o tom, že pre fungovanie aplikácie je toto povolenie nevyhnutné.

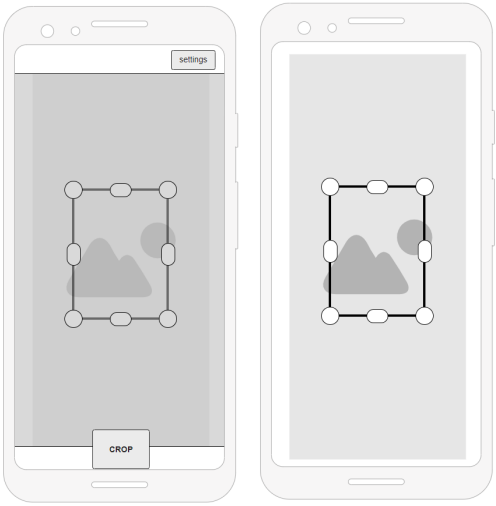
Prvou obrazovkou je galéria. V nej sa zobrazujú všetky obrázky uložené v telefóne. Aby prostredie galérie zbytočne nevyťažovalo výpočtový výkon zariadenia, zobrazené obrázky majú nižšiu kvalitu. Aby sa pamäť zbytočne nezahlcovala, sú načítané len tie obrázky, ktoré sa aktuálne zobrazujú na obrazovke. To znamená, že akonáhle sa užívateľ posunie nižšie, obrázky navrchu obrazovky, ktoré už nie je vidieť, sa z pamäte odstránia. Cieľom galérie je zobraziť čo najviac obrázkov naraz, aby užívateľ nemusel dlho listovať. Zároveň nesmú byť obrázky príliš malé, aby bolo možné identifikovať, čo sa na nich nachádza. Zvolil som teda mriežkový variant, kedy sa v jednom riadku naraz nachádzajú tri obrázky.



Obr. 6.1: Obrazovka „detail obrázka“

Po zvolení obrázka sa užívateľ presunie na druhú obrazovku. Tá nesie názov „detail obrázka“ (obr. 6.1). V nej je možné obrázok približovať a prezerať si ho v plnej kvalite. Táto funkcionalita je možná vďaka použitiu komponentu *Subsampling Scale Image View* 2.9. V spodnej lište sa nachádzajú tlačidlá pre akcie s obrázkom pre rýchly prístup pravákov aj ľavákov. Túto lištu je možné skryť a zase zobraziť kliknutím na obrazovku. Na nej sa nachádza automatický orez obrázka zo snímky obrazovky, odstraňovanie textu, zdieľanie obrázka a odstránenie obrázka.

Po zvolení možnosti nájdenia automatického orezu je užívateľ presunutý na tretiu obrazovku „automatický orez“ (obr. 6.2). Zároveň sa spustí algoritmus hľadania. Počas tohto procesu sa užívateľovi zobrazuje, na koľko percent je proces hľadania orezu hotový. Pôvodná implementácia zobrazovala iba točiace sa koliesko. Užívateľ ale takýmto spôsobom nemal žiadne informácie o tom, ako dlho sa bude vykonávať hľadanie a pri zložitejších obrázkoch mohlo dôjsť k pocitu, že sa program zasekol. Následne sa zobrazí obrázok spolu s orezovým rámčekom. Užívateľ si ďalej hrany rámčeka môže posunúť, alebo len kliknúť na tlačidlo orezať a výrez obrázka sa uloží. Tlačidlo je umiestnené v lište na dolnej časti obrazovky pre ľahký prístup. V hornej lište napravo sa nachádza tlačidlo pre otvorenie nastavení orezu. V nich je možné vypnúť pomocné čiary pre orez, nastaviť rýchlosť orezu na úkor presnosti, alebo spustiť proces hľadania orezu znova. Pod pojmom pomocné čiary orezu sa myslí funkcionalita, ktorá napomáha užívateľovi zastaviť sa na pozícii potenciálnych hrán, ktoré boli algoritmom určené, ale neboli použité na finálny návrh orezu. Rýchlosť orezu na úkor presnosti je nastavenie, kedy sa určí, koľko hranových funkcií sa použije pri procese hľadania orezu.



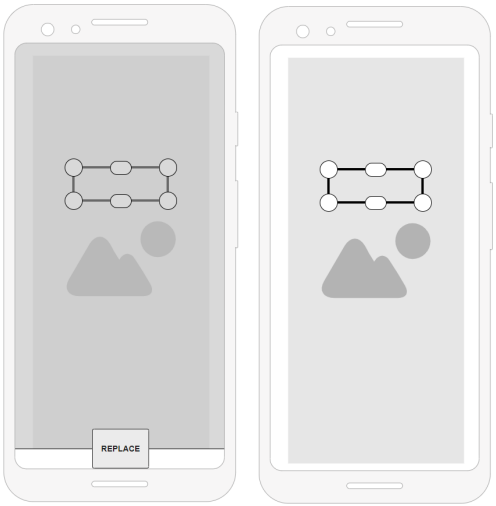
Obr. 6.2: Obrazovka „automatický orez“

Odstránenie textu z obrázka je plne vykonávané užívateľom. Toto sa nachádza vo štvrtej obrazovke s názvom „nahradiť“ (obr. 6.3). Užívateľ pomocou rovnakého orámovania, aké je používané pri oreze obrázka, určí obdĺžnikovú oblasť, ktorú chce z obrázka odstrániť. Po stlačení tlačidla na spodnej lište sa mu ponúknu dve možnosti, ktorými je možné oblasť nahradiť. Prvá je vypočítaná pomocou algoritmu Telea a druhá zase algoritmom Navier-Stokes.

Tlačidlá v spodných lištách pre ukladanie obrázka v režime orezávania a odstraňovania textu sú pomerne veľké a prekrývajú časť snímky. Preto sú počas upravovania rámčekov skryté. Užívateľ ich môže zobraziť kliknutím na obrazovku. Kým je tlačidlo zobrazené, rámčeky nie je možné upravovať. Priestor za tlačidlom je zatmavený, čím užívateľ má nadobudnúť pocit, že sa s rámčekom nebude dať manipulovať. Opätovné kliknutie na obrazovku skryje tlačidlo a sprístupní upravovanie rámčekov.

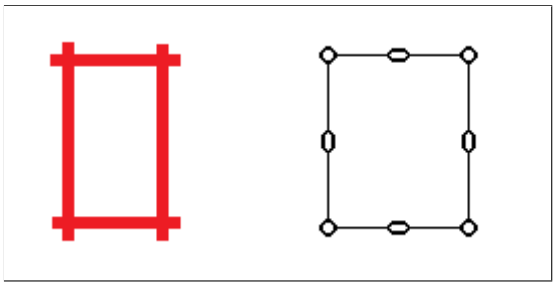
Pre ukladanie upraveného obrázka po odstránení textu alebo výrezu obrazovky si užívateľ môže vybrať, či nahradí pôvodný obrázok, alebo sa vytvorí nový. Následne je tento obrázok zobrazený v obrazovke „detail obrázka“. Je to z toho dôvodu, aby mohol užíva-

teľ rýchlo upravovať obrázok a poprípade ho ďalej zdieľať bez nutnosti jeho vyhľadania v galérii.



Obr. 6.3: Obrazovka „nahradiť“

Ohraničenie, ktoré sa používa pre orez, ale aj pre označenie oblasti odstránenia, bolo pôvodne navrhnuté ako štyri červené čiary. Od okrajov obrazovky až po toto ohraničenie bol polopriehľadný šedý tieň. Tento návrh ale mal niekoľko problémov. Vždy sa dalo posúvať iba s jednou hranou. Napríklad nebolo možné chytiť ohraničenie za pravý horný roh a tým posúvať naraz hornú hranu spolu s pravou. Tiež nebolo možné posúvať orámovanie po snímke bez zmeny veľkosti. Čiary museli byť hrubé, aby sa dali jednoducho chytiť prstom, čo ale spôsobovalo veľké prekrývanie obrázka a teda bolo obťažné trafiť sa presne na hrany v obrázku. Červená farba čiary dokonca splývala na špecifických obrázkoch, ktoré obsahovali červenú farbu. Druhá verzia tohto ohraničenia sa skladá z tenkej bielej čiary zo všetkých štyroch strán. Od okrajov obrazovky až po tieto biele čiary je polopriehľadný šedý tieň. Táto kombinácia zabezpečuje jasný okraj ohraničenia pri tmavých obrázkoch, ale aj pri svetlých. Vďaka tenkej čiare ohraničenie neprekrýva veľkú oblasť obrázka a teda je jej umiestnenie jednoduchšie. Pre posúvanie a zmenu veľkosti ohraničenia v ôsmich smeroch sú v rohoch a v strede každého okraja ohraničenia umiestnené body úchytu. Tieto body sú dostatočne veľké, aby bolo možné s nimi manipulovať a zároveň neprekážajú pri vizuálnom určení vodorovných a zvislých hrán na obrázku. Oproti predošlej verzii sa ohraničenie dá presúvať po obrazovke bez toho, aby sa jeho rozmery museli meniť. Táto funkcionalita je užitočná najmä pri používaní tohto ohraničenia pri výbere oblasti na odstránenie.



Obr. 6.4: Ohraničenie orezu a oblasti odstránenia

V ľavej časti obrázka 6.4 je zobrazené pôvodné ohraničenie a v pravej zase finálna verzia s bodmi úchytu.

Aplikácia má implementovanú multijazyčnosť. Aktuálne je preložená len do jazykov Slovenčina a Angličtina. V budúcnosti ale nie je problém pridať aj ďalšie jazyky. V prípade, že by bola aplikácia spustená na zariadení s iným jazykom ako ponúka, zvolí sa jazyk anglický.

Po nainštalovaní sa aplikácia registruje v zariadení užívateľa pre zobrazovanie súborov typu obrázok. To znamená, že užívateľ pri vytvorení snímky obrazovky nemusí otvárať aplikáciu a hľadať túto snímku v galérii. Stačí ju rozkliknúť a okrem ostatných galérií sa v zozname zobrazí aj táto aplikácia. Po jej vybratí sa obrázok zobrazí rovno v obrazovke „detail obrázka“.

6.1 Zdieľanie a testovanie

Aby som mohol počas vývoja testovať užívateľské prostredie, bolo potrebné nájsť spôsob, akým by som túto aplikáciu mohol jednoducho zdieľať a inštalovať aj na iné zariadenia bez nutnosti ich pripájať k môjmu počítaču. Preto som aplikáciu nahral na *Obchod Play* v nastavení testovania. Ďalšou výhodou tohto riešenia je, že testeri pri inštalácii nemuseli meniť žiadne nastavenia v ich zariadení, ako by tomu bolo pri zdieľaní inštalácie priamo. Takýto spôsob mi zabezpečoval aj jednoduché preinštalovanie na novšiu verziu, nakoľko sa testerovi v *Obchod Play* automaticky zobrazovali najnovšie aktualizácie.

Pre priebežné testovanie som za pomoci dvoch hlavných testerov pracoval nasledovne. Priebežne som svoje rozhodnutia konzultoval s prvým testerom, takže aplikáciu poznal veľmi dobre. Vďaka tomu, že používa iné mobilné zariadenie so staršou verziou Androidu ako som používal pri vývoji, bol som schopný odhaliť množstvo skrytých problémov. Jedným z nich bol prípad, kedy pre dialógové okná používam knižnicu *Material design*. Táto knižnica zabezpečuje to, aby sa jej komponenty prispôsobovali podľa nastavenia zariadenia. V mojom prípade nastal problém v tom, že pre jeden z nadpisov som použil fixne bielu farbu. Nakoľko ja používam tmavý mód v zariadení, aj pozadie tohto dialógového okna bolo čierne. Lenže tester používa svetlý mód, tým pádom bolo pozadie biele, čo viedlo k tomu, že nadpis nebolo vidieť.

Druhý tester dostal o aplikácii iba základné informácie. Vedel, na čo má aplikácia slúžiť, ale užívateľské prostredie videl prvýkrát, až keď si aplikáciu nainštaloval. Vďaka tomuto testu som sa dozvedel, ako veľmi je táto aplikácia intuitívna. Po spätnej väzbe som musel vyriešiť tieto záležitosti, ktoré by som si bez tohto testu naslepo neuvedomil.

- V obrazovke „automatický orez“ a „nahradiť“ sú málo výrazné nápisy, a tým pádom si ich užívateľ nemusí všimnúť, čo môže viesť k tomu, že užívateľ nebude vedieť, čo má urobiť.
- Na obrazovke „automatický orez“ je málo výrazné tlačidlo pre nastavenia.
- Názov nastavenia „pomocné čiary“ je zmätočný a pre užívateľa nejednoznačný.
- V nastaveniach je prednastavené používanie dvoch hranových funkcií. Ukázalo sa, že na začiatku je lepšie užívateľovi prednastaviť najväčšiu presnosť, ktorú aplikácia ponúka aj za cenu dlhšieho výpočtu (pre testera bola rýchlosť pri používaní všetkých štyroch funkcií dostatočná).

- Bolo mi odporučené, aby som pridal buď nápovede, alebo krátky tutoriál, kde by bol užívateľ oboznámený s prostredím.
- Tester namietal, že v nastavení jazyka Slovenčina zostal názov aplikácie anglický.

V rámci vyriešenia týchto pripomienok som sa rozhodol implementovať tutoriál pre obrazovky „detail obrázka“, „automatický orez“ a „nahradiť“. Obrazovku „galéria“ som vynechal, pretože jej ovládanie nie je zložité a nelíši sa od iných bežne používaných galérií obrázkov.

Tutoriál je implementovaný ako dialógové okno, ktoré obsahuje text vysvetľujúci funkcionalitu jednotlivých prvkov aplikácie a *gif* obrázok poskytujúci vizuálnu prezentáciu textu. Jednotlivé animované *gif* obrázky som vytváral v štýle makety. Tento dizajn bol vybraný, pretože užívateľovi jednoznačne ukazuje nástroje aplikácie bez zbytočného rušenia rôznofarebných kombinácií aplikácie a obrázkov, ktoré sú v nej upravované. Pre znázornenie miesta stlačenia obrazovky som zvolil červený krúžok. Pre načítanie *gif* obrázkov som použil rozhranie Glide 2.8.

Pre obrazovku „detail obrázka“ sú tutoriálom objasnené funkcie jednotlivých tlačidiel, nakoľko v aplikácii tieto tlačidlá nereprezentuje text, ale ikony.

Obrazovka „automatický orez“ obsahuje okrem hlavnej funkcionality aj nastavenia. Preto je pre ňu tutoriál najdlhší a najdôležitejší. Opisuje spôsob zobrazovania a skrývania líšt s tlačidlami. Vysvetľuje význam každého z nich. Tutoriál ďalej vysvetľuje prácu s orámovaním orezu. V rámci nastavení objasňuje význam znižovania počtu hranových funkcií. Asi najdôležitejšie je vysvetlenie prepínača „pomocné čiary“. Toto nastavenie nebolo dostatočne jasné ani jednému testerovi.

V rámci obrazovky „nahradiť“ sa zobrazujú typy, ako zobrazovať a skrývať lišty s tlačidlami podobne ako pri obrazovke „automatický orez“. Taktiež je rovnaké aj vysvetlenie používania orámovania oblasti odstránenia, nakoľko sa používa rovnaký komponent ako pri oreze. Hlavným prínosom ale je to, že užívateľ ešte pred použitím tohto nástroja zistí, že najprv musí vyznačiť oblasť, ktorú chce odstrániť a následne mu program ponúkne dva obrázky, ktorými bude táto oblasť nahradená.

Tieto tutoriály sa užívateľovi automaticky spustia pri prvom príchode na jednotlivé obrazovky. V prípade, že by si ich chcel v budúcnosti znova prejsť, v ľavom hornom rohu bola pridaná ikona so symbolom otáznika. Toto tlačidlo ale chýba pre „detail obrázka“, nakoľko si nemyslím, že by funkcionalita tejto obrazovky bola natoľko zložitá, aby bolo nutné si tutoriál prezrieť viackrát.

Podľa odporúčania testera som počet hranových funkcií prednastavil na štyri, čo je maximálny počet. Taktiež som zmenil názov v nastavení orezu z „pomocné čiary“ na „pomocník posunu“. Tento názov lepšie vystihuje účel tejto pomôcky. Vďaka tutoriálu už nebolo potrebné zväčšovať, alebo inak zvýrazňovať nadpisy, nakoľko užívateľ už získal vedomosti o funkcionalite aplikácie. Aj napriek výhradám testera som názov aplikácie nepreložil v prípade nastavenia jazyka zariadenia na Slovenčina.

Kapitola 7

Výkonnostné testovanie

Testovanie aplikácie sa dá rozdeliť na tri samostatné okruhy. Prvým je sledovanie rýchlosti spracovania snímky obrazovky. Druhým je percentuálna úspešnosť algoritmu pre správne určenie okrajov orezu. Tretím je získanie spätnej väzby od užívateľov počas používania tejto mobilnej aplikácie, či sa podarilo dosiahnuť jednoduchosť a intuitívnosť.

7.1 Rýchlosť výpočtu

Pre odmeranie rýchlosti výpočtu som použil rovnaký postup ako pri procese odladovania spomínanom v sekcii 5.1. Opäť bol použitý program VisualVM.

7.1.1 Rýchlosť hľadania vzhľadom na veľkosť snímky

Účelom tohto testu bolo zistiť, ako sa bude líšiť rýchlosť nájdenia hrán orezu pri rozličnej veľkosti snímky. Táto veľkosť je daná rozlíšením displeja. Preto bola použitá tretia sada obrázkov. Táto sada obsahuje dve skupiny. V prvej sa nachádzajú snímky, kde každá je tvorená **1 065 600** pixelmi. Druhá skupina so snímkami rovnakých obrázkov má väčšie rozlíšenie. Každá snímka je tu tvorená **4 262 400** pixelmi. To znamená, že snímky v druhej skupine sú presne štyrikrát väčšie, ako snímky v prvej skupine. Vyhľadávanie pre každú skupinu bolo vykonané päťkrát. Je to z toho istého dôvodu, ako pri predchádzajúcom teste. Chcel som týmto eliminovať následky výkyvov výpočtového výkonu zariadenia, na ktorom test prebiehal. Test bol uskutočnený na tom istom zariadení pri rovnakých podmienkach ako v predchádzajúcom teste.

Tabuľka 7.1: Čas potrebný na výpočet orezových hrán

	1. skupina		2. skupina	
počet fotiek	20	1	20	1
priemer	20,939s	1,05s	82,388s	4,12s
smerodajná odchýlka	0,609s	0,03s	0,96s	0,048s

Z tabuľky 7.1 môžeme vyčítať, že pri prvej skupine nájdenie orezu pre jednu snímku trvalo približne jednu sekundu. V druhej skupine výpočet jednej snímky trval približne štyri sekundy. To je štvornásobný nárast času. Vzhľadom na to, že snímky v druhej skupine majú štvornásobnú veľkosť, je možné prehlásiť, že zložitosť algoritmu vzhľadom na veľkosť snímky je lineárna.

7.2 Úspešnosť návrhu orezu

Aby som mohol prehlásiť, že algoritmus funguje, je potrebné otestovať jeho úspešnosť pri návrhu orezu. Pre tento test bola vytvorená sada, ktorá obsahuje 100 snímok. Z toho 78 snímok má jasné hrany medzi obrázkom a prostredím sociálnej siete. Ostatných 22 snímok má minimálne jednu hranu splývajúcu s prostredím a v niektorých prípadoch voľným okom skoro nerozpoznateľnú.

Počas testu sa kontroluje, či bol orez určený správne, alebo či boli aspoň správne nájdené hrany, len pre finálny orez neboli použité.

Rozhodol som sa tento test vykonať dvakrát. Najprv algoritmus používal všetky 4 hranové funkcie. V opakovanom teste som algoritmu nastavil, aby použil iba 2. Pri tomto opakovanom teste by sa malo prejaviť, či bolo potrebné implementovať algoritmus so 4 hranovými detekciami.

Tabuľka 7.2: Úspešnosť určenia správneho orezu pri použití 4 hranových funkcií

	orez správny	orez nesprávny, hrany nájdené	orez nesprávny, hrany nenájdené
78 snímok	48	27	3
22 snímok	2	5	15
spolu	50	32	18

Ako je možné vidieť v tabuľke 7.2, algoritmus dokázal celkovo určiť 50% orezov snímok správne. Tento výsledok by nebol dostačujúci v prípade, ak by sa nerátalo s tým, že tento orez môže užívateľ následne upraviť. V takom prípade mu pomôžu práve nájdené hrany, ktoré neboli použité pre finálny návrh orezu. Vtedy je vhodné vyhodnotiť, pre koľko snímok algoritmus dokázal nájsť hrany obrázka. Ak zoberieme čisto iba obrázky, kde dokáže hrany určiť bezproblémovo človek, tak zo 78 snímok iba 3 neboli správne určené. Pričom pri snímkach, kde som nepredpokladal nájdenie hrán, algoritmus dokázal správne nájsť hrany v 7 snímkach a dokonca určiť správny orez v 2 z 22 snímok.

Počas tohto testu sa ukázalo, v akých prípadoch má algoritmus problém buď nájsť hrany obrázka, alebo problém určiť, ktorá z hrán by mala byť použitá pre orez. Boli potvrdené úvodné domnienky pri návrhu algoritmu. Hrany boli často prerušené v prípade, že prostredie sociálnej siete malo napríklad čiernu farbu a na kraji obrázka sa nachádzal čierny predmet. Veľkým problémom boli obrázky, v ktorých sa rýchlo menili farby. Toto striedanie vytváralo množstvo falošných hrán, ktoré boli neskôr určené za okraj orezu. Na obrázku 7.1 je možné vidieť príklad tohto problému. Na ľavej časti sa nachádza časť pôvodného obrázka, v strednej časti sú znázornené určené hrany a v pravej časti je znázornený horný okraj navrhovaného orezu.

Obr. 7.1: Falošné hrany

Jedným z častých problémom, pri ktorom algoritmus určoval nesprávne hrany orezu bolo, keď autor obrázka pridal vlastné orámovanie.

Tabuľka 7.3: Úspešnosť určenia správneho orezu pri použití 2 hranových funkcií

	orez správny	orez nesprávny, hrany nájdené	orez nesprávny, hrany nenájdené
78 snímok	43	32	3
22 snímok	1	6	15
spolu	44	38	18

Výsledky opakovaného testu pri použití 2 hranových funkcií sú zobrazené v tabuľke 7.3. Z nich je možné vyčítať, že algoritmus funguje správne aj pri nižšom počte použitých hranových detekcií. Ak sa pozeráme čisto iba na určovanie potenciálnych hrán orezu, výsledok je rovnaký ako pri použití 4 hranových funkcií. Iba v 18 prípadoch neboli nájdené všetky hrany. Čo sa týka určenia správneho orezu, tak tu sa algoritmus pomýlil o 6 snímok viacej. Je to z toho dôvodu, že hrany, ktoré sa nájdu, sú často kratšie, čo v týchto 6 prípadoch spôsobilo to, že sa pre finálny orez vybrala iná čiara.

7.3 Test užívateľského prostredia

Ďalším atribútom tejto práce bolo vytvoriť jednoduché, intuitívne prostredie pre mobilné zariadenia s operačným systémom Android. Hlavnou myšlienkou je, aby bolo možné aplikáciu používať napríklad aj jednou rukou. Použitie takejto aplikácie je napríklad pri orezávaní snímok pri preprave mestskou hromadnou dopravou. Pre otestovanie, či sa tento bod práce podarilo splniť, bolo potrebné rozšíriť aplikáciu medzi užívateľov a zozbierať od nich spätnú väzbu. Pre zdieľanie aplikácie som použil rovnaký spôsob ako pri zdieľaní kvôli testovaniu počas vývoja. Vďaka tomu, že je aplikácia nahratá na *Obchod Play*, mi stačilo jednotlivým užívateľom poslať jej link a oni si ju mohli stiahnuť pohodlne jedným kliknutím. Pre zozbieranie spätnej väzby som vytvoril google dotazník. Celý dotazník aj s odpoveďami sa nachádza v prílohe B. Aplikácia spolu s dotazníkom bola verejne dostupná. Nakoniec bol dotazník vyplnený celkovo 19 užívateľmi.

Účelom prvej časti dotazníka bolo zistiť, či existuje u užívateľov potreba mať takúto aplikáciu nainštalovanú. Z 19 ľudí iba jeden jediný nepoužíva žiadnu sociálnu sieť, kde je možné zverejňovať obrázky. Z ostatných 18 jedincov si nikdy žiadnu snímku obrazovky sociálnej siete nerobí opäť iba jeden. Až 66% opýtaných robí snímku obrazovky pri prezeraní sociálnych sietí pravidelne a teda takáto mobilná aplikácia by im mohla pomôcť. Z toho iba traja užívatelia uviedli, že používajú aplikáciu, ktorá vykonáva automaticky návrh orezu. Z týchto údajov je teda možné potvrdiť, že existencia takejto aplikácie má zmysel.

Druhá časť dotazníka bola zameraná na používanie samotnej aplikácie. Cieľom bolo zistiť, či ovládacie prvky boli pre užívateľov intuitívne a ovládanie aplikácie prostredníctvom nich bolo komfortné.

Prvou preverovanou funkcionalitou bol tutoriál. Až 44,4% opýtaných uviedlo, že bez tutoriálu by si niektoré funkcie aplikácie nevšimli, alebo by nevedeli, ako ich majú použiť. 55,6% by pravdepodobne aj bez tutoriálu bolo schopných aplikáciu používať bez väčších problémov aj v prípade, keby sa v nej tutoriál nenachádzal. Nikto z opýtaných ale neuviedol, žeby mu v tutoriáli chýbalo vysvetlenie nejakej funkcionality, alebo žeby aj napriek tutoriálu nebol schopný sa v aplikácii orientovať.

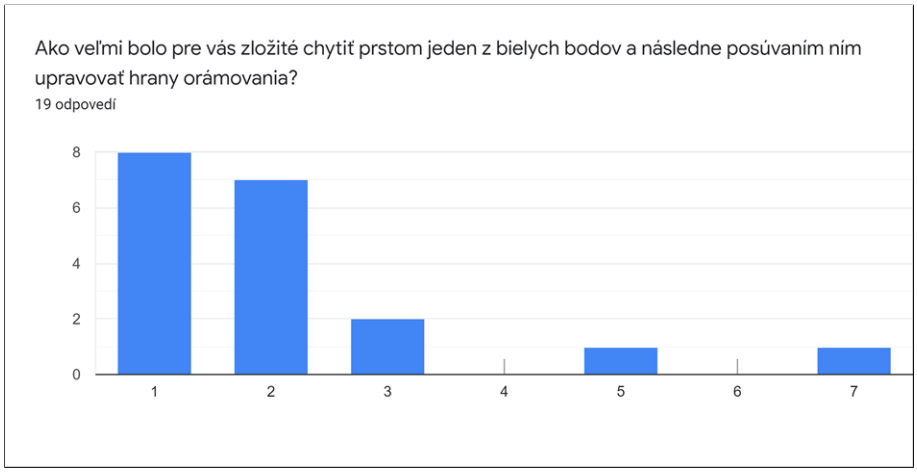
Ďalšia séria otázok bola zameraná na hlavnú funkcionalitu aplikácie a to automatický návrh orezu. Pri otázke týkajúcej sa rýchlosti, akou bol navrhnutý orez snímky obrazovky,

až 57,9% opýtaných uviedlo, že proces hľadania bol pre nich dostatočne rýchly. 36,8% uviedlo, že by privítali, keby návrh orezu bol nájdený o niečo rýchlejšie. Len jeden jediný užívateľ nebol spokojný s rýchlosťou tejto aplikácie. Pri otázke, či pri používaní aplikácie v budúcnosti by si ju prenastavili na rýchlejší spôsob hľadania orezu na úkor presnosti, uviedlo 31,6% žeby toto nastavenie urobilo a 68,4% by nechalo pôvodné nastavenie, čiže používanie všetkých hranových funkcií. Odpovede na otázku „ako často bol návrh orezu určený správne“ ukázali, že až 84,2% užívateľov bolo spokojných s návrhmi orezov, ktoré algoritmus navrhol.

Funkcionalita, ktorá mala pre užívateľov veľký prínos, je „pomocník posunu“. Až pre 94,7% opýtaných bola užitočná a nedostali sa do situácie, kedy by im prekážala. Ostatných 5,3% uviedlo, že funkcionalita je užitočná, ale v ojedinelých prípadoch si ju museli vypnúť.

Druhým nástrojom aplikácie je odstraňovanie malých oblastí z obrázka. V tejto funkcionalite užívateľ pomocou obdĺžnikového orámovania vyberie oblasť, ktorú chce odstrániť. Orámovanie v tomto nástroji má nastavenú obmedzenú maximálnu šírku a výšku. Preto bolo vhodné prostredníctvom dotazníka zistiť, či toto obmedzenie nebránilo užívateľom pri jej používaní. Až 68,4% uviedlo, že s maximálnou veľkosťou nemali problém. Na konci dotazníka sa objavila pripomienka, že v prípade, keď chceli odstrániť text zasahujúci do celej šírky obrazovky, museli ho odstraňovať na dvakrát. Preto by bolo vhodné do budúcna odstrániť obmedzenie maximálnej šírky orámovania.

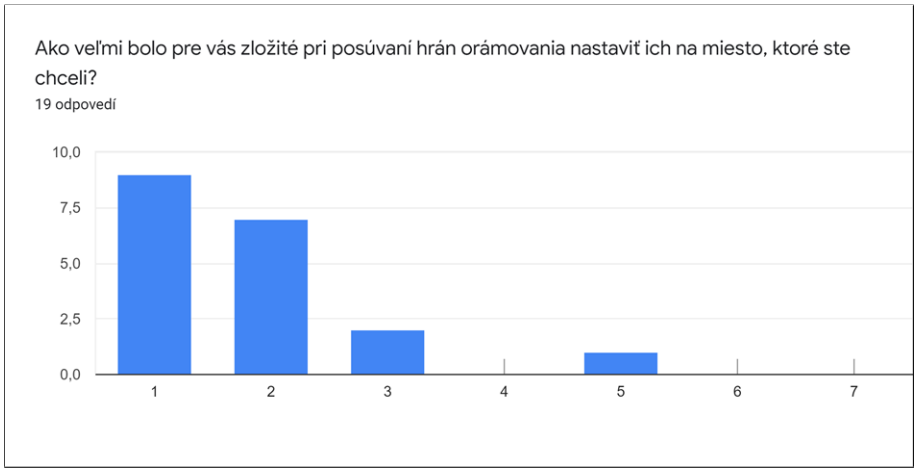
Ovládací prvok orámovanie, ktoré sa používa pri výbere orezu, ale aj pri výbere oblasti pre odstránenie, bolo ďalšou témou otázok. Pri otázke ohľadom minimálnej veľkosti, akú je možné nastaviť pre orámovanie, traja užívatelia uviedli, že by v niektorých prípadoch potrebovali menšie orámovanie, ako im aplikácia povolila. Tu ale nevidím priestor pre možnosť umožniť menšie orámovanie, nakoľko by došlo k prekrývaniu úchytných bodov, ktorými je možné posúvať okraje orámovania.



Obr. 7.2: Graf - Jednoduchosť používania bodov úchytu pre orámovanie

Obrázok 7.2 znázorňuje graf, v ktorom opýtaní uvádzali na stupnici od 1 po 7 ako veľmi bolo pre nich jednoduché trafiť sa prstom na niektorý bod úchytu a tým posúvať niektorú hranu orezu. Číslo 1 reprezentuje bezproblémové ovládanie a číslo 7 zase najhoršie. Z tohto grafu teda môžeme usúdiť, že práca s úchytmi orámovania pre väčšinu opýtaných nepredstavovala žiadny veľký problém.

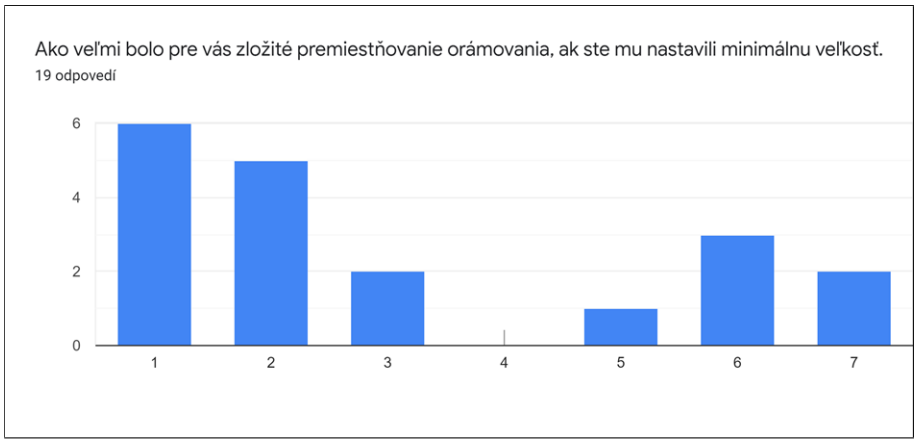
V prvej verzii orámovania boli rámčeky príliš hrubé a to komplikovalo umiestnenie jeho hrán na správne miesto. Pri finálnej verzii orámovania je teda namieste zistiť, či sa nevyskytuje podobný problém. Na grafe na obrázku 7.3 je ale vidieť, že užívatelia s umiestnením hrán orámovania nemali väčšie problémy.



Obr. 7.3: Graf - Posúvanie hrán orámovania na správne miesto

Pri presune okrajov som sa obával, že by mohol nastať problém posunúť ich až na okraje snímky. Predpokladal som, že pri používaní mobilu v ochrannom obale by mohlo dôjsť k prekážaniu posunu až na okraj displeja. Táto obava sa nenaplnila a až na jednu výnimku opýtaní uviedli, že s posunom až na okraj snímky nebol väčší problém. Z toho viac ako 50% na stupnici od 1 do 7 zadalo hodnotu 1, čo reprezentuje, že posun bol bezproblémový.

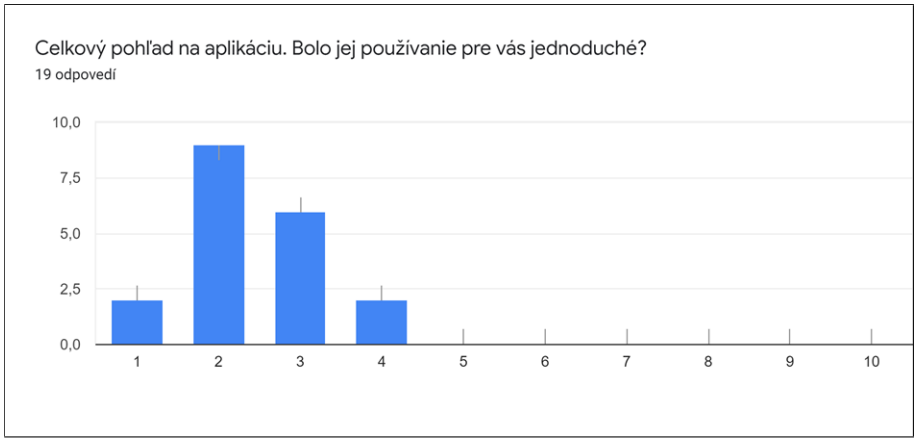
Posledná otázka k orámovaniu bola zameraná na jeho ovládanie v prípade, že mu bola nastavená minimálna veľkosť. Z grafu na obrázku 7.4 je možné vidieť, že už v súčasnom nastavení nie je ovládanie vôbec ideálne a niečo cez 30% užívateľov uviedlo, že jeho manipulovanie bolo skôr obťažné. Odpovede na túto otázku len potvrdzujú to, že súčasné riešenie orámovania nemôže mať povolené ešte menšie minimálne rozmery, ako sú nastavené práve teraz.



Obr. 7.4: Graf - Posúvanie orámovania pri minimálnej veľkosti

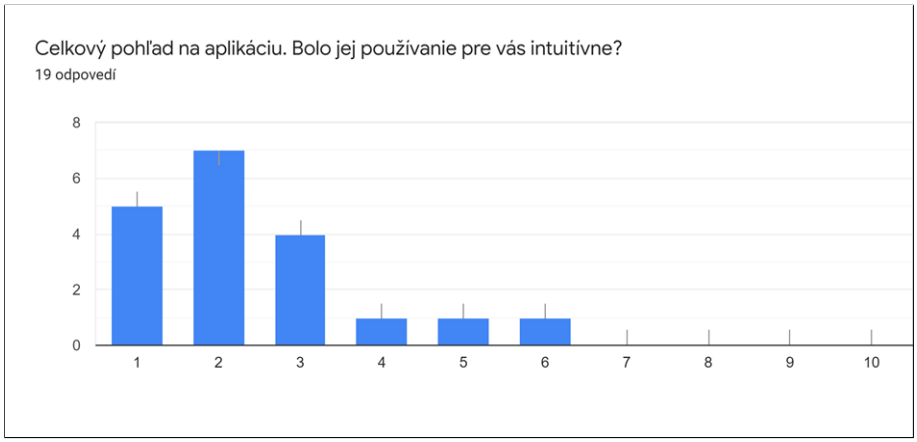
V ďalšej časti dotazníka mali užívatelia zhodnotiť, ako dobre sa im pracovalo v jednotlivých obrazovkách aplikácie. Každú obrazovku mali ohodnotiť číslom od 1 po 7, kde 1 reprezentuje ovládanie bolo bezproblémové a 7 ovládanie bolo obťažné. Obrazovka „galéria“ dostala priemerné hodnotenie 1,37. Obrazovka „detail obrázka“ získala priemerné hodnotenie 1,95. Obrazovka „návrh orezu“ bola priemerne ohodnotená číslom 1,53. Obrazovka „nahradiť“ dostala najhoršie priemerné hodnotenie a to 2,37. Pod toto hodnotenie sa podpísalo to, že práca s orámovaním v tejto obrazovke závisí iba od užívateľa bez pomoci algoritmu. Druhým hlavným problémom je to, že na tejto obrazovke sa prevažne pracuje s orámovaním s malou veľkosťou, čo sa v predchádzajúcich otázkach ukázalo ako mierne nekomfortné.

Užívatelia boli taktiež požiadaní, aby zhodnotili ovládanie aplikácie ako celku. Tu bola stupnica od 1 až po 10. Tak ako aj pri predchádzajúcich otázkach číslo 1 reprezentuje bezproblémové ovládanie a číslo 10 zase obťažné. Priemerné hodnotenie ovládateľnosti aplikácie je 2,42. Hodnotenie jednotlivých užívateľov je zobrazené na grafe na obrázku 7.5.



Obr. 7.5: Graf - Obťažnosť ovládania aplikácie

Posledná sada otázok bola zameraná na intuitívnosť aplikácie. Účastníci dotazníka ju mali za úlohu ohodnotiť v každej otázke číslom od 1 po 7, kde 1 reprezentuje intuitívne ovládanie a 7 veľmi mätúce. Pri otázke ohľadom intuitívnosti nadpisov, názvov nastavení a označenia tlačidiel dali užívatelia priemerné hodnotenie 1,89. Intuitívnosť vyskakovacích okien dosiahla priemerné hodnotenie 1,74. Poslednou otázkou dotazníka mali užívatelia ohodnotiť celkovú intuitívnosť celej aplikácie. Tá mala stupnicu väčšiu, a to od 1 po 10, kde hodnota 1 reprezentuje intuitívne ovládanie a 10 zase veľmi mätúce. Aplikácia dostala priemerné hodnotenie 2,42. Hodnotenie intuitívnosti celkovej aplikácie je zobrazené v grafe na obrázku 7.6.



Obr. 7.6: Graf - Intuitívnosť ovládania aplikácie

Na konci celého dotazníka mali užívatelia možnosť napísať svoje postrehy a pripomienky. Najčastejšie sa opakovala požiadavka na pridanie možnosti preskočiť tutoriál. Užívateľovi sa pri prvom navštívení obrazoviek aplikácie zobrazil ich tutoriál, ktorým si musel prejsť. Vzhľadom na percento užívateľov, ktorí uviedli, že tutoriál nepotrebovali, som túto pripomienku zapracoval a do aplikácie som pridal tlačidlo „preskočiť“. Dobrým nápadom je pridanie možnosti zumovať v režime orezu. Takýmto spôsobom by bolo možné umiestňovať hrany orezu presnejšie. Implementovanie tohto vyžaduje komplexnejší zásah do aplikácie, a preto je toto len možnosť ako aplikáciu vylepšiť do budúcna. Jeden užívateľ uviedol, že názov tlačidla v nastaveniach orezu „obnoviť“ v ňom vyvolal dojem, že sa tým resetujú nastavenia orezu. Bolo preto vhodné toto tlačidlo premenovať na niečo výstižnejšie. Tlačidlu som teda zmenil názov na „opätovne navrhnúť“.

Výsledkom tohto testu je zistenie, že sa podarilo vytvoriť relatívne intuitívnu aplikáciu, ktorej používanie je jednoduché. Taktiež sa ukázalo, že rýchlosť hľadania orezu je pre užívateľov prijateľná a úprava navrhnutého orezu pohodlná. Vďaka pripomienkam sa odhalili možnosti, ktorými by mohla byť aplikácia do budúcna vylepšená.

Kapitola 8

Záver

Cieľom projektu bolo vytvoriť intuitívnu mobilnú aplikáciu v prostredí Android. Jej používanie malo byť jednoduché a pohodlné s použitím len jednej ruky. Hlavnou funkcionalitou tejto aplikácie mal byť nástroj na automatické navrhovanie orezu snímok obrazovky. Prostredníctvom tohto nástroja si mal byť užívateľ schopný orezať snímku z niektorej sociálnej siete za pomoci jedného kliku. V prípade zlého určenia mal mať užívateľ možnosť tento orez upraviť. Druhou funkcionalitou aplikácie mal byť nástroj na odstraňovanie textu z obrázka.

Vzhľadom na výstupy jednotlivých testov môžem konštatovať, že úvodná myšlienka práce bola splnená. Vďaka pridaniu tutoriálu sa podarilo jednoduchou formou informovať nových užívateľov o prvkoch aplikácie, čím jej používanie sa stáva intuitívnym. Kombinácia algoritmu návrhu orezu a nástroja „pomocník posunu“ je orezávanie snímok obrazovky jednoduché, a to aj napr. pri cestovaní mestskou hromadnou dopravou, kedy užívateľ nemá stabilné okolie. Rýchlosť spracovávania snímok sa ukázala ako dostatočná a vďaka možnosti nastavenia, ktoré určuje koľko hranových funkcií sa použije pre odhad orezu, si užívateľ môže túto rýchlosť vylepšiť. Podarilo sa splniť aj požiadavku na vyretušovanie textu prostredníctvom funkcie odstránenie oblasti z obrázka.

Prácou na tomto projekte som získal vedomosti o počítačovom videní. Naučil som sa, akým spôsobom sa spracovávajú obrázky pri hľadaní objektov v nich. Pri tvorbe užívateľského rozhrania som získal poznatky potrebné na umiestnenie komponentov tak, aby bolo ich používanie pre užívateľa intuitívne. Ďalej som sa naučil vytvárať aplikácie pre mobilné zariadenia s operačným systémom Android. Novou skúsenosťou bolo aj zdieľanie aplikácie prostredníctvom *Obchod Play*.

Vďaka údajom z dotazníka by som na práci mohol ďalej vylepšiť ovládanie orámovania vďaka funkcii priblíženia na špecifické miesto na obrázku. Pre obrazovku „galéria“ by som ďalej chcel do budúcna implementovať možnosť zobraziť len tie obrázky, ktoré sú uložené v priečinku, ktorý si užívateľ vyberie. Poprípade graficky predeliť jednotlivé obrázky podľa dátumu ich vytvorenia. Čo sa týka rýchlosti nájdenia orezu, tá by sa dala vylepšiť za pomoci technológie JNI. To v podstate znamená, že by som musel časť kódu, pre ktorú jej vykonanie trvá najdlhšie, naprogramovať v jazyku C a vložiť ju do Android aplikácie. To ale znamená prispôsobenie celého algoritmu.

Literatúra

[1] Dharampal a Mutneja, V. Methods of Image Edge Detection: A Review. *Journal of Electrical & Electronic Systems* [online]. 2015, zv. 04, [cit. 2022-03-22]. DOI: 10.4172/2332-0796.1000150. ISSN 2332–0796.

[2] Ding, L. a Goshtasby, A. On the Canny edge detector. *Pattern recognition*. Elsevier. 2001, zv. 34, č. 3, s. 721–725. ISSN 0031 - 3203.

[3] Google. *Access media files from shared storage* [online]. 2022 [cit. 2022-04-20]. Dostupné z: https://developer.android.com/training/data-storage/shared/media.

[4] Jain, A. K. *Fundamentals of Digital Image Processing*. USA: Prentice-Hall, Inc., 1989. ISBN 0133361659.

[5] Jain, R., Kasturi, R., Schunck, B. G. et al. *Machine Vision*. McGraw-Hill New York, 1995. ISBN 0–07–032018–7.

[6] Judd, Sam. *Glide* [online]. 2022 [cit. 2022-05-25]. Dostupné z: https://github.com/bumptech/glide.

[7] Kasyap, K. *How to convert OpenCV Mat object to BufferedImage object using Java?* [online]. Englewood, NJ, USA: Tutorials Point, 2020 [cit. 2022-04-12]. Dostupné z: https://www.tutorialspoint.com/how-to-convert-opencv-mat-object-to-bufferedimage-object-using-java.

[8] Kaur, P. a Singh, J. A Study on the Effect of Gaussian Noise on PSNR Value for Digital Images. *International Journal of Computer and Electrical Engineering*. IACSIT Press. 2011, zv. 3, č. 2, s. 319, [cit. 2022-03-17].

[9] Kumar, Nishant. *Digital Image Processing Basics* [online]. 2021 [cit. 2022-04-20]. Dostupné z: https://www.geeksforgeeks.org/digital-image-processing-basics/.

[10] Maini, R. a Aggarwal, D. H. *Study and Comparison of Various Image Edge Detection Techniques* [online]. 2001 [cit. 2022-03-09]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.301.927&rep=rep1&type=pdf.

[11] Melin, P., Gonzalez, C. I., Castro, J. R., Mendoza, O. a Castillo, O. Edge-Detection Method for Image Processing Based on Generalized Type-2 Fuzzy Logic. *IEEE Transactions on Fuzzy Systems*. 2014, zv. 22, č. 6, s. 1515–1525, [cit. 2022-02-20]. DOI: 10.1109/TFUZZ.2013.2297159.

[12] Morrissey, David. *Subsampling Scale Image View* [online]. 2020 [cit. 2022-04-10]. Dostupné z: https://github.com/davemorrissey/subsampling-scale-image-view.

[13] OpenCV. *About* [online]. 2022 [cit. 2022-04-18]. Dostupné z: https://opencv.org/about/.

[14] OpenCV. *Image Inpainting* [online]. 2022 [cit. 2022-04-15]. Dostupné z: https://docs.opencv.org/4.x/df/d3d/tutorial_py_inpainting.html.

[15] Panigrahi, M., Mahakud, R., Samantaray, M. a Mohapatra, S. K. Comparative analysis of different Edge detection techniques for biomedical images using MATLAB. *Engineering and Scientific International Journal (ESIJ)*. 2014, [cit. 2022-03-10]. ISSN 2394–7179.

[16] Saif, J. A. M., Hammad, M. a Alqubati, I. Gradient Based Image Edge Detection. *International Journal of Engineering and Technology*. 2016, zv. 8, s. 153–156, [cit. 2022-03-25]. DOI: 10.7763/IJET.2016.V6.876. ISSN 2227 - 524X.

[17] Sharma, A. a Jaswal, S. Analysis of Sobel Edge Detection Technique for Face Recognition. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)* [online]. 2015, zv. 4, [cit. 2022-03-25]. ISSN 2278 - 1323.

[18] Verma, R. a Ali, J. A Comparative Study of Various Types of Image Noise and Efficient Noise Removal Techniques. *International Journal of Advanced Research in Computer Science and Software Engineering*. 2013, zv. 3, č. 10, s. 617–622, [cit. 2022-03-29]. ISSN 2277-128X.

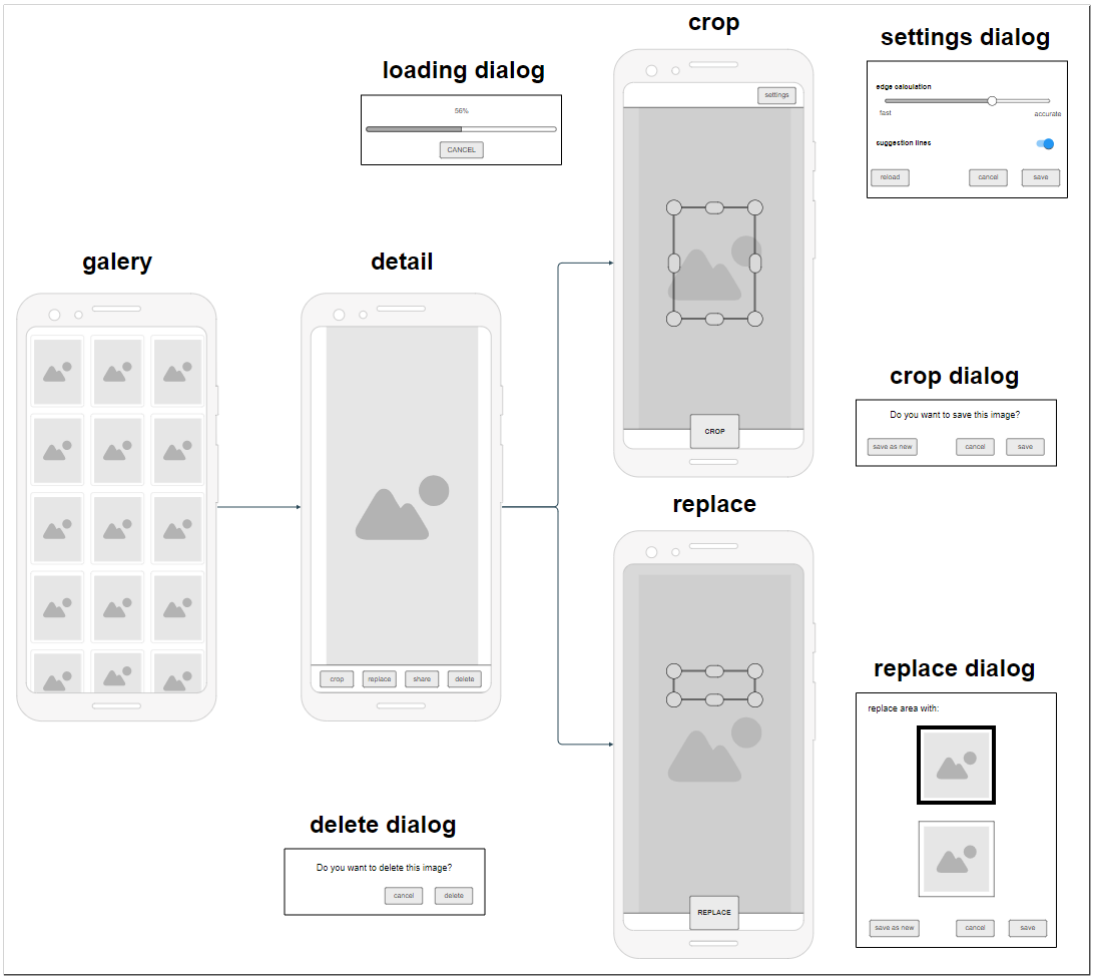
[19] Vijayarani, S. a Sakila, A. A Performance Comparison of Edge Detection Techniques for Printed and Handwritten Document Images. *International Journal of Innovative Research in Computer and Communication Engineering* [online]. 2016, zv. 4, č. 5, s. 8327–8337, [cit. 2022-04-05]. ISSN 2320-9801.

[20] Wanjari, M. T., Kalaskar, K. D. a Dhore, M. P. Document Image Segmentation Using Edge Detection Method. *International Journal of Computer & Mathematical Sciences IJCMS*. 2015, zv. 4, [cit. 2022-04-25]. ISSN 2347-8527.

[21] Yu, J., Lin, Z., Yang, J., Shen, X., Lu, X. et al. Generative Image Inpainting with Contextual Attention. In: *Proceedings of the IEEE conference on computer vision and pattern recognition* [online]. 2018, s. 5505–5514 [cit. 2022-04-25]. ISBN 9781538664209.

Príloha A

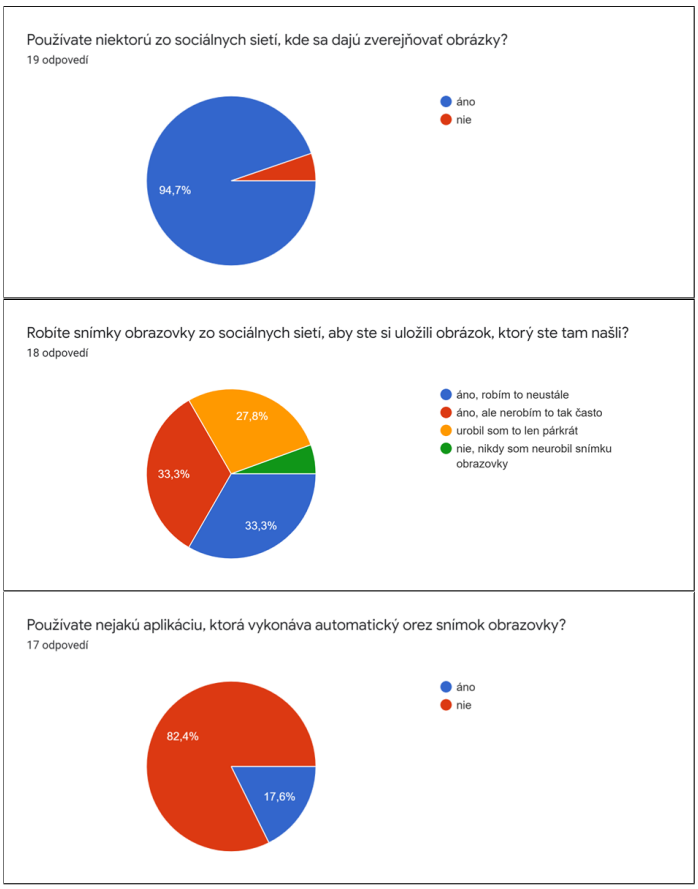
Návrh mobilného užívateľského rozhrania



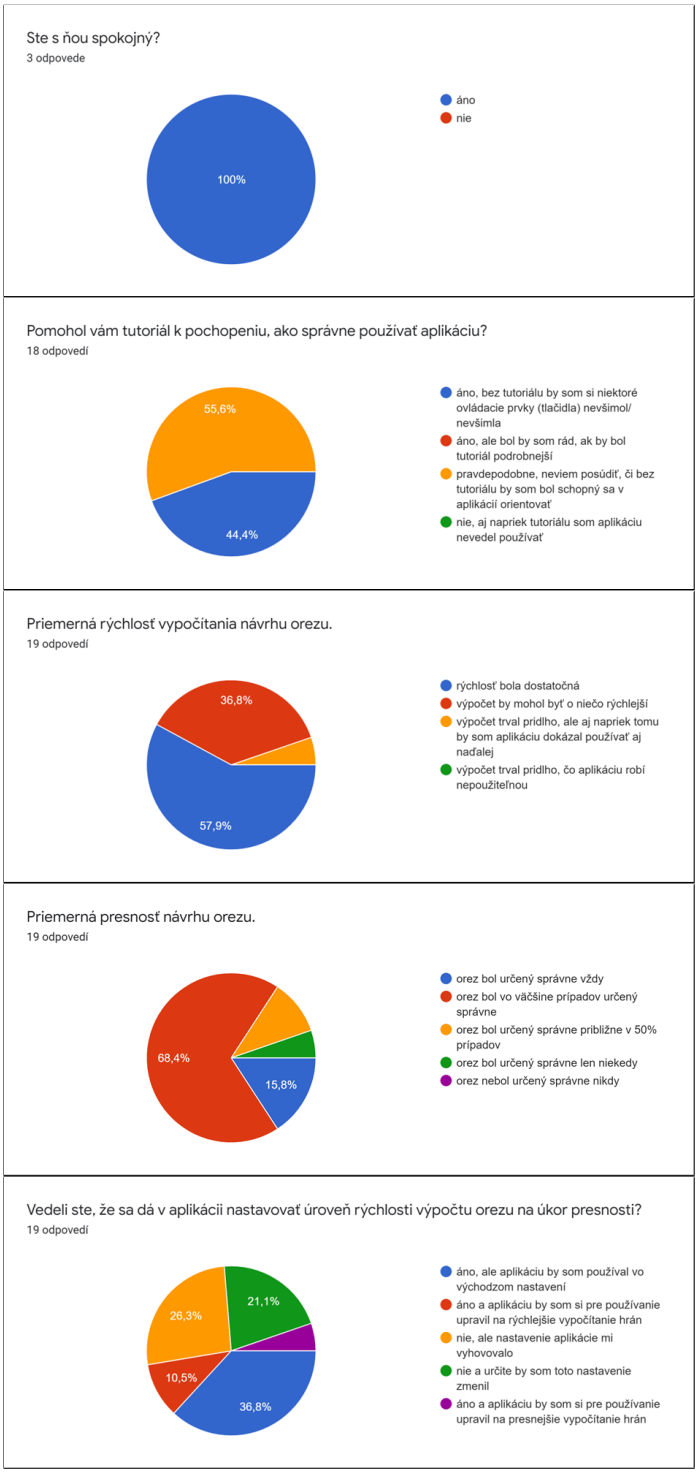
Obr. A.1: Návrh užívateľského rozhrania

Príloha B

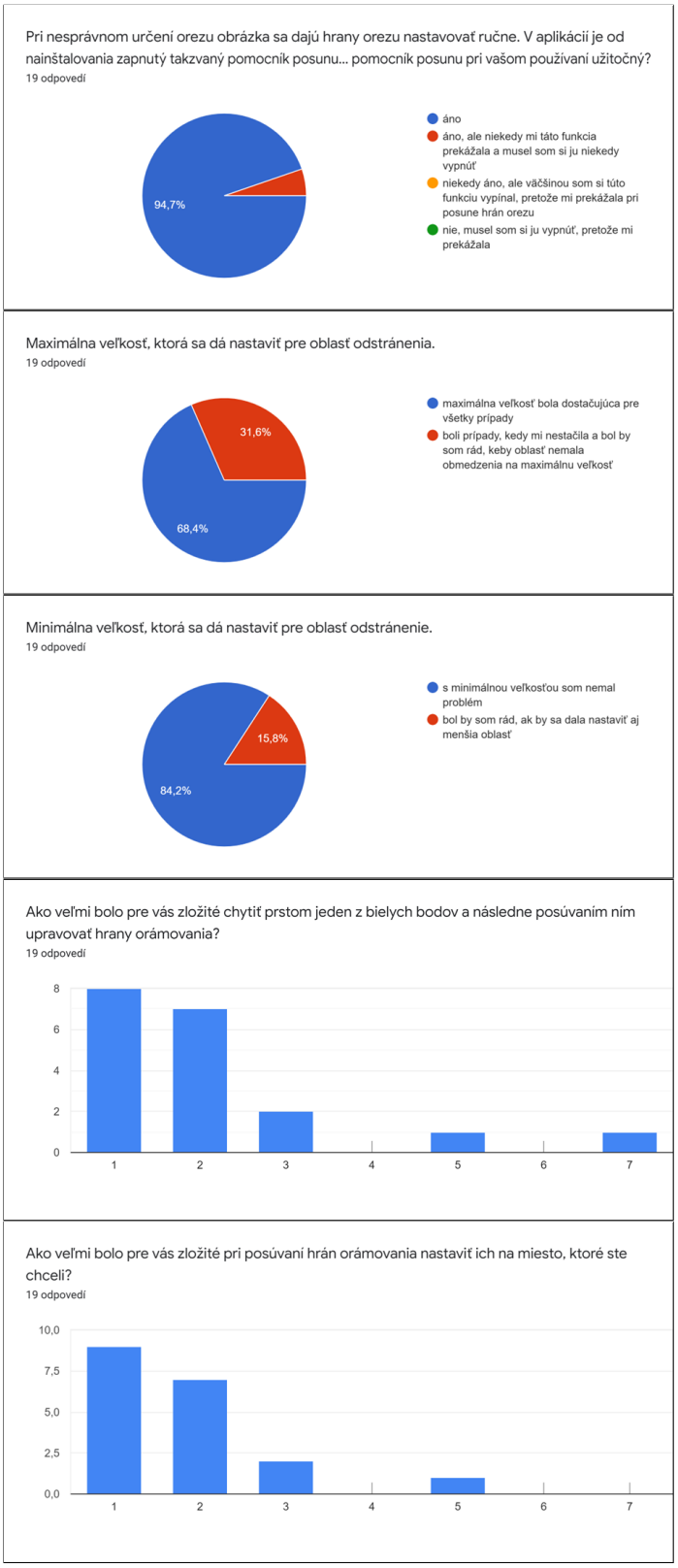
Dotazník



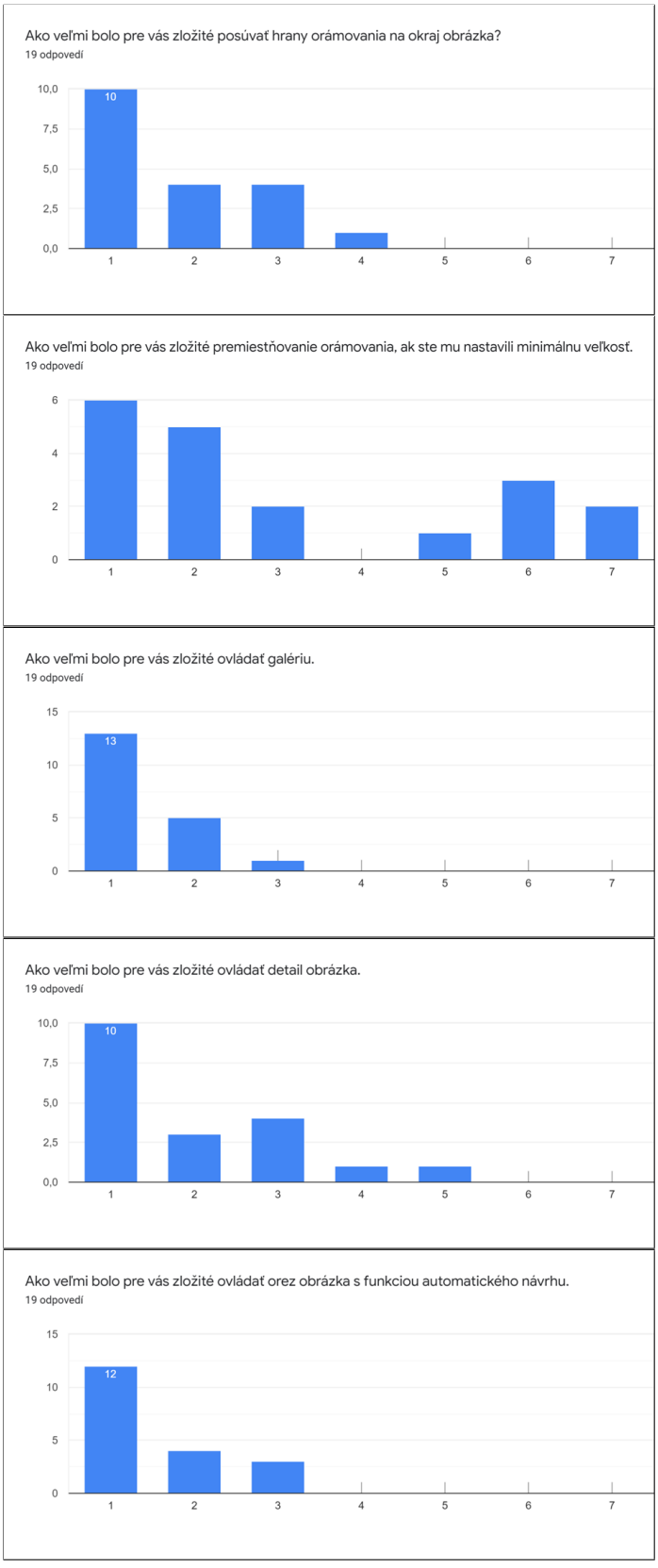
Obr. B.1: Prvá skupina otázok



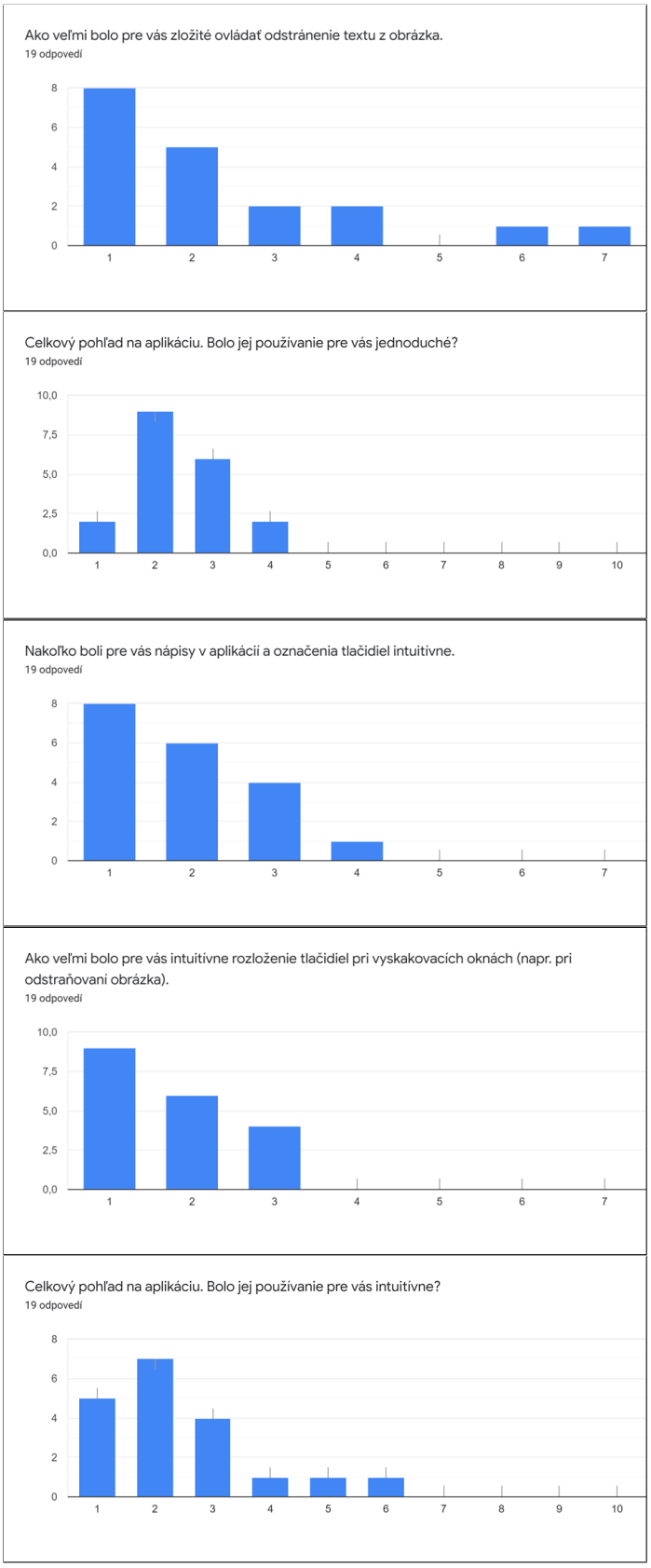
Obr. B.2: Druhá skupina otázok



Obr. B.3: Tretia skupina otázok



Obr. B.4: Štvrtá skupina otázok



Obr. B.5: Piata skupina otázok