Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Mobilní aplikace pro rozpoznávání registračních značek vozidel

Plzeň 2020 Stanislav Král

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta aplikovaných věd

Akademický rok: 2019/2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Stanislav KRÁL**
Osobní číslo:	**A17B0260P**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informatika**
Téma práce:	**Mobilní aplikace pro rozpoznávání registračních značek vozidel**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s technologiemi a frameworky pro vývoj multiplatformních mobilních aplikací, které umožňují využití technik počítačového vidění a rozpoznávání znaků (např. Flutter, OpenCV, Firebase, apod.).
2. Navrhněte a s využitím vhodného frameworku implementujte multiplatformní mobilní aplikaci, která bude snímat scénu, detekuje v ní registrační značky motorových vozidel a rozpozná text na registrační značce, který uloží v lokální databázi.
3. Aplikaci důkladně otestujte v reálném provozu, ověřte, že snímá registrační značky i za horších světelných podmínek, apod.
4. Vše řádně zdokumentujte v průvodním textu práce.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce: **Ing. Kamil Ekštein, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **7. října 2019**
Termín odevzdání bakalářské práce: **7. května 2020**

L.S.

Doc. Dr. Ing. Vlasta Radová
děkanka

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 15. října 2019

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V textu práce jsou použity názvy produktů, software, firem apod., které mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

V Plzni dne 6. května 2020

Stanislav Král

Poděkování

Děkuji panu Ing. Kamilu Ekšteinovi, Ph.D za ochotu při odborném vedení bakalářské práce a cenné rady s jejím vypracováním.

Abstract

The main goal of this bachelor thesis is to design and implement a method that will apply computer vision approaches and techniques to recognize registration plates of vehicles appearing in digital images. The next goal was to develop a multiplatform mobile application which would use this method to analyze images coming from the device's camera in real time. The theoretical part of this thesis introduces the characteristics of registration plates and then focuses on certain techniques from the computer vision field that could be applied to create an efficient registration plate recognizer. Such multiplatform mobile application was created by using the Flutter framework and the Firebase ML Kit library while the description of the development of the application is presented in the practical part. At last, the performance of the method in varying lighting conditions was measured.

Abstrakt

Hlavním cílem této bakalářské práce je navrhnout a implementovat metodu, která používá techniky z oblasti počítačového vidění k rozpoznávání registračních značek vozidel v reálném čase. Dalším cílem této práce bylo vytvořit multiplatformní mobilní aplikaci, jež tuto metodu využívá a zpracovává snímky z fotoaparátu zařízení za účelem rozpoznání registrační značky. Teoretická část nastiňuje charakteristické vlastnosti registračních značek a popisuje některé techniky z oblasti počítačového vidění, jež by mohly být použity při implementaci takové metody. S využitím frameworku Flutter a knihovny Firebase ML Kit je multiplatformní mobilní aplikace, jež tuto metodu využívá, vytvořena, a její vývoj je dále popsán v praktické části této práce. V poslední části této práce jsou popsány výsledky testování této metody v různých světelných podmínkách.

Obsah

1 Úvod

V dnešní době, kdy lze ve společnosti pozorovat rozšíření výkonných mobilních zařízení disponujících fotoaparátem, nachází metody z oblasti počítačového vidění stále širší uplatnění v běžném lidském životě. Jedním z praktických využití technik této oblasti je usnadnění digitalizace záznamů, která je i v dnešní době mnohdy prováděna ručně. Typickým příkladem automatické digitalizace záznamů je rozpoznávání registračních značek motorových vozidel a ačkoliv jsou již v tomto století metody k automatickému rozpoznávání značek používány například při provozu placených parkovišť nebo při monitorování dodržování silničních pravidel na veřejných komunikacích, jejich využití na mobilních zařízeních osobami v terénu již není tak časté.

Tato práce se zabývá problematikou implementace metody k rozpoznávání registračních značek motorových vozidel na mobilních zařízeních využívajících operační systémy Android nebo iOS.

Cílem této práce je vytvořit rozšiřitelnou multiplatformní aplikaci, která dokáže pomocí fotoaparátu v reálném čase rozpoznávat registrační značky ve snímané scéně a ukládat je do lokální databáze. Předpokládá se, že tato obecná aplikace bude v budoucnu sloužit jako platforma pro tvorbu aplikací, jež budou stavět na výše zmíněné funkcionalitě. Taková platforma by mohla například nalézt uplatnění v oblasti evidence vozidel v autoservisech, monitorování pohybu vozidel v dočasně zřízených kontrolních stanovištích nebo pořizování důkazních materiálů při pojistných událostech.

K implementaci této metody je třeba se seznámit s technikami z oblasti počítačového vidění, a proto se jimi tato práce v kapitolách 3 a 4 zabývá. V kapitole 2 je čtenář seznámen s konceptem a charakteristickými vlastnostmi registračních značek. V rámci této práce je také v kapitole 5 objasněna problematika vývoje mobilních aplikací. V následujících kapitolách 6 a 7 je popsáno jakých postupů bylo při tvorbě této mobilní aplikace využito a nakonec je v kapitole 8 diskutováno nad výsledky testování implementované metody k rozpoznávání registračních značek.

2 Registrační značky vozidel

Již brzy poté, co se na silnicích začala objevovat motorová vozidla, vznikla potřeba je jednoznačně identifikovat. Nejprve však metody označování vozidel nebyly zcela konzistentní a identifikátor mohl být umístěn kdekoliv na vozidle. Až s dalším vývojem a rozšířením vozů do společnosti vznikly první pokusy standardizovat jejich označování tak, jak to známe v dnešní době. Avšak jelikož se vývoj vozidel lišil napříč jednotlivými státy, tak se lišil i vývoj registračních značek, které se i dodnes mohou mezi sebou významně lišit. Nejčastěji se liší například rozměrem značky, použitým typem písma, formátem identifikačního řetězce nebo barvou pozadí. Hlavním z důvodů odlišnosti formátů identifikačního řetězce napříč státy je maximální počet vozidel, kterým lze přiřadit unikátní registrační značku. Proto si například státy s nižším počtem obyvatel mohou dovolit zvolit jednodušší formát identifikátoru, jelikož celkový počet registrovaných vozidel bude relativně malý.

2.1 Registrační značky v České republice

V dnešní době je vzhled registračních značek vozidel v České republice, pro které se používá spíše zastaralé označení státní poznávací značka, definován ve vyhlášce Ministerstva dopravy č. 343/2014 Sb, která dále popisuje celý proces registrace vozidel. Dle specifikace je určena pro registrační značky silničních motorových vozidel a přípojných vozidel standardně jednořádková značka s bílým pozadím a černým textem o rozměrech 520 x 110 mm. Taková značka musí být v levé části doplněna modrým pruhem se znakem Evropské unie a zkratkou názvu státu. Pro některá vozidla, typicky nákladní vozidla nebo autobusy, se používá také značka o rozměrech 340 x 200 mm a dvou řádcích. Pro identifikaci motocyklů se používá dvouřádková značka o rozměrech 200 x 160.

Obrázek 2.1: Příklady registrační značky používané v České republice.

Text na značce musí obsahovat pět až sedm znaků, přičemž je nutné, aby text vždy obsahoval alespoň jedno písmeno a jednu číslici. První písmeno zleva označuje kraj, ve kterém bylo vozidlo registrováno. Používaný

formát textu, jež slouží jako identifikátor, umožňuje vyrobit celkem 20 900 118 400 unikátních registračních značek. Mezi třetím a čtvrtým znakem identifikátoru se nachází nálepka, jež na sobě nese informaci o poslední kontrole daného vozidla ve stanici technické kontroly. Do ledna 2015 se pod toutou nálepkou nacházela i nálepka potvrzující, že vozidlo splňuje emisní testy.

Od roku 2016 se vystavují takzvané registrační značky na přání, na kterých může být vytištěn vlastní text. Vlastní text se musí skládat z osmi alfanumerických znaků obsahujících alespoň jednu číslici a dále nesmí obsahovat vulgární text či zakázaná písmena G, O, Q a W.

2.2 Registrační značky v Evropské unii

V Evropské unii se nyní vydávají značky, které mají v levé části svislý modrý pruh, ve kterém se nachází znak Evropské unie a kód státu, ve kterém bylo vozidlo registrováno, avšak starší historické značky, jež jsou v některých státech stále platné, tento modrý pruh neobsahují. V ostatních aspektech se jednotlivé značky napříč členskými státy liší. Není kladen požadavek na specifickou barvu pozadí, styl písma či na formát identifikátoru.

2.3 Registrační značky ve zbytku světa

Formát a podoba registračních značek se mezi státy po celém zbytku světa liší. Například velmi odlišné jsou od sebe i registrační značky mezi jednotlivými státy USA, které mohou být různě barevné a používat vlastní formát identifikátoru.

3 Počítačové vidění

Počítačové vidění je obor z oblasti informatiky, který se zabývá extrakcí informací ze snímku. Hlavní motivací tohoto oboru je snaha napodobit schopnosti člověka rozpoznávat a klasifikovat objekty v libovolné obrazové scéně. Úlohy z této oblasti, které mohou být pro člověka velice triviální a jednoduché, často bývají pro počítač velmi složité až dokonce neřešitelné. Jedná se například o rozpoznávání hran, lidských tváří či čtení tištěného textu. Jedním z hlavních důvodů, proč tyto úlohy mohou být pro počítač složité, je ten, že se jedná o takzvaný *inverzní problém*. To je takový problém, kdy na základě pozorovaných jevů chceme vyhledat příčiny, které tyto jevy způsobily.

V dnešní době nachází metody počítačového vidění široké uplatnění v praxi, kdy pomáhají řešit například následující problémy:

- **Bezpečnost v automobilovém průmyslu** – moderní vozidla dokážou v reálném čase analyzovat scénu a detekovat tak ostatní vozidla v silničním provozu, chodce a dopravní značení, čímž mohou zlepšovat bezpečnost v dopravě.
- **Ověřování kvality v továrnách** – nad pásem výrobní linky v továrně může být umístěna kamera připojená k počítači, který dokáže analyzovat snímky z kamery a automaticky tak kontrolovat kvalitu výrobků.
- **Skenování dokumentů** – automatická digitalizace dokumentů, jakými jsou například knihy, doklady či smlouvy.
- **Biometrie** – možnost autentizace pomocí otisku prstu, snímku duhovek či uplatnění v oblasti forenzní vědy.

S nárůstem výkonu dnešních počítačů můžeme sledovat i nárůst uplatnění počítačového vidění, kdy úlohy, které vyžadovaly vysoký výpočetní výkon a nebyly tak dříve řešitelné v praktickém světě, jsou dnes řešitelné dokonce i v reálném čase a každý den se s nimi setkáváme. Vliv na tento nárůst mělo i rozšíření fotoaparátů mezi širokou veřejnost, a to například prostřednictvím mobilních telefonů, které jsou v dnešní době schopné označit významné objekty ve snímané scéně, rozpoznávat jednotlivé osoby na snímcích nebo vytvořit 3D model objektu z několika snímků tohoto objektu.

3.1 Segmentace snímku

Úkolem segmentace snímku je automaticky rozdělit snímek na oblasti, v rámci kterých má snímek společné vlastnosti, jež jsou pro nás nějakým způsobem zajímavé. Typicky se tak segmentace používá pro lokalizaci objektů a jejich hranic ve snímku. Kromě počítačového vidění tato úloha v praxi také často nachází uplatnění v oblastech komprese obrazových dat nebo zpracování lékařských obrazových dat.

3.1.1 Prahování

Prahování je jedna z nejjednodušších technik segmentace snímku, která pracuje s jeho histogramem a intenzitou jasu každého pixelu. Hlavní myšlenkou této techniky je nalezení prahu – hodnoty skaláru, která rozděluje body snímku podle hodnoty úrovně jejich jasu na body popředí a pozadí. Kvalita segmentace na základě prahování je tedy přímo ovlivněna výběrem tohoto prahu, přičemž některé techniky používají pro prahování i více než pouze jeden práh. Pokud prahování pracuje pouze s jedním prahem, tak ve skutečnosti provádí binarizaci snímku: Tedy dle stanoveného prahu barvu jednotlivých bodů snímku nahrazuje buď zcela černou nebo bílou barvou.

(a) Příliš nízký práh (b) Příliš vysoký práh

Obrázek 3.1: Ukázka nesprávně zvoleného prahu při prahování snímku.

Klíčovou problematikou, kterou se technika prahování zabývá, je v rámci automatizace požadavek, aby byl pro každý snímek práh nalezen automaticky a poskytl nejlepší segmentaci. Metody pro automatické nalezení prahu se dělí na prostorové, lokální, shlukující, založené na entropii či využívající histogram snímku.

Jestliže pozadí ve vstupním snímku není uniformní (různé světelné podmínky podél snímku), pak metody k automatickému určení prahu většinou

neposkytují optimální výsledky. Řešením může být takzvané adaptivní prahování, kdy se vstupní snímek nejprve rozdělí do několika segmentů a zvolená metoda se tak aplikuje vždy pouze na danou oblast, čímž se výsledky prahování často zlepší. Obecně lze tedy říct, že metody pro automatické nalezení prahu dosahují nejlepších výsledků, pokud jsou použity na snímky zachycující jednoduché scény, ve kterých se objevují právě dva objekty s odlišnou odrazivostí či pohltivostí svého povrchu, kdy jeden představuje pozadí a druhý představuje popředí.

Vyvážený histogram

Jedná se o velmi jednoduchou metodu automatického nalezení prahu z histogramu snímku, jež navzdory své jednoduchosti dosahuje překvapivě dobrých výsledků při binárním prahování. Tato metoda hledá takový práh, který představuje lokální minimum histogramu mezi dvěma lokálními maximy histogramu [3]. Tento práh je vyhledáván tak, že je histogram dle svého středu rozdělen na dvě části, a ta část, ve které je součet jednotlivých četností vyšší, je zmenšena. Zmenšení probíhá tak, že z části se odebere ta četnost, která je vzdálenější od četnosti, která tyto části odděluje. Poté se vypočítá nový střed mezi těmito dvěma částmi. Celý proces se opakuje, dokud se obě vzdálenější četnosti od středu obou částí nesetkají na nějaké četnosti histogramu. Hodnota této četnosti je poté označena jako optimální nalezený práh.

(a) Vstupní snímek (b) Výstupní snímek

Obrázek 3.2: Ukázka prahování pomocí metody vyváženého histogramu.

Otsuova metoda

Otsuova metoda rozděluje histogram snímku do dvou tříd: První třídou jsou četnosti v histogramu, jež představují pozadí objektů ve snímku, a druhou třídou jsou četnosti, které představují popředí objektů ve snímku. Hlavním principem této metody je nalezení takového prahu, aby byl vážený vnitřní rozptyl tříd, které tento práh odděluje, co nejmenší. Následující vzorec tento rozptyl tříd popisuje:

$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t) \ ,$$

kde t je zkoumaný práh, $\omega_{0,1}(t)$ jsou pravděpodobnosti jednotlivých tříd a $\sigma^2_{0,1}(t)$ jsou rozptyly tříd. Algoritmus pro nalezení prahu funguje tak, že postupně vypočítává rozptyl pro všechny hodnoty, jichž může práh nabývat (typicky hodnoty od 0 do 255). Po vypočtení všech rozptylů je vybrán ten práh, pro který byl vypočten nejmenší rozptyl.

V praxi se však často používá jiný, výpočetně méně náročný postup, který je založen na tom, že celkový rozptyl je roven součtu vnitřního rozptylu tříd a rozptylu mezi třídami. Celý proces je tedy velmi podobný tomu, který byl v předchozím odstavci vysvětlen, avšak s tím rozdílem, že se snažíme maximalizovat mezitřídní rozptyl. Následujícím vzorcem je tento rozptyl vyjádřen:

$$\sigma_b^2(t) = \sigma^2 - \sigma_w^2(t) = \omega_0(t)\omega_1(t)\left[\mu_0(t) - \mu_1(t)\right]^2$$

kde σ^2 je rozptyl celého histogramu a $\mu_{0,1}(t)$ jsou aritmetické průměry jednotlivých tříd.

(a) Vstupní snímek (b) Výstupní snímek

Obrázek 3.3: Ukázka prahování pomocí Otsuovy metody.

3.2 Detekce hran

Detekce hran je v oblasti počítačového vidění velmi často používaný postup, který se v nějaké podobě se vyskytuje skoro ve všech úlohách tohoto oboru. Avšak již pouhá definice hrany může být problematická, jelikož hrany ve snímku mohou vznikat či zanikat v závislosti na úhlu pohledu. Někdy bývá jako hrana označována oblast ve snímku, ve které dochází k významné změně jasu. Z hlediska optiky je hrana považována za hlavního nositele informací o hranicích objektů.

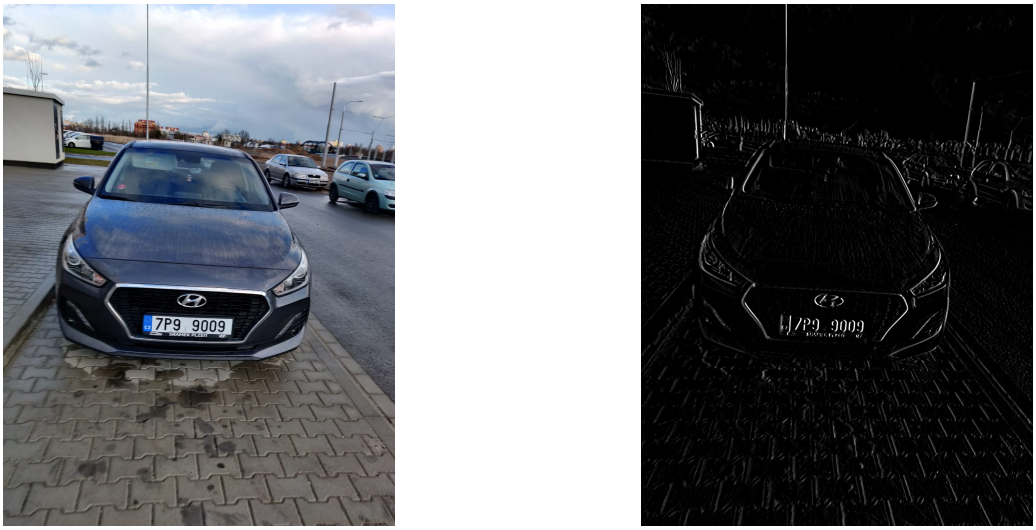
Před detekcí hran se ve většině případech provádí drobné rozostření snímku za účelem redukce šumu, který by mohl mít negativní vliv na výsledky detekčních metod.

3.2.1 Sobelův operátor

Sobelův operátor je konvoluční operátor, který ve snímku zvýrazní hrany. Před aplikací tohoto operátoru je třeba vstupní snímek převést do barevného prostoru, který má pouze jednu barevnou složku. Proces zvýraznění hran probíhá tak, že se zvlášť zvýrazňují horizontální a vertikální hrany pomocí následujících konvolučních operátorů:

$$G_h = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \qquad G_v = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}. \tag{3.1}$$

Jednotlivé složky, ze kterých se skládá vektor intenzity hrany, představují aproximaci derivace horizontálních a vertikálních změn ve snímku.

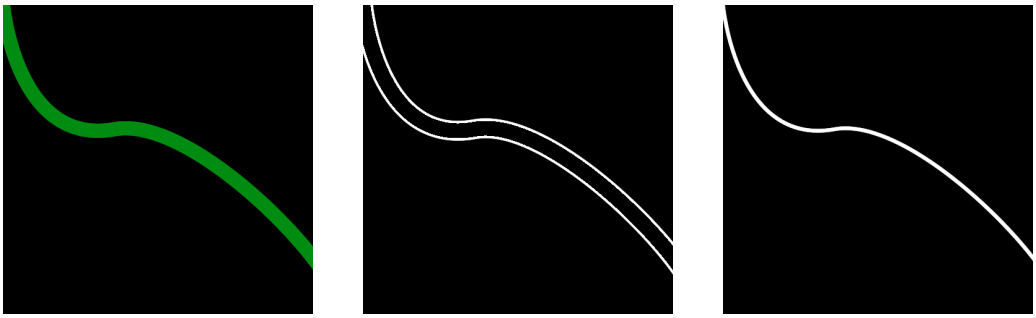


(a) Vstupní snímek (b) Výstupní snímek

Obrázek 3.4: Ukázka aplikace Sobelova operátoru.

3.2.2 Cannyho hranový detektor

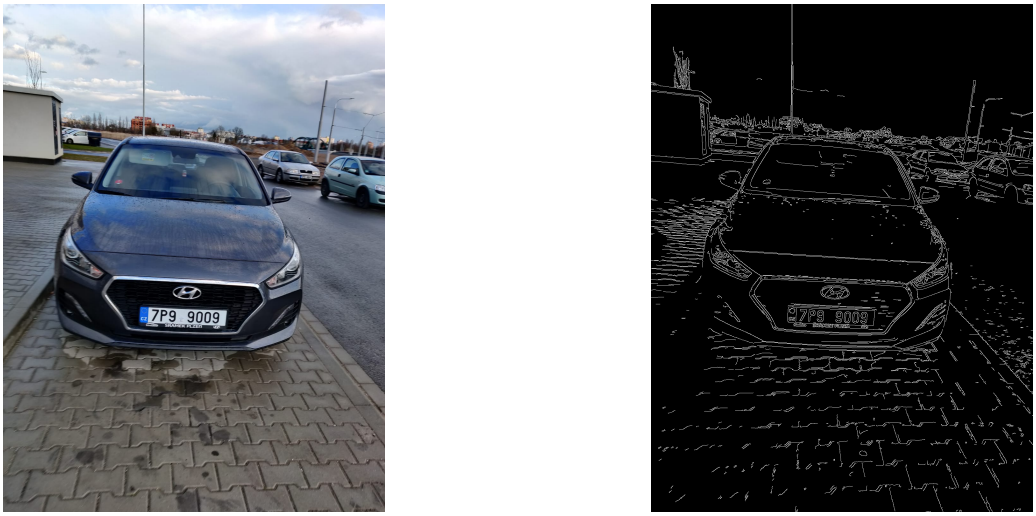
Tato metoda předpokládá, že vstupní snímek je černobílý a pracuje s gradientem změny intenzity snímku, který je ve většině implementací získáván pomocí Sobelova operátoru [9]. Pokud se v původním snímku vyskytovala široká hrana, tak se ve snímku, ve kterém jsou zobrazeny jeho derivace změn v jasové intenzitě, objeví zvýrazněné přechody mezi hranou a objekty, jež tato hrana odděluje. Z těchto přechodů lze pomocí hledání lokálního maxima ve směru hrany získat snímkové body, jež skutečně reprezentují nějakou hranu. Tato technika se nazývá ztenčení hran.



(a) Vstupní snímek (b) Aplikace Sobelova operátoru (c) Obraz po ztenčení hran

Obrázek 3.5: Ukázka ztenčení hran.

Avšak nyní, po zjištění přesných pozic hran, ve snímku stále zůstávají i relativně jemné hrany, které nemusí mít při zpracování snímku velký význam. V následujícím kroce, kdy je na hrany aplikováno prahování s hysterezí, tyto nevýznamné hrany odstraníme. Pro prahování se zvolí dva prahy – minimální a maximální. Jestliže je magnituda vektoru, který obsahuje intenzity jasové změny daného bodu ve snímku, vyšší než maximální práh, tak je bod automaticky považován za bod hrany. Pokud je hodnota menší než maximální práh a větší než minimální práh, tak je uznán jako bod hrany pouze tehdy, jestliže je některý z jeho sousedů považován za bod hrany. Body, ve kterých je magnituda vektoru obsahujícího intenzity jasových změn menší než je minimální práh, jsou klasifikovány jako body, na kterých hrana neleží.



(a) Vstupní snímek (b) Výstupní snímek

Obrázek 3.6: Ukázka aplikace Cannyho detektoru hran.

4 Optické rozpoznávání znaků

Optické rozpoznávání znaků, často také označované jako **OCR** (Optical Character Recognition), je technika umožňující převést psaný či tištěný text do digitální podoby sekvence alfanumerických znaků. Velký význam má pro digitalizaci knih, rukopisů či jiných objektů, na kterých se vyskytuje text. Tato technika je silně spjatá s oblastmi zpracování přirozeného jazyka, počítačového vidění a umělé inteligence.

4.1 Historie optického rozpoznávání znaků

První využití techniky optického rozpoznávání znaků lze pozorovat ve čtecích zařízeních, které pomáhaly zrakově postiženým pacientům číst psaný text. V roce 1914 Emanuel Goldberg vynalezl zařízení, které dokázalo číst jednotlivé znaky textu a převádět je do telegrafického kódu. Ve stejný čas byl vynalezen i první optofon, zařízení, jež je primárně určeno pro zrakově postižené a převádí psaný text do akordů, které uživateli následně přehrává. Na veřejné ukázce v roce 1918, kde byl optofon představen, dosahoval optofon rychlosti převedení jedno slovo za minutu. Další modely optofonu dosahovaly rychlosti až 60 slov za minutu.

V následujících letech byla této oblasti věnována pozornost za účelem automatizace zpracování textu v oblasti průmyslu a obchodu, kde se rozpoznávání znaků například používalo k čtení údajů z kreditních karet. Pro zjednodušení a zlepšení přesnosti optického rozpoznávání znaků byly vytvářeny speciální druhy písma. Některá písma, například ta použita na kreditních kartách, se používají v této roli dodnes.

S postupujícím vývojem počítačů postupoval i vývoj optického rozpoznávání znaků a již v průběhu druhé poloviny 20. století lze sledovat častější využití této techniky. Již v této době bylo možné se setkat s přenosnými zařízeními pro čtení tištěného textu, jež se využívalo pro čtení adres z poštovních zásilek, cenovek ze zboží a údajů z cestovních pasů.

Ke konci dvacátého století se na trhu objevují první komerční programy dostupné veřejnosti, které nachází širší uplatnění v praxi a umožňují i čtení textu z digitálního snímku.

Začátkem 21. století lze v této oblasti sledovat snahu nabízet technologii optického rozpoznávání znaků ještě širší veřejnosti. Objevují se první nástroje, které tuto funkcionalitu poskytují zdarma a online. OCR se již hojně

využívá v praxi pro digitalizaci dokumentů, knih a také například rozpoznávání registračních značek vozidel.

4.2 Postupy při rozpoznávání

4.2.1 Předzpracování snímku

Důležitou roli při rozpoznávání znaků ze snímku hraje jeho důkladné předzpracování. Čím lépe bude vstupní snímek předzpracován, tím lépe a přesněji v něm půjde rozpoznávat text.

Rotace snímku

Při rozpoznávnání je důležité, aby byl skenovaný dokument správně zarovnán. Řádky skenovaného textu by měly být rovnoběžné s horizontálními hranami snímku. Často tedy bývá potřeba vstupní snímek nějak transformovat, konkrétně provést rotaci a oříznutí.

Redukce šumu

Redukce šumu je proces odstraňování šumu ze signálu. Často se používá při zpracování zvuku či obrazových dat. Před rozpoznáváním textu je třeba tento šum minimalizovat. Většinou se to provádí jemným rozostřením snímku pomocí gausovského rozostření.

Binarizace

V rámci předzpracování je nutné provést binarizaci snímku, což je proces, při kterém se snímek převede do spektra pouze dvou barev (nejčastěji černé a bílé). Binarizace se provádí za účelem oddělení textu od pozadí a výrazně ovlivňuje přesnost rozpoznávacích algoritmů.

4.2.2 Rozpoznávání vzorů

K rozpoznávání jednotlivých znaků ve snímku se používá metod z oblasti rozpoznávání vzorů, které se zabývá automatickým vyhledáváním pravidelností v nějakých datech. Tyto metody využívají algoritmy schopné například klasifikace těchto pravidelností do tříd [4]. Tato oblast staví na základech teorie pravděpodobnosti a lineární algebry, které jsou pro pochopení této problematiky klíčové.

Při řešení problémů z této oblasti lze použít metody sestavené přesně na míru konkrétnímu problému a konkrétním datům, které by sice mohly dosahovat dobrých výsledků, avšak v praxi se takové metody příliš nepoužívají kvůli tomu, že zpracovávaná data se v reálném světě neustále mění či se v nich objevují různé artefakty. Proto se tedy v praxi velmi často k řešení těchto problémů používá metod strojového učení, ve kterých vystupují takzvaná *trénovací data* používaná ve fázi učení metody. Trénovací data, pro která se spíše používá název *trénovací množina* (dále TM), obsahují data, jež jsou podobná těm, která se objevují na vstupu při řešení problému. Pokud jsou trénovací data označkována a TM obsahuje informaci o očekávaném výsledku, pak se učící fáze metody nazývá *učení s učitelem*. Při absenci očekávaného výsledku se učící fáze nazývá *učení bez učitele*. Během fáze učení si metoda dle trénovacích dat vytvoří model, podle kterého následně klasifikuje vstupní data.

Metody z oblasti rozpoznávání vzorů při rozpoznávání znaků mohou také ve snímku vyhledávat charakteristické rysy jednotlivých znaků. Před vyhledáváním je však třeba pečlivě tyto rysy definovat, aby jejich vyhledávání bylo co nejpřesnější.

4.3 Rozpoznávání ze snímku s očekávaným obsahem

Velmi často se stává, že při optickém rozpoznávání znaků ze snímku předem nevíme, jaký text může daný snímek obsahovat. Avšak v některých situacích je nám kontext, v rámci kterého byl snímek pořízen, předem znám. V takové situaci můžeme následně předvídat, jaký text snímek obsahuje a vhodně tak upravit klasifikaci jednotlivých znaků nebo ověřit správnost výsledku metody. Dle kontextu je tedy možné vytvořit množinu znaků, ze kterých se má rozpoznaný text skládat a při nalezení znaků nacházejících se mimo tuto množinu výsledek zamítnout či nějak transformovat.

Tyto úpravy a validace rozpoznaného textu na základě očekávaného obsahu snímku mohou významně vylepšit přesnost vybrané metody rozpoznávání znaků s prakticky nulovým dopadem na časovou náročnost, která již tak může být poměrně vysoká. V případech, kdy je OCR používané v reálném čase, je tedy velmi vhodné brát zřetel na kontext pořízení snímku, jelikož časová náročnost a přesnost hraje klíčovou roli v použitelnosti dané metody.

4.3.1 Rozpoznávání ze snímku obsahující registrační značku vozidla

V případě rozpoznávání znaků ze snímku, kde víme, že by se měla objevovat registrační značka vozidla (dále jen RZ), můžeme ověřit správnost výsledku použité metody rozpoznávání znaků tak, že rozpoznaný text zkontrolujeme dle očekávaného formátu RZ. Například při rozpoznávání jednořádkových registračních značek vozidel v České republice víme, že značka nesmí obsahovat znaky G, O, Q a W (viz kapitola 2.1). Pokud tedy vybraná metoda rozpoznávání textu objeví znak `O`, tak zcela jistě můžeme říct, že tento znak nebyl v kontextu rozpoznávání RZ rozpoznán správně. Na základě okolností můžeme následně tento výsledek zcela zamítnout nebo jej nějakým způsobem vhodně upravit. V úvahu připadá špatně rozpoznaný znak nahradit jiným znakem, který mu je podobný a je u něj velká pravděpodobnost záměny. V případě rozpoznaného znaku `O`, jež se v klasické jednořádkové RZ nacházet nemůže, se jeví jako správné ho nahradit znakem `0`, kterému je velmi podobný a v registračních značkách se může objevovat.

Obrázek 4.1: Ukázka špatného výsledku OCR, jež lze opravit jednoduchou záměnou znaku `O` za `0`.

4.4 Knihovny pro optické rozpoznávání znaků

4.4.1 Firebase ML Kit

Firebase je platforma poskytující základní infrastrukturu pro tvorbu mobilních a webových aplikacích a je spravována společností Google [1]. Tato platforma je dostupná v rámci placených i bezplatných tarifů a poskytuje širokou škálu prostředků a služeb, které mohou být při vývoji aplikací potřeba. Mezi ně patří například škálovatelné cloudové úložiště, možnost vykonávat výpočetně náročné operace na serveru bez nutnosti tento server spravovat nebo testování vyvíjené aplikace na široké škále fyzických mobilních zařízení.

Jednou z dalších služeb je také možnost využít knihovnu ML Kit, která poskytuje základní i pokročilé funkce z oblasti strojového učení a počíta-

čového vidění. Tato knihovna obsahuje i modul pro optické rozpoznávání znaků, které je možné využít buď lokálně na zařízení nebo v cloudu, kde poskytuje za cenu delší odezvy přesnější výsledky. K rozpoznávání textu knihovna ML Kit kombinuje technologie Google Cloud Vision API, TensorFlow Lite a Android Neural Networks API.

K využití této knihovny v mobilní aplikaci je třeba se zaregistrovat na stránkách platformy Firebase a následně v aplikaci uvést vývojářský API klíč, který je dostupný po úspěšné registraci.

4.4.2 Tesseract

Jedná se o otevřený software naprogramovaný v jazyce C++, jenž slouží pro účely optického rozpoznávání textu a je dostupný na platformách Linux, Windows a macOS[10]. Tesseract byl původně vyvíjen společnost Hewlett-Packard Co. v jazyce C. Od roku 2005 je dostupný jako otevřený software a na vývoji se nejvíce podílí společnost Google.

Nejnovější verze 4.1.1 této knihovny používá neuronové sítě typu LSTM (Long-Short-Term-Memory) a uživateli poskytuje předpřipravené naučené modely s vysokou přesností ve více než 100 jazycích. Lze ji použít v mnoha programovacích jazycích včetně C#, Python, Java či Swift. Z důvodu podpory jazyků Java či Swift nachází tato knihovna časté využití při vývoji mobilních aplikacích.

4.4.3 Asprise OCR

Asprise OCR je komerční software pro optické rozpoznávání znaků, jež umožňuje výstup rozpoznávání do formátu XML, PDF nebo prostého textu. V rámci svého **SDK** (Software Development Kit) nabízí **API** (Application Programming Interface), které lze použít v programovacích jazycích Java, C# VB.NET nebo C++ bez nutnosti poskytovat naučený model.

5 Vývoj mobilních aplikací

Vývoj mobilních aplikací cílí na operační systémy mobilních zařízení. Mezi operační systémy mobilních zařízení se řadí operační systémy pro mobilní telefony, tablety nebo také chytré hodinky. Ve třetím kvartálu roku 2019 má mobilní operační systém Android největší podíl na trhu všech operačních systémů včetně těch určených pro stolní počítače a laptopy [7]. Druhým nejvíce zastoupeným mobilním operačním systémem je iOS, který je vyvíjen společností Apple Inc.

Ve většině případů mají mobilní aplikace složitější strukturu než aplikace desktopové, a to z toho důvodu, že desktopové aplikace mívají pouze jeden vstupní bod a běží jako jeden monolitický proces. Oproti tomu mobilní aplikace mají svůj životní cyklus mnohem složitější: Mají hned několik vstupních bodů a mohou být kdykoliv, i bez akce uživatele, přesunuty do pozadí. Příkladem může být příchozí hovor, který aplikaci pozastaví a zminimalizuje. Po skončení hovoru uživatel očekává plynulý návrat do aplikace s tím, že se bude nacházet v takovém stavu, v jakém byla před jejím opuštěním. Dalším příkladem může být mobilní aplikace, jež slouží jako emailový klient, u kterého se typicky po spuštění má zobrazit seznam příchozích zpráv, ale pokud uživatel vznesl v jiné aplikaci explicitní požadavek na vytvoření nové emailové zprávy, předpokládá se, že tato aplikace po spuštění zobrazí právě obrazovku pro vytvoření nové zprávy. Toto chování se mnohdy objevuje i v desktopových aplikací, avšak v mobilních aplikacích se s ním setkáváme výrazně častěji.

Vývojář mobilních aplikací musí dále počítat s tím, že mobilní zařízení typicky disponují omezenějšími hardwarovými prostředky než stolní počítače či notebooky, a tudíž je třeba věnovat jistou pozornost optimalizaci těchto aplikací. Nevhodně naprogramovaná aplikace by mohla mít například negativní vliv na výdrž baterie.

5.1 Historie mobilních operačních systémů

Před nástupem operačních systémů, které uživateli nabízejí velkou přizpůsobitelnost svého zařízení, se v mobilních zařízeních nacházely jednoduché operační systémy, které nebyly příliš modulární. Každý výrobce vyvíjel a používal ve svých zařízeních vlastní operační systém. Tyto systémy většinou nebývaly v průběhu života zařízení aktualizovány, takže uživatel měl po celý čas používání svého telefonu jen jednu a tu samou verzi operačního systému.

To znamenalo, že vydaný operační systém musel být stabilní a odladěný, jelikož možnost aktualizace měl jen malý zlomek uživatelů. Ve výsledku tedy byly tyto systémy velmi jednoduché a nabízely pouze základní funkčnost, která byla od nich očekávána.

Jedním z prvních rozšiřitelných operačních systémů byl **Symbian**, který umožňoval uživatelům do svého telefonu instalovat další aplikace. Objevoval se například v telefonech značky Nokia, Sony Ericsson nebo Motorola. V roce 2005 společnost Apple Inc. představila mobilní telefon **iPhone** s operačním systémem iOS, který odstartoval revoluci v oblasti vývoje mobilních telefonů, kdy se začal klást větší důraz na multimedialitu těchto zařízení. O rok později byl uskupením výrobců mobilních telefonů **Open Handset Alliance** představen telefon **HTC Dream** s operačním systémem **Android** založeným na jádře **Linuxu**. V této době vyvíjela a nabízela vlastní mobilní operační systém i společnost Microsoft, ale na trhu mobilních zařízeních měl tento systém pouze minoritní zastoupení. V roce 2012 Nokia, hlavní výrobce telefonů využívající platformu Symbian, vydává své poslední zařízení s tímto operačním systémem, jelikož se rozhodla, že bude používat operační systém založený na platformě **Windows Phone** od společnosti Microsoft. Ani tato platforma však nebyla schopna konkurovat úspěšnosti platforem Android a iOS. V prosinci roku 2019 skončila podpora operačního systému **Windows 10 Mobile**. Jiné operační systémy, které jsou dnes stále v aktivním vývoji a stojí za zmínku, jsou například **KaiOS**, **Sailfish OS** a **PureOS**.

5.2 Vývoj pro Android

Android je mobilní operační systém, který je založen na upravené verzi Linuxového jádra a je primárně určen pro zařízení s dotykovou obrazovkou. Je používán na široké škále mobilních zařízení, od telefonů, přes přenosnou elektroniku až po tablety či pokladní systémy. Narozdíl od konkurenčního operačního systému iOS není možné otestovat vyvíjené aplikace na všech zařízeních, na kterých mohou být teoreticky spuštěny. Fakt, že tento operační systém je používán na mnoha typech zařízení, ovlivňuje i kvalitu Android SDK, které musí brát v potaz širokou škálu zařízení, na kterých bude používáno. Příkladem roztříštěnosti Android SDK může být modul pro práci s fotoaparátem zařízení, který za historii vývoje Androidu prošel třemi hlavními verzemi, které se snažily sjednotit a zlepšit přístup k funkcím fotoaparátu. Poslední verze, která se nazývá **Camera X**, slibuje konzistentní chování až na 90 % všech zařízeních používající Android OS [5].

Aplikace pro platformu Android jsou primárně psané v programovacím

jazyce **Java** nebo **Kotlin**. Oficiální dokumentace Androidu doporučuje používat programovací jazyk Kotlin, který lze přeložit do bytekódu virtuálního stroje **JVM** (Java Virtual Machine). V neposlední řadě je možné vytvářet nativní knihovny přímo v programovacím jazyce C++.

5.2.1 Aktivity a fragmenty

Každá aplikace operačního systému Android má alespoň jednu, avšak typicky více, tzv. **aktivit**. Při každém spuštění aplikace slouží aktivita, jež má v obecném popisu dle dokumentace Androidu představovat jednu obrazovku, jako vstupní bod aplikace. V hlavním konfiguračním souboru aplikace, tzv. **manifestu** aplikací Android OS, se nachází deklarace všech aktivit, které například určují, na které požadavky aktivity reagují nebo jaká práva vyžadují ke svému spuštění.
V průběhu svého životního cyklu aktivita prochází několika stavy a následující metody, jež některé musí programátor ve své aplikaci implementovat, jsou volány při přechodech mezi stavy:

- `onCreate()` – volána při vytvoření aktivity a má za úkol zinicializovat aplikaci, vytvořit prvky uživatelského rozhraní či provést regeneraci z předešlého stavu aplikace.

- `onStart()` – zviditelňuje aktivitu uživateli a je volána po vytvoření aktivity. Má za úkol provést poslední změny v uživatelském rozhraní, které je v tuto chvíli již inicializované. Obsluha této metody by typicky neměla trvat příliš dlouho.

- `onResume()` – představuje přechod do stavu aktivity, ve kterém již uživatel provádí interakci s uživatelským rozhraním. V tomto stavu aktivita přetrvává do té doby, dokud není volána metoda `onPause()`, jež může být vyvolána například příchozím hovorem nebo uzamčením zařízení.

- `onPause()` – volání této metody reprezentuje první signál toho, že uživatel opouští aktivitu a aktivita se přesouvá do pozadí. V rámci tohoto volání se nejčastěji uvolňují zabrané prostředky, ale očekává se, že její provádění není příliš časově náročné. Pokud byla aktivita zcela přesunuta do pozadí, je volána metoda `onStop()`.

- `onStop()` – v momentu, kdy aktivita již není pro uživatele viditelná, je volána tato metoda. V rámci této metody se mohou provádět časově náročnější operace či další uvolňování prostředků.

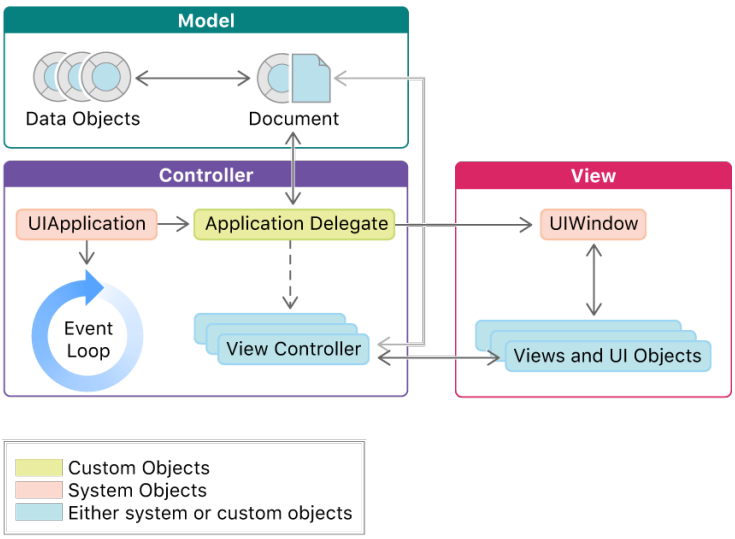
- `onDestroy()` – tato metoda se volá těsně před tím, než je aktivita zcela zrušena. Zrušení může být vyvoláno buď tím, že uživatel ukončil aplikaci nebo tím, že byla změněna orientace obrazovky, kdy hned dalším voláním bude volání metody `onCreate()`.

Důležitým prvkem pro tvorbu aplikací operačního systému Android jsou také fragmenty, které si lze představit jako modulární část aktivity, která má svůj vlastní životní cyklus. Každý fragment musí být umístěn v aktivitě a aktivita může obsahovat více fragmentů. Životní cyklus fragmentů většinou kopíruje cyklus své hostitelské aktivity. V dnešní době se velmi často používá taková architektura aplikace, která obsahuje jednu aktivitu a jednotlivé obrazovky aplikace jsou implementovány pomocí fragmentů.

5.3 Vývoj pro iOS

Operační systém iOS se v současné době používá na mobilních telefonech iPhone a jeho aktualizace vycházejí na devět nejnovějších modelů tohoto zařízení, které jsou podporovány. Aplikace na iOS lze vyvíjet buď v jazyce **Objective-C** nebo **Swift** pomocí vývojového prostředí **XCode**, které je dostupné pouze na platformě **macOS**.

Pro tvorbu uživatelských rozhraní se používá framework **UIKit**, který například poskytuje infrastrukturu pro podporu dotykových obrazovek, animací a vykreslování na plátno. Struktura aplikace UIKit vychází z návrhového vzoru MVC (*Model-View-Controller*), který odděluje objekty podle jejich funkce v aplikaci.

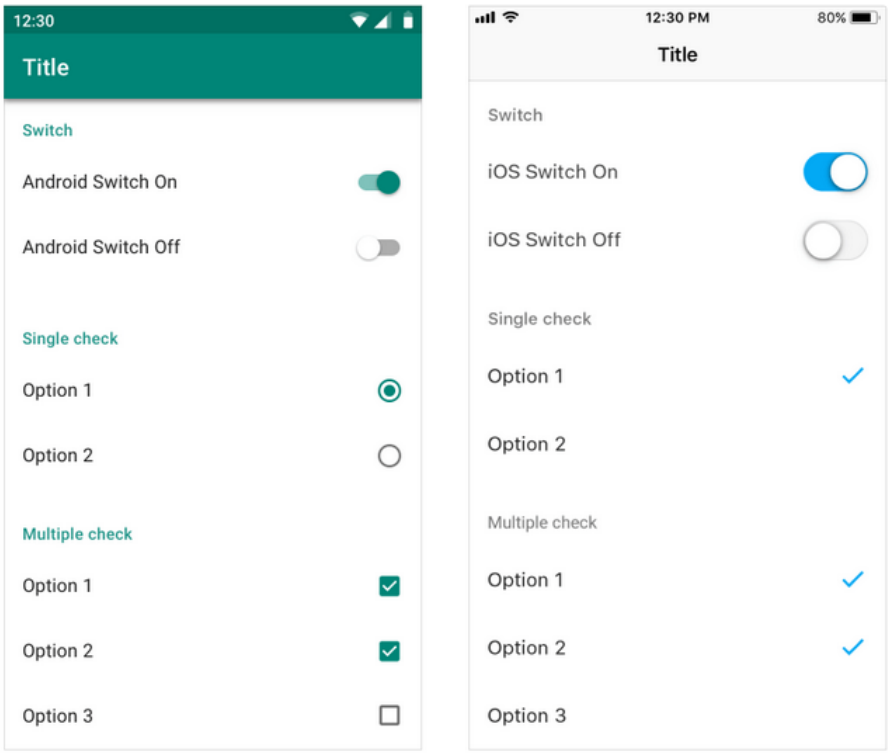


Obrázek 5.1: Zjednodušený diagram architektury UIKit aplikace.

5.4 Multiplatformní vývoj

V dnešní době je velmi žádoucí, aby vyvíjená mobilní aplikace byla dostupná pro obě hlavní platformy, tj. Android a iOS. Takové mobilní aplikace se neustále vyvíjí a bývají často aktualizovány. S tím přichází povinnost aktualizovat a udržovat obě verze aplikace. Tyto dvě platformy jsou však od sebe natolik odlišné, že vývojáři nativních mobilních aplikací se často specializují pouze na jednu z uvedených platforem. To by znamenalo, že na údržbu jednoho produktu by bylo zapotřebí dvou týmů, které by spolu musely úzce spolupracovat, aby zachovaly stejnou funkčnost napříč platformami. Tato možnost, kdy sdílení a znovupoužití kódu je minimální, se často jeví jako velmi neefektivní a časově náročný proces. Z tohoto důvodu vzniklo již několik mobilních frameworků, které se snaží tento problém vyřešit.

Mezi úskalí, se kterými se musí frameworky pro vývoj multiplatformních mobilních aplikací mnohdy potýkat, patří návrh designu těchto aplikací, jelikož vzhledově se obě platformy významně liší. Ačkoliv jsou si některými uživatelskými prvky tyto platformy podobné, tak umístění a rozložení komponent uživatelského rozhraní bývá ve většině případů vždy jiné.



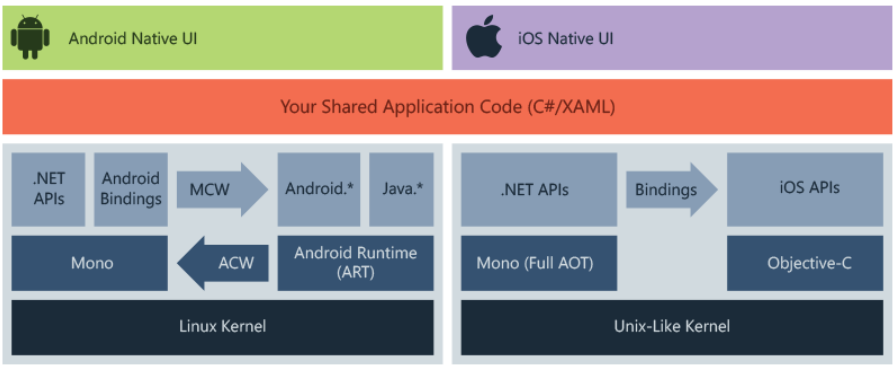
Obrázek 5.2: Porovnání prvků uživatelského rozhraní na platformách Android a iOS.

Avšak dnes se mnohdy setkáváme s tím, že aplikace implementují svůj vlastní design a nedodržují doporučený vzhled dané platformy. Často tedy nevadí, naopak to může být i žádoucí, když aplikace vypadá stejně na obou platformách.

Při rozhodování, zdali budeme používat framework pro tvorbu multiplatformních aplikací nebo budeme vyvíjet dvě verze aplikace, je třeba se zamyslet, jakou bude mít naše aplikace klíčovou funkcionalitu. Pokud plánujeme primárně využívat specifickou funkcionalitu zařízení, jako je například fotoaparát, 3D grafika nebo různé senzory, tak je rozumnější volbou zvolit vývoj dvou nativních aplikací. Zpřístupnění funkcionalit, které jsou specifické danému zařízení (např. použití některých senzorů zařízení), bývá právě tím největším nedostatkem knihoven pro vývoj multiplatformních mobilních aplikací, jelikož je nemožné zcela zobecnit přístup k API jednotlivých SDK. Pokud naopak naše aplikace provádí například pouze čtení a zápis do souborů nebo zobrazuje data na obrazovku, tak je velmi vhodné vyvíjet pouze jednu multiplatformní aplikaci.

5.4.1 Framework Xamarin

Xamarin je open-source framework primárně vyvíjený společností Microsoft pro vytváření moderních a výkonných aplikací pro Android, iOS a Windows postavených na .NET platformě [11]. Xamarin představuje abstrahovanou vrstvu zajišťující komunikaci kódu, který je mezi platformami sdílen, s cílovou platformou, pro kterou v daný moment aplikaci sestavujeme. V průměru je možné dosáhnout sdílení až 90% kódu napříč platformami. To umožňuje vývojářům psát veškerou logiku aplikace v jednom moderním jazyce, konkrétně v jazyce C#, a pro jednotlivé platformy pak vytvořit vlastní uživatelské rozhraní s nativním vzhledem a výkonem. Xamarin staví na knihovně *Mono* představující open-source verzi knihovny .NET.



Obrázek 5.3: Diagram architektury Xamarin.

Pokud by chtěl vývojář sdílet i kód uživatelského rozhraní, má možnost použít knihovnu *Xamarin.Forms*, která tuto funkcionalitu poskytuje a nabízí výběr z omezené sady prvků uživatelského rozhraní.

5.4.2 Framework React Native

V roce 2015 představuje společnost Facebook svůj open-source framework React Native, který umožňuje tvořit multiplatformní mobilní aplikace v jazyce JavaScript. Lze tak sdílet kód uživatelského rozhraní i logiky aplikace mezi platformami. Je založený na knihovně React, která se používá při vývoji webových aplikací pro tvorbu uživatelských rozhraní.

React Native používá pro vykreslení obecné definice uživatelského rozhraní v jazyce JavaScript nativní komponenty dané platformy, čímž je zajištěno dodržování základních designových směrnic. Ve specifických případech je nutné oddělit definici uživatelského rozhraní mezi platformami z důvodů optimalizace frekvence vykreslování snímků obrazovky. Jedná se například o přepínání mezi obrazovkami aplikace nebo vykreslování objemného seznamu položek.

Zatímco na platformě iOS je pro interpretaci JavaScript kódu použitý virtuální stroj **JavaScriptCore**, který je mimo jiné využíván i ve výchozím prohlížeči Safari, tak na platformě Android je použit virtuální stroj **V8**. Použití různých interpretů může v některých situacích vést k nekonzistentnímu chování napříč platformami [8]. Komunikace mezi virtuálním strojem a nativním prostředím probíhá asynchronní výměnou zpráv ve formátu JSON, kdy nativní prostředí interpretuje tyto zprávy dle své platformy a prostředků.

Pomocí tohoto frameworku je také možné psát nativní moduly pro jednotlivé platformy, se kterými lze z aplikace React Native komunikovat. Vytváření nativních modulů je vhodné v případech, kdy chceme vykonávat nějaký výpočetně náročný kód, jehož vykonání by mohlo interpretací JavaScriptu výrazně ovlivnit plynulost aplikace.

5.4.3 Framework Flutter

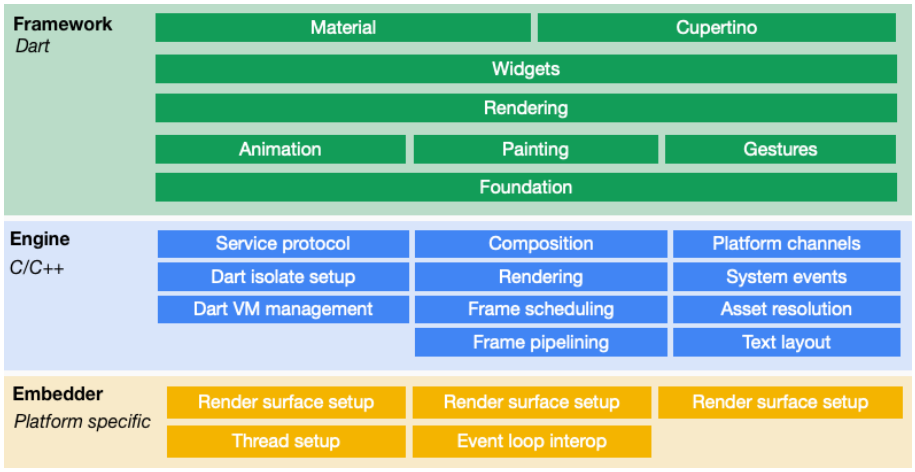
Flutter je open-source framework na jehož vývoji se podílí převážně Google. Cílem Flutteru je nabídnout vývojářům možnost vyvíjet výkonné aplikace, které působí nativně na všech nejrozšířenějších platformách (tzn. že tyto aplikace nelze jednoduše rozeznat od aplikací, jež jsou vyvíjené speciálně na danou platformu). V současné době je pomocí tohoto frameworku možné vytvářet kromě mobilních aplikací i desktopové nebo webové aplikace [6].

Flutter umožňuje sdílet veškerý kód mezi všemi platformami. Toho je docíleno tak, že knihovna nepoužívá žádné nativní komponenty uživatelského rozhraní dané platformy. Okno aplikace totiž pouze slouží jako plátno a všechny komponenty si Flutter vykresluje sám. K vykreslování používá open-source knihovnu Skia, která je napsaná v jazyce C++, čímž dosahuje plynulosti v zobrazování uživatelského rozhraní. Pro tvorbu aplikací pomocí Flutteru se používá programovací jazyk Dart.

Flutter respektuje rozdíly v chování uživatelských rozhraní mezi mobilními operačními systémy, čímž pomáhá aplikacím psaným v tomto frameworku působit tak, jako kdyby byly nativní a cílené právě pro danou platformu. Mezi hlavní rozdíly mezi platformami, jež tato knihovna implementuje na jednotlivých platformách zvlášť, patří například vertikální posun obrazovky, ikony nebo typografie.

Při nasazení na platformu Android je C a C++ kód enginu kompilován pomocí Android NDK (Native Development Kit), zatímco Dart kód se s využitím kompilace typu ahead-of-time kompiluje do nativních knihoven pro procesory ARM a x86. Tyto knihovny jsou poté použity ve vygenerované nativní aplikaci, která slouží jako hostitel, a z celého projektu je následně vytvořen APK balík.

Podobně probíhá i nasazení aplikace na platformu iOS, kdy je kód enginu překládán pomocí LLVM. Kompilace do nativních knihoven znamená, že narozdíl od ostatních frameworků pro tvorbu multiplatformních aplikací, jsou Flutter aplikace zcela nativní.



Obrázek 5.4: Diagram architektury Flutter.

Při vyvíjení aplikací pomocí Flutteru je použit virtuální stroj, který umožňuje upravovat zdrojový kód aplikace bez nutnosti ji restartovat. Tato

funkce má největší význam při tvorbě uživatelského rozhraní, kdy se jednotlivé změny v designu aplikace ihned projeví na zařízení.

Prvky uživatelského rozhraní, které Flutter nabízí, se snaží co nejvěrněji napodobit ty nativní. Prvky napodobující prvky platformy Android se nachází v balíku `Material` a prvky platformy iOS v balíku `Cupertino`. Tým vývojářů Flutteru bere ohledy na aktualizace mobilních operačních systémů a změny v uživatelských rozhraní včas implementuje do svého frameworku.

Flutter dále umožňuje psát nativní kód specifický pro danou platformu pomocí konstrukce zvané `platform channel`, která funguje na principu asynchronního předávání zpráv. Část aplikace Flutter pošle hostitelské nativní aplikaci zprávu, která asynchronně na tuto zprávu odpoví. To umožňuje přístup k nativnímu API dané platformy.

Na konci roku 2019, v každoročním shrnutí služby GitHub, je programovací jazyk Dart, který se v dnešní době používá nejvíce právě v aplikacích Flutter, označen jako jazyk s nejrychleji rostoucím počtem vývojářů, jež ho používají pro vývoj aplikací [2].

5.4.4 Frameworky využívající WebView

S nárůstem výkonu hardwaru dnešních mobilních zařízení lze pozorovat i nárůst multiplatformních mobilních aplikací, které využívají komponentu **WebView**. Tato komponenta je na každé platformě implementovaná zvlášť a představuje okno, jež funguje jako plnohodnotný internetový prohlížeč podporující HTML5, CSS3 a JavaScript. Aplikace postavené na této komponentě fungují tak, že v nativní aplikaci je přes celou obrazovku zobrazena právě tato komponenta, jejíž obsah, tvořený webovými stránkami, je načítán z lokálního úložiště či je dokonce součástí aplikace. Při používání těchto aplikací si uživatel většinou ani nevšimne, že se nejedná o nativní aplikaci. Tyto frameworky pomocí komunitních pluginů také nabízí možnost přistupovat k nativnímu SDK a využít tak například senzory zařízení, fotoaparát nebo Bluetooth. Mezi takové frameworky patří například Ionic nebo Adobe Phonegap.

5.4.5 Výběr frameworku pro vývoj aplikace rozpoznávající registrační značky vozidel

Aplikace, která bude rozpoznávat registrační značky vozidel, musí být optimalizována a používat fotoaparát mobilního zařízení. Jako nejvhodnější řešení se tedy jeví vytvořit nativní verze aplikace pro platformy Android a iOS. Avšak vývoj dvou nativních verzí aplikace, z nichž by každá použí-

vala úplně jiné SDK, by byl nad rámec této práce. Takové řešení by bylo realizovatelné v rámci diplomové práce nebo například v nějaké firmě, kde pro obě platformy mají zvláštní tým zkušených vývojářů. Proto je tedy nutné zvolit nějakou knihovnu pro vývoj multiplatformních aplikací.

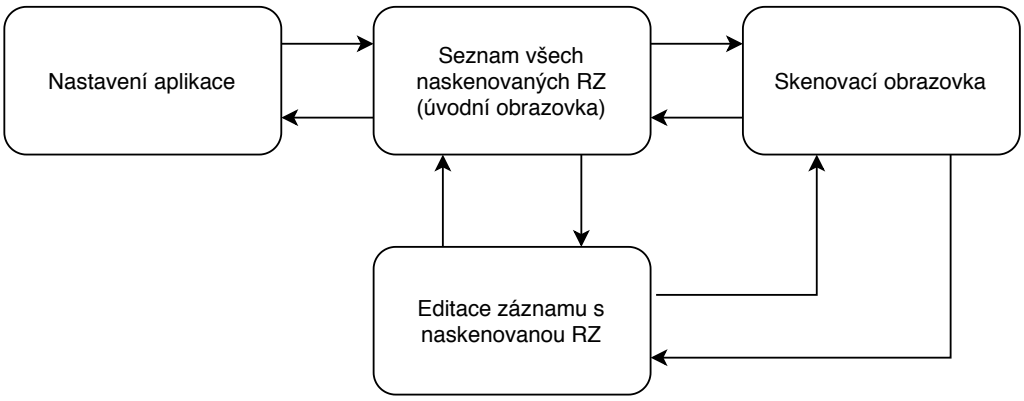
Aplikace vytvořené pomocí knihovny Xamarin, bez využití balíčku *Xamarin.Forms*, dosahují podobného výkonu jako nativní aplikace. Při použití této knihovny je však nutná alespoň základní znalost obou platforem, jelikož je napříč platformami sdílena pouze logika aplikace. Například obsluha fotoaparátu by musela být implementována pro každou platformu zvlášť.

Při vyvíjení multiplatformních aplikací s využitím frameworku React Native a dostupných komunitních pluginů je sice možné sdílet veškerý kód aplikace, ale kvůli architektuře této knihovny, jež zahrnuje výměnu zpráv mezi virtuálním strojem JavaScript a nativním prostředím, jsou tyto aplikace oproti těm nativním značně pomalejší, pokud často provádí výpočetně náročné operace, což by skenování registračních značek v reálném čase bylo. Proto volba tohoto frameworku, při tvorbě aplikace hojně využívající techniky z oblasti počítačového vidění, jež typicky bývají výpočetně náročné, není příliš vhodná.

Naopak framework Flutter nabízí kompromis, kdy s využitím komunitních pluginů není prakticky vůbec nutné mít znalost jednotlivých platforem a zároveň díky kompilaci do nativních knihoven ARM a x86 dosahuje srovnatelného výkonu jako již například zmíněný framework Xamarin. Avšak je třeba brát zřetel na to, že tato knihovna je oproti ostatním zmíněným knihovnám relativně nová a nemusí být ve všech případech zcela stabilní.

6 Uživatelské rozhraní mobilní aplikace k rozpoznávání registračních značek

Při vývoji této aplikace byl zvolen jednoduchý a přehledný vzhled, jenž kombinuje vizuální prvky platforem Android a iOS. Vzhled je na obou těchto platformách identický a poskytuje rychlou navigaci mezi jednotlivými funkcemi aplikace.

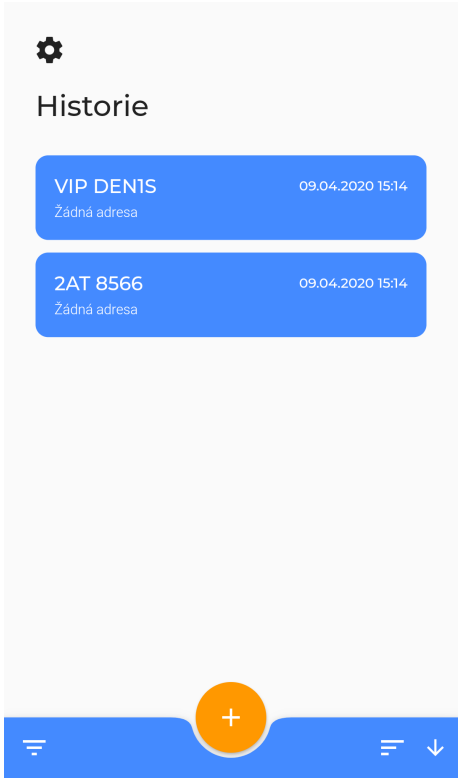


Obrázek 6.1: Diagram přechodů mezi obrazovkami aplikace.

6.1 Popis jednotlivých obrazovek

6.1.1 Úvodní obrazovka

Po spuštění aplikace se zobrazí seznam již naskenovaných registračních značek. Ke každé položce v seznamu je zobrazen identifikátor RZ, adresa, na které byla RZ naskenována a datum, kdy byla tato registrační značka naskenována. Kliknutím na položku seznamu se otevře obrazovka editace této položky. Uprostřed spodní lišty se nachází oranžové tlačítko, které slouží k spuštění skenování registračních značek. Ostatní tlačítka v této liště slouží k filtraci a seřazení položek seznamu. V levém horní rohu této obrazovky se dále nachází tlačítko pro otevření obrazovky s nastavením aplikace.



Obrázek 6.2: Úvodní obrazovka aplikace

6.1.2 Obrazovka skenování registračních značek

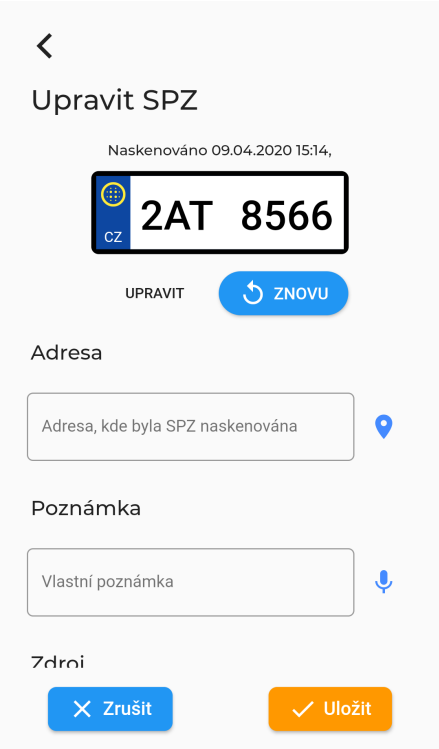
Tato obrazovka je velmi jednoduchá a slouží ke skenování registračních značek. Přes celou obrazovku je zobrazen výstup fotoaparátu, ve kterém je v reálném čase vyhledávána registrační značka. Pro ukončení skenování je ve spodní části obrazovky umístěno tlačítko s ikonou šipky.

Pokud displej používaného mobilního zařízení má jiný poměr stran, než je výstup jeho fotoaparátu, tak je tento výstup transformován tak, aby vyplnil celý displej zařízení, avšak je na něj aplikováno drobné rozostření a je překryt modrou průhlednou vrstvou. Přes tento výstup je znovu umístěn výstup fotoaparátu, jehož poměr stran je zachován a je zarovnán na střed. Tímto je tak docíleno toho, že na obrazovce, která může být různě velká, není nikdy prázdné místo.

6.1.3 Obrazovka editace naskenované registrační značky

Na této obrazovce má uživatel aplikace možnost upravit naskenovanou registrační značku. V horní části obrazovky je zobrazen identifikátor rozpoznané registrační značky, nad kterým se nachází datum, kdy byla RZ naskenována.

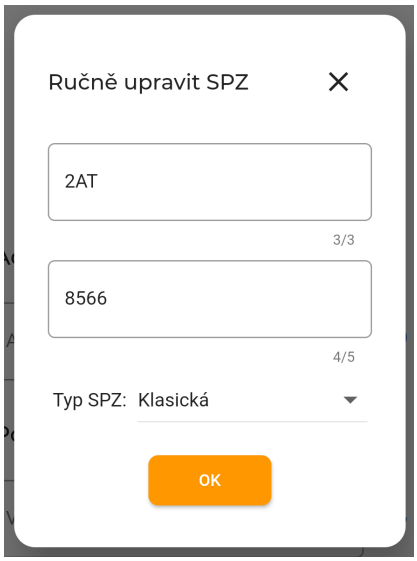
Vzhled zobrazeného identifikátoru se mění podle typu naskenované RZ. Pod identifikátorem se nachází tlačítka pro upravení RZ. Dále se na této obrazovce nachází dvě vstupní pole pro přidání dodatečných informací k této registrační značce, kterými jsou adresa místa, kde byla naskenována, a vlastní poznámka uživatele. Pro automatické získání adresy pomocí GPS se vedle vstupního pole pro zadání adresy nachází modré tlačítko. Vedle vstupního pole pro zadání vlastní poznámky, se nachází tlačítko s ikonou mikrofonu, jež slouží pro nahrání hlasové poznámky. Během nahrávání hlasové poznámky je zobrazována aktuální délka záznamu. Pod těmito vstupními poli je zobrazen snímek, ve kterém byla registrační značka rozpoznána. V nejspodnější části obrazovky jsou umístěna tlačítka pro uložení či zrušení provedených změn.



Obrázek 6.3: Obrazovka detailu registrační značky

6.1.4 Dialog pro úpravu identifikátoru registrační značky

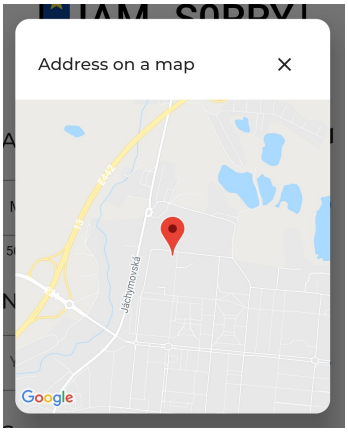
Na této obrazovce je možné k úpravě identifikátoru a typu registrační značky vyvolat dialog, pomocí kterého lze upravit naskenovanou RZ. Pomocí dvou vstupních polí lze zadat nový identifikátor registrační značky. Pro změnu typu RZ se pod těmito poli nachází rozbalovací menu, v němž jsou zobrazeny dostupné typy registračních značek, které lze zvolit.



Obrázek 6.4: Obrazovka editace identifikátoru registrační značky

6.1.5 Okno s vyznačením místa naskenování na mapě

Pokud jsou k naskenované registrační značce dostupné i souřadnice lokace, kde byla tato značka rozpoznána, tak je možné zobrazit okno, ve kterém je na zobrazené mapě vyznačené místo naskenování. Zobrazení map je dosaženo pomocí služby Google Maps a vyžaduje připojení k internetu.

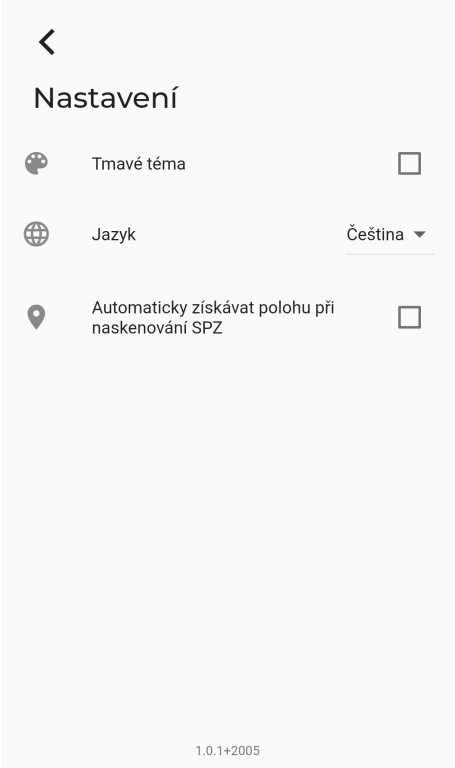


Obrázek 6.5: Okno zobrazující polohu na mapě, kde byla RZ naskenována.

6.1.6 Obrazovka s nastavením aplikace

Pro úpravu nastavení aplikace slouží právě tato obrazovka, kde jsou pro přepínání jednotlivých nastavení zobrazena zaškrtávací políčka. Pro výběr jazyka aplikace je použito rozbalovací menu, kde položkami jsou jednotlivé

dostupné jazyky, jež aplikace podporuje. Ve spodní části obrazovky je zobrazena aktuální verze aplikace, která je na zařízení nainstalována.



Obrázek 6.6: Obrazovka nastavení aplikace

6.2 Tmavý režim aplikace

V dnešní době je velmi časté, že většina aplikací, a to nejen ty mobilní, uživatelům poskytuje možnost přepnout aplikaci do tmavého režimu. Touto funkcí aplikace vytvořená v rámci této práce také disponuje a všechny obrazovky jsou tedy dostupné i v tmavé verzi. Přepínání mezi světlým a tmavým režimem lze provést v nastavení aplikace a nevyžaduje restart aplikace.

6.3 Jazyk aplikace

Uživatelské rozhraní této aplikace je dostupné jak v českém, tak ale i v anglickém jazyce, kdy přepínání mezi těmito jazyky lze provést na obrazovce nastavení aplikace a nevyžaduje restart aplikace. Při prvním spuštění aplikace je jazyk zvolen dle jazyku zařízení.

7 Vývoj mobilní aplikace k rozpoznávání registračních značek

V rámci analýzy byl jako framework pro vytvoření multiplatformní aplikace zvolen Flutter.

7.1 Správa stavů aplikace

Při vývoji mobilních aplikací ve frameworku Flutter se často setkáváme s pojmem **state management**, jež lze volně přeložit do češtiny jako **správa stavů aplikace**. Tento pojem se při využití tohoto frameworku používá proto, že na rozdíl od vývoje nativních aplikací se pro tvorbu uživatelského rozhraní v knihovně Flutter využívá spíše deklarativního paradigmatu, kdy se jasně deklarují jednotlivé stavy, ve kterých se aplikace může nacházet. Základní myšlenku tvorby uživatelského rozhraní v tomto frameworku představuje následující rovnost:

$$U = f(S),$$

kde U představuje stav uživatelského rozhraní aplikace a S stav aplikace.

V praxi to znamená, že v celém kódu aplikace Flutter například nelze nalézt žádné imperativní volání nastavení textu komponenty pro zobrazení textu na obrazovce, zatímco se například na platformě Android v nativním vývoji pro tuto operaci používá metoda `setText`. Každá část aplikace může mít vlastní stav, který se v kódu implementuje tak, že se vytvoří potomek třídy `State` a nadefinují se jednotlivé atributy tohoto stavu. Při aktualizaci stavu se tato celá část aplikace vykreslí znovu. Tento koncept lze nejlépe pochopit na následující ukázce kódu:

```
1  class _MyHomePageState extends State<MyHomePage> {
2    int _counter = 0;
3    void _incrementCounter() {
4      // setState method forces the UI to redraw by
5      // saying the state of the screen has changed
6      setState(() {
7        _counter++;
8      });
9    }
```

```
10
11    @override
12    Widget build(context) {
13      return Scaffold(
14        body: Center(
15          child: Text(
16            'Counter value: ' + _counter.toString(),
17          ),
18        ),
19        floatingActionButton: FloatingActionButton(
20            onPressed: _incrementCounter),
21      );
22    }
23  }
```

Ukázka kódu 7.1: Jednoduchá demonstrace aktualizace uživatelského rozhraní ve frameworku Flutter.

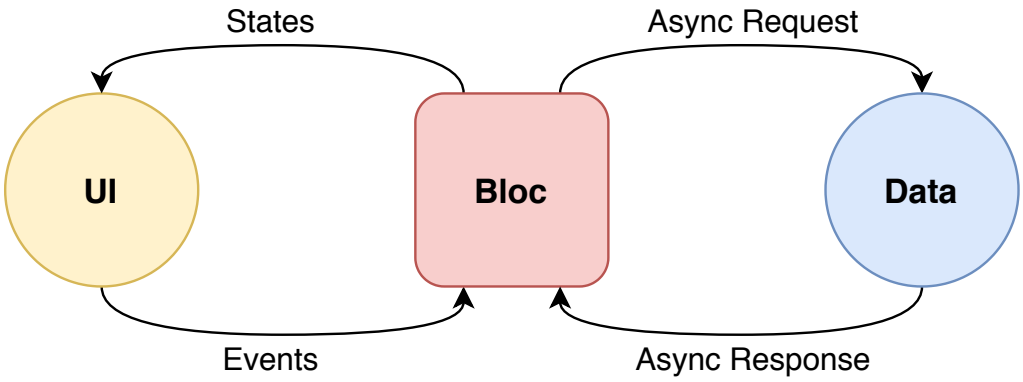
Avšak při vývoji složitější aplikace brzy zjistíme, že při definici velké množiny stavů se bez sofistikovanější správy stavů kód aplikace brzy stane nepřehledným. Přístup ukázaný v ukázce 7.1 slouží jen k demonstraci konceptu aktualizace uživatelského rozhraní a nepředstavuje vhodné řešení managementu stavů aplikace. V praxi se zásadně používají komunitou vytvořené knihovny, mezi které patří například `provider`, `MobX`, `redux` nebo `flutter_bloc`. Jelikož aplikace pro rozpoznávání registračních značek vozidel je netriviální aplikací, tak pro správu stavů složitějších obrazovek a komponent byla zvolena knihovna `flutter_bloc`, jež implementuje architekturu **BloC**.

7.1.1 Použitá architektura pro správu stavů

Ve vytvořené aplikaci je pro správu stavů použita architektura BloC (Bussiness Logic Components), jež rozděluje kód aplikací do tří hlavních částí:

- **data** – datová část je zodpovědná za uložení a manipulaci s daty aplikace;
- **aplikační logika** – úkolem této vrstvy je reagovat na události z prezentační vrstvy novými stavy aplikace;
- **prezentační vrstva** – uživatelské rozhraní aplikace je vykreslováno na základě získaného stavu z aplikační logiky.

Obsluha jednotlivých událostí je asynchronní a prezentační vrstvě může být během jedné obsluhy události předáno několik stavů.



Obrázek 7.1: Diagram architektury BloC

7.1.2 Management stavů obrazovky vyhledávání RZ

Pro management stavů obrazovky vyhledávání RZ je vytvořena komponenta aplikační logiky `VrpFinderBloc`, která reaguje na následující události:

- **LoadCamera** – událost, která je vyvolána při otevření obrazovky pro vyhledávání RZ. Tato logika v případě nalezení dostupného fotoaparátu a jeho úspěšné inicializace vrátí stav `CameraLoaded`. Během inicializace fotoaparátu je prezentační vrstvě vrácen stav `CameraLoading`. V případě nenalezení fotoaparátu či jeho neúspěšné inicializace je vrácen stav `CameraFailure`. Obsluha této události dále zahajuje zpracování snímků z fotoaparátu za účelem rozpoznání registrační značky.
- **VrpFound** – tato událost je vyvolána, pokud je během zpracování některého ze snímků nalezena registrační značka. Tato událost obsahuje informace o nalezené RZ. Obsluha této události vrátí stav `VrpFoundState`.
- **ShowLoadingScreen** – obsluha této události pouze vrací stav `CameraLoading`. Vyvolána je v případě nalezení registrační značky ve snímku, kdy se RZ ukládá do databáze a snímek je ukládán do souboru.

7.1.3 Management stavu obrazovky editace RZ

Pro management stavů obrazovky, jež umožňuje editaci rozpoznané registrační značky, je vytvořena komponenta aplikační logiky `VrpScreenBloc`, která reaguje na následující události:

- **GetAddressByPosition** – událost, která je vyvolána, pokud uživatel chce automaticky načíst svou lokaci pomocí GPS. Dle toho, jestli

uživatel udělil aplikaci přístup k poloze zařízení a poloha byla úspěšně získána, tak je vrácen stav `PositionLoaded` nebo `PositionFailure`.

- **VrpRescanned** – událost, jež je vyvolána, pokud uživatel znovu naskenoval registrační značku. Obsluha této události nevrací žádný stav, pouze smaže z disku dočasný soubor předtím naskenované registrační značky.

- **DiscardVRP** – událost, jež je vyvolána, pokud uživatel zavře editaci RZ. Pokud byl předtím vytvořen dočasný soubor obsahující snímek rozpoznané registrační značky, tak je tento soubor v rámci této obsluhy smazán.

- **SubmitVRP** – pokud uživatel uloží editaci naskenované registrační značky, tak je vyvolána tato událost. Její obsluha má na starost zapsat registrační značku do databáze, zkopírovat dočasný soubor obsahující snímek registrační značky, který je následně smazán. Pokud byla nahrána hlasová poznámka, tak se souborem obsahujícím tuto poznámku jsou prováděny stejné operace, jako se souborem obsahující snímek s registrační značkou. Na konci obsluhy této události je vrácen stav `VrpSubmitted`.

7.1.4 Management stavu komponenty pro nahrávání hlasové poznámky

Na obrazovce editace registrační značky se nachází komponenta sloužící pro vytvoření hlasové poznámky, k jejíž správě stavů slouží aplikační logika `VrpRecordingBloc` a obsluhuje následující události:

- **RecordingStarted** – událost, jež je vyvolána, když uživatel začne nahrávat hlasovou poznámku. Obsluha této události začne, pokud uživatel aplikaci udělil oprávnění používat mikrofon zařízení, nahrávat zvuk z mikrofonu. Obsluha této události vyvolává v průběhu nahrávání události `RecordingUpdated`. Pokud během nahrávání zvukového záznamu dojde k chybě, je vrácen stav `RecordingFailed`.

- **RecordingUpdated** – událost, která je vyvolávána periodicky během nahrávání audio záznamu a nese informaci o délce aktuálně nahrávaného záznamu. Obsluha této události vrací stav `RecordingInProgress`.

- **RecordingStopped** – v momentě, kdy uživatel zastaví nahrávání záznamu, je vyvolána tato událost, jejíž obsluha, v případě úspěšného

dokončení nahrávání, vrátí stav `RecordingSuccess`. Pokud se nahrávání nepodaří korektně ukončit, je vrácen stav `RecordingFailed`.

- **PlaybackStarted** – událost, jež je vyvolána, když uživatel začne přehrávat nahranou hlasovou poznámku. Obsluha této události vyvolává v průběhu přehrávání události `PlaybackUpdated`. Pokud je již záznam přehrán celý, je vyvolána událost `PlaybackStopped`. Pokud se nepodařilo spustit přehrávání nahraného záznamu, tak je vrácen stav `RecordingFailed`, čímž uživatel získá možnost nahrát novou poznámku.

- **PlaybackUpdated** – událost, která je vyvolávána periodicky během přehrávání audio záznamu a nese informaci o stavu přehrávání nahraného záznamu. Obsluha této události vrací stav `PlaybackInProgress`.

- **PlaybackStopped** – v momentě, kdy uživatel zastaví přehrávání záznamu nebo je záznam přehrán celý, je vyvolána tato událost, jejíž obsluha zastaví přehrávání záznamu a vrátí stav `RecordingSuccess`. Pokud se nepodaří ukončit přehrávání záznamu, je vrácen stav `RecordingFailed`.

- **RecordRemoved** – pokud chce uživatel smazat hlasovou poznámku, je vyvolána tato událost, jejíž obsluha smaže soubor obsahující audio záznam, pokud existuje.

7.1.5 Management stavu komponenty zobrazující snímek obsahující RZ

Na obrazovce, kde je zobrazena rozpoznaná RZ, se nachází komponenta, která zobrazuje snímek, ve kterém byla nalezena registrační značka. Při interakci s touto komponentou je rozpoznaná registrační značka v zobrazeném snímku zvýrazněna. Pro správu stavů této komponenty byla vytvořena komponenta aplikační logiky `VrpSourceDetailBloc`, která obsluhuje následující události:

- **Highlighted** – obsluha této události, jež je vyvolána, pokud uživatel chce zvýraznit ve snímku RZ, vrátí stav `DetailHighlighted`.

- **NotHighlighted** – obsluha této události, jež je vyvolána, pokud uživatel již nechce, aby v zobrazeném snímku byla zvýrazněna RZ, vrátí stav `ClassicDetail`.

7.2 Implementace rozpoznávání registračních značek

Jelikož hlavním požadavkem na aplikaci byla možnost rozpoznávat registrační značky v reálném čase, bylo třeba zvolit takový postup rozpoznávání, který není časově náročný a zároveň disponuje přijatelnou přesností.

Jako jedna z prvních možností, jak provádět rozpoznávání značek ve snímku, se jevilo použití knihovny OpenCV, která programátorovi poskytuje kvalitní a výkonné API, jež implementuje mnoho metod z oblasti počítačového vidění a zpracování digitálního obrazu. Bohužel v době psaní této práce není tato knihovna v rámci žádného pluginu ve Flutteru dostupná. Možností by bylo takový plugin, jež zpřístupňuje API knihovny OpenCV, vytvořit, avšak to by znamenalo naimportovat tuto knihovnu na obou platformách a následně na každé platformě provést mapování nativního API OpenCV na metody, které by byly volané z aplikace Flutter.

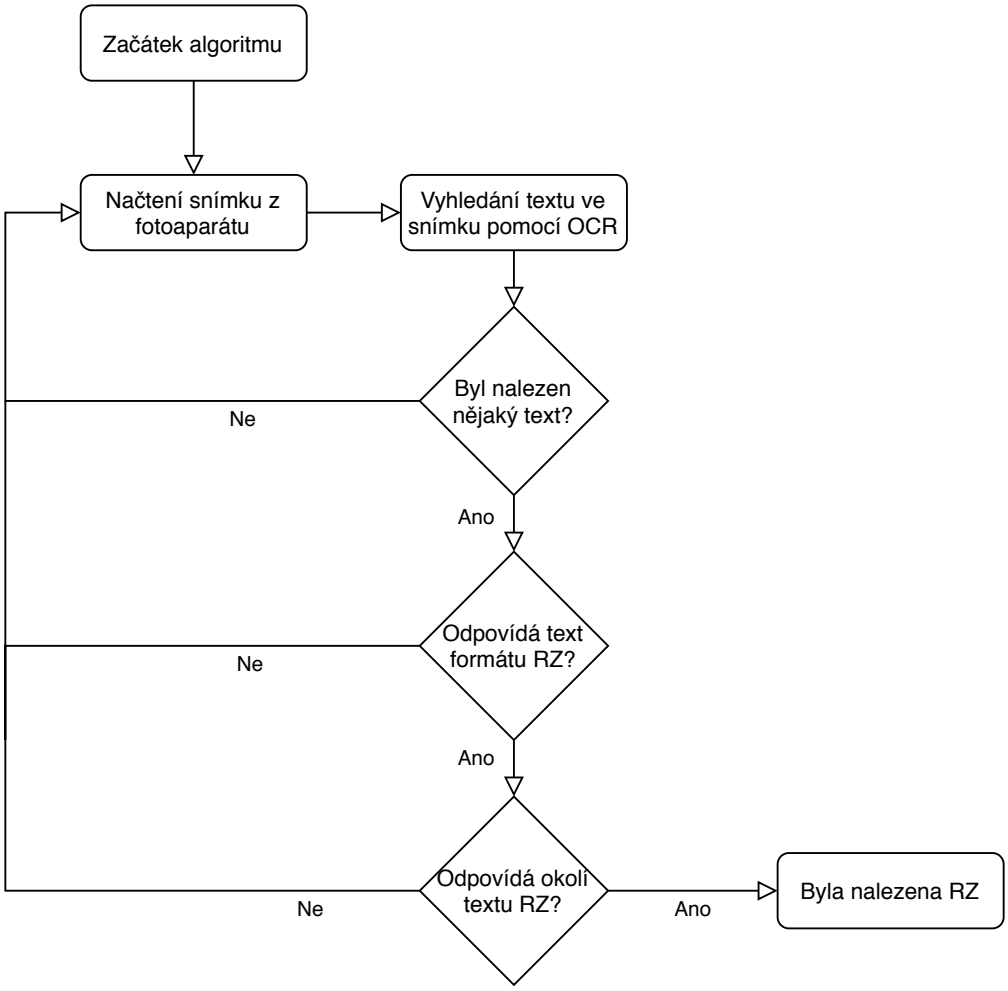
Pro implementaci rozpoznávání registračních značek byl tedy následně zvolen jednoduchý postup, který je dostatečně rychlý, aby mohl být použit v reálném čase: V každém zpracovávaném snímku se vyhledává text pomocí knihovny Firebase ML Kit. Při spuštění skenování se začne zpracovávat výstup z fotoaparátu. Během zpracovávání jednoho snímku jsou ostatní snímky, které fotoaparát vysílá, zahazovány. Právě z tohoto důvodu je důležité, aby zpracování jednoho snímku netrvalo příliš dlouho, jelikož by za daný časový úsek, kdy se zpracovával nějaký snímek, bylo zahozeno příliš mnoho snímků a celý proces skenování by se uživateli jevil jako nepřesný a pomalý.

Důležité také je, aby zpracování každého snímku bylo prováděno mimo vlákno uživatelského rozhraní. V opačném případě by se totiž zastavila aktualizace hledáčku fotoaparátu.

7.2.1 Úvodní vyhledání možných RZ ve snímku

S využitím knihovny Firebase ML Kit, jejíž metoda pro provedení OCR na daném snímku vrací kolekci instancí třídy `TextBlock`, jež je potomkem abstraktní třídy `TextContainer`, která popisuje oblast ve snímku obsahující nějaký text. Tato třída obsahuje následující atributy:

- `text` – nalezený text v oblasti;
- `boundingBox` – obdélník ohraničující oblast s nalezeným textem, který byl zarovnán tak, aby jeho jednotlivé strany byly rovnoběžné s horizontální a vertikální osou snímku;



Obrázek 7.2: Zjednodušený diagram rozpoznávání RZ

- `cornerPoints` – body, které definují čtyřúhelník, jenž přesně ohraničuje oblast textu. První bod v této kolekci představuje levý horní bod obdélníku;
- `confidency` – představuje míru jistoty, se kterou tato knihovna určila, že se jedná o oblast s textem. Tento atribut není při lokálním provedení OCR vyplněn. Atribut je vyplněn pouze tehdy, pokud se snímek odesílá ke zpracování do cloudu;
- `recognizedLanguages` – představuje kolekci jazyků, které byly v oblasti nalezeny.

Třída `TextBlock` také obsahuje atribut `lines`, jenž představuje kolekci oblastí ve snímku představující řádky textu. Jednotlivé řádky následně obsahují atribut `elements`, který reprezentuje jednotlivá slova na řádce.

Po získání oblastí obsahujících text ve snímku se po této kolekci iteruje a vyhledávají se takové oblasti, jež obsahují text, který splňuje požadavky formátu identifikátoru registračních značek v České republice. Tato metoda rozpoznává typy registračních značek uvedené v obrázku 2.1. Dále také rozpoznává historické registrační značky a registrační značky na přání.

Kontrola probíhá tak, že se dle počtu řádků a délky jednotlivých slov rozdělí textové oblasti do dvou skupin. Do první skupiny jsou přidány ty textové oblasti, jež by mohly obsahovat jednořádkové registrační značky. Do druhé skupiny jsou přidány takové oblasti, které by mohly obsahovat dvouřádkové registrační značky. Pokud jednotlivá slova odpovídají jednotlivým částem identifikátoru, který se může objevovat na registračních značkách, tak se ještě dále kontroluje vzdálenost a jejich vzájemné posunutí mezi jednotlivými částmi.

Kontrola formátu jednotlivých typů značek je implementována tak, že je velmi jednoduché přidat do aplikace další podporované typy. Stačí vytvořit potomka abstraktní třídy `VRPValidator`, naimplementovat požadovanou metodu pro validaci textové oblasti a přidat její instanci do seznamu validátorů registračních značek. Kontrola končí při první úspěšné validaci textové oblasti.

Pokud je nalezena alespoň jedna textová oblast, jejíž text splňuje formát identifikátoru registračních značek, tak se na snímek aplikují další kontroly.

7.2.2 Prahování oblasti kandidáta na registrační značku

Pro každou textovou oblast, jež byla nějakým validátorem označena jako kandidát obsahující registrační značku, se pomocí atributu `boundingBox` této oblasti provede výřez z původního snímku. Na tento výřez, jež by mohl obsahovat registrační značku, se použije prahování pomocí metody, která automaticky stanoví práh pomocí vyvažování histogramu snímku. V naprahovaném snímku se poté vypočítá poměr mezi černou a bílou barvou. Dle možného typu značky se stanoví práh, kterým se dle poměru černé a bílé barvy vyfiltrují textové oblasti, jejichž text se nenachází na relativně uniformním pozadí. Tento práh bere v potaz i předpoklad, že na každé registrační značce má vůči bílému pozadí černý text určitý poměr.

Pro lepší analýzu jednotlivých pixelů daného snímku musí být snímek převeden do formátu ARGB, který je vyžadován při použití pluginu `Image` pro práci se snímky. Jelikož tato konverzní operace zabírá nezanedbatelné množství času, je oproti prvním verzím aplikace prováděna až na konec celého procesu rozpoznávání, a to pouze v případě, že byly nalezené nějaké oblasti s možným výskytem registračních značek.

Toto prahování může mít negativní vliv na rozpoznávání značek, které jsou nějakým způsobem výrazně poškozeny či znečištěny, avšak v praxi se s tímto jevem setkáváme málokdy.

Nutné je zmínit, že kontrola oblasti na základě prahování s využitím automatického stanovení prahu nedokáže vždy odfiltrovat falešně pozitivní výsledky. Tato skutečnost je způsobena i tím, že oblast ohraničující nalezený text se během jednoho záběru napříč snímky z fotoaparátu mění. Mění se tedy i poměr textu k ploše oblasti. Často se tedy stává, že uvnitř oblasti se nachází i jiné části vozidla, než pouze registrační značka, což negativně ovlivňuje validaci dle porovnání poměru černé a bílé barvy v naprahované oblasti.

Tato metoda také vychází z předpokladu, že hlavní využití aplikace je takové, že ve scéně, již fotoaparát zachycuje, se většinou objevuje pouze jedna textová oblast, a to konkrétně oblast obsahující registrační značku. Pravděpodobnost, že se ve scéně objevují i jiné textové oblasti s takovým obsahem, který by odpovídal formátu registrační značky, je malá. Pro toto využití aplikace je tedy vhodné učinit kompromis, kdy za cenu občasného výskytu falešně pozitivních výsledků je zvýšena rychlost rozpoznávání.

7.3 Perzistentní uložení naskenovaných RZ

Jedním z požadavků na tuto práci byla možnost ukládat naskenované registrační značky do lokální databáze. Jako databázový systém byl zvolen SQLite, který se v oblasti vývoje mobilních aplikací hojně používá při potřebě ukládat perzistentní strukturovaná data do lokálního úložiště. Typ systému, jenž umožňuje tvořit relační databáze, byl zvolen z důvodu předpokladu, že tato práce může být dále rozšířena a nad uloženými záznamy budou tvořeny nějaké relace.

Pro použití databází typu SQLite se v aplikacích vytvořených ve frameworku Flutter používá knihovna `sqflite`, jež se stará o implementaci prostředků pro práci s databázemi tohoto typu na platformách iOS, Android a macOS. V této práci byla použita knihovna `moor`, jež poskytuje abstrakci nad knihovnou `sqflite` a zjednodušuje tak práci s databází. Prakticky není nutné, aby programátor ve svém programu psal nějaké dotazy v jazyce SQL, jelikož je tato knihovna pomocí generování kódu za něj vytvoří. Při použití této knihovny není dokonce ani potřeba psát skript pro definici tabulky a jejích sloupců, protože tento skript je automaticky vygenerován podle třídy, která představuje definici jednoho řádku tabulky. Automatické vygenerování kódu a skriptů se provádí spuštěním příkazu `flutter packages pub run`

`build_runner build` v kořenovém adresáři projektu.

V databázi aplikace, jež byla vytvořena v rámci této práce, se nachází pouze jedna tabulka, jejíž záznamy představují již rozpoznané a uložené RZ. Tato tabulka, která je identifikována názvem `found_vrp_records`, obsahuje následující sloupce:

- **id** typu `int` – unikátní identifikátor záznamu;
- **first_part** typu `text` – první část identifikátoru na naskenované značce;
- **second_part** typu `text` – druhá část identifikátoru na naskenované značce;
- **type** typu `int` – kód typu RZ: Hodnota `0` představuje klasickou jednořádkovou RZ, `1` historickou RZ, `2` RZ na vlastní přání, `3` dvouřádkovou RZ používanou pro identifikaci motocyklů a `4` dvouřádkovou RZ o rozměrech 340 x 200 mm;
- **latitude** typu `double` – představuje zeměpisnou šířku lokace, kde byla RZ naskenována;
- **longitude** typu `double` – představuje zeměpisnou délku lokace, kde byla RZ naskenována;
- **address** typu `text` – adresa lokace, kde byla RZ naskenována;
- **note** typu `text` – poznámka uživatele k naskenované RZ;
- **date** typu `datetime` – datum, kdy byla RZ naskenována;
- **sourceImagePath** typu `text` – cesta k souboru na disku, který představuje snímek, ze kterého byla RZ rozpoznána;
- **audioNotePath** typu `text` – cesta k souboru na disku představující audio záznam, který slouží jako poznámka uživatele k danému záznamu;
- **top** typu `int` – vertikální souřadnice bodu, jenž ve snímku, ze kterého byla RZ rozpoznána, označuje levý horní roh obdélníku ohraničujícího RZ;
- **right** typu `int` – horizontální souřadnice bodu, jenž ve snímku, ze kterého byla RZ rozpoznána, označuje levý horní roh obdélníku ohraničujícího RZ;

- **width** typu `int` – šířka obdélníku, který ve snímku, ze kterého byla RZ rozpoznána, ohraničuje rozpoznanou RZ;
- **height** typu `int` – výška obdélníku, který ve snímku, ze kterého byla RZ rozpoznána, ohraničuje rozpoznanou RZ.

7.4 Získání polohy zařízení

Funkce automatického získání polohy zařízení při naskenování registrační značky vozidla se v praxi jeví jako velmi užitečná, jelikož nás často při zpětném zkoumání naskenovaných RZ zajímá i místo, kde byl záznam pořízen, avšak při pořizování tohoto záznamu, například při vzniku pojistné události, uživatel často nemá čas na ruční vyplňování adresy do vstupního pole. Proto by tedy bylo vhodné, kdyby aplikace sama dokázala určit polohu, kde byl záznam pořízen.

Aplikace tuto funkci implementuje a k získání lokace mobilního zařízení používá knihovnu `geolocator`, jež poskytuje programátorovi jednoduché rozhraní sloužící k přístupu k službám GPS. Tato knihovna dále disponuje funkcí, která, pokud je zařízení připojené k internetu, dokáže dle zeměpisné šířky a délky nalézt adresu této lokace.

7.5 Problémy při vývoji mobilní aplikace

7.5.1 Duplicitní rozpoznání registrační značky

Tato chyba se v praxi projevovala tak, že když uživatel naskenoval nějakou registrační značku, přistoupil k jinému vozidlu a spustil skenování za účelem skenování další značky, aplikace rozpoznala a zobrazila již předtím rozpoznanou značku. Pokud se ve scéně, kde byla nalezena první značka, objevovala i jiná RZ, tak existovala jistá pravděpodobnost, že ji při druhém skenování v jiné scéně aplikace rozpozná. Tato chyba se neprojevovala zcela vždy a její výskyt při použití aplikace se často jevil jako náhodný, což ztěžovalo její odhalení. Tato skutečnost zapříčinila to, že během vývoje bylo řešení této chyby odloženo až na konec samotného vývoje, kdy bude aplikace hotová, jelikož její řešení se jevilo jako časově velmi náročné a zbytečně by tak brzdilo celý vývoj.

Odhalit a opravit tuto chybu bylo velmi složité, jelikož se chyba nevyskytovala v aplikaci samotné, nýbrž v balíčku `camera` pro použití fotoaparátu. Tento balíček je přímo spravován týmem vývojářů frameworku Flutter.

Chyba se projevovala na zařízeních s operačním systémem Android. Příčinou rozpoznání RZ z minulé scény bylo to, že rozpoznávací metoda vždy po spuštění dalšího vyhledávání obdržela zhruba 5 snímků z předešlé scény. V této scéně tedy opět nalezla již předtím rozpoznanou registrační značku. Po prozkoumání implementace knihovny pro přístup k fotoaparátu na platformě Android bylo zjištěno, že přenos snímků z fotoaparátu není v knihovně korektně ukončován a při dalším spuštění se tedy nechová správně. Opravou bylo přidání korektního ukončení přenosu a chyba tak byla vyřešena.

Oprava této chyby byla zdokumentována a nahrána do oficiálního repozitáře frameworku Flutter ve službě GitHub, kde čeká na schválení a implementaci týmem vývojářů tohoto frameworku. Do té doby je tedy aplikace, vytvořená v rámci této práce, nucena používat vlastní verzi knihovny, kde je zmíněná chyba opravena.

8 Testování implementované metody pro rozpoznávání registračních značek

V rámci této kapitoly je čtenář seznámen s postupy, které byly použity pro otestování implementované metody k rozpoznávání registračních značek vozidel.

8.1 Galerie snímků použitých k testování

K otestování metody bylo třeba pořídit sady snímků, ve kterých budou vyhledávány RZ. Jednotlivé sady by měly obsahovat snímky, které metoda obdrží při běžném použití vytvořené mobilní aplikace. Dále by tyto sady měly být pořízeny za různých světelných podmínek a zachycovat registrační značky různých vlastností, jako například znečištění nebo poškození.

Všechny snímky z galerií byly foceny vertikálně a následně transformovány tak, aby byly v HD rozlišení, tedy v rozlišení, ve kterém jsou získávány snímky z fotoaparátu při použití vytvořené mobilní aplikace.

Nutno podotknout, že kvalita snímků pořízených pomocí fotoaparátu je větší, než kvalita snímků, se kterými aplikace normálně pracuje při běžném použití. Tento fakt je způsoben tím, že při pořízení snímku pomocí fotoaparátu je tento snímek dále upraven postprocessingem, čímž je vylepšena jeho kvalita. Míra změny během postprocessingu je ovlivněna softwarem zařízení, na kterém byl snímek pořízen. Vytvořená aplikace během rozpoznávání z fotoaparátu kontinuálně odebírá snímky, které nebyly postprocessingem nijak upraveny.

8.1.1 Galerie „slunecno“

Tato galerie obsahující 21 snímků byla pořízena mobilním zařízením Oneplus 6, který disponuje snímačem Sony IMX 519 a při pořízení snímků pomocí předinstalované aplikace používá technologii **HDR** (High Dynamic Range)[1]. Jednotlivé snímky byly pořizovány kolem desáté hodiny dopoledne v prvním

[1]Fotoaparát pořídí ve stejnou dobu více snímků s použitím různé doby expozice a následně je sloučí do jednoho snímku, ve kterém jsou lépe vidět detaily stinných i světlých míst ve scéně.

týdnu dubna. Během pořizování snímků panovalo slunečné počasí. Na některých registračních značkách ve snímcích je karosérií vozidla vržen stín.

Obrázek 8.1: Ukázka snímků objevujících se v galerii „slunecno“.

8.1.2 Galerie „zatazeno“

Tato galerie, jež obsahuje 5 snímků, byla pořízena již zmíněným mobilním zařízením Oneplus 6 a jednotlivé snímky byly pořizovány kolem desáté hodiny dopoledne v předposledním týdnu února. V době pořízení snímků byla obloha polojasná a slunce výrazně neovlivňovalo osvětlení registrační značky.

Obrázek 8.2: Ukázka snímků objevujících se v galerii „zatazeno“.

8.1.3 Galerie „podvecer“

Tato galerie, která obsahuje 18 snímků, byla pořízena mobilním zařízením Oneplus 6 a jednotlivé snímky byly pořizovány kolem osmé hodiny odpoledne v prvním týdnu dubna. V době pořízení snímků byla obloha polojasná a již se v důsledku západu slunce začínaly zhoršovat světelné podmínky.

Obrázek 8.3: Ukázka snímků objevujících se v galerii „podvecer“.

8.1.4 Galerie „vecer“

Galerie „vecer“ obsahující 23 snímků, byla pořízena mobilním zařízením Oneplus 6 a jednotlivé snímky byly pořizovány kolem půl deváté hodiny večer v první týdnu dubna. V době pořízení se v důsledku západu slunce začínaly objevovat výrazně zhoršené světelné podmínky.

8.2 Postupy při testování

8.2.1 Vybrání snímků k testování

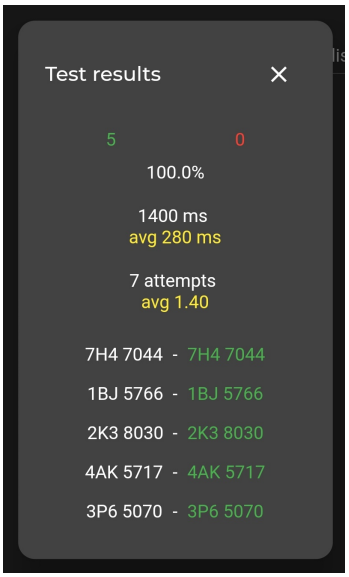
Pokud je aplikace spuštěná v ladícím režimu, je po třech klepnutích na verzi aplikace v obrazovce nastavení zobrazen dialog pro vybrání souborů, jež obsahují snímky, které budou při testování využity. Testovací metoda předpokládá, že vybrané soubory jsou speciálně pojmenovány tak, že se v názvu souboru vyskytuje identifikátor registrační značky ve snímku a že jednotlivé části tohoto identifikátoru jsou odděleny podtržítkem.

Obrázek 8.4: Ukázka snímků objevujících se v galerii „vecer“.

8.2.2 Testování implementované metody na jednotlivých snímcích

Po vybrání snímků určených k testování je na tyto snímky postupně v rámci testovacího scénáře aplikována metoda rozpoznávání registrační značky. Pokud ve snímku není nalezena registrační značka, je celý snímek pootočen o jednotky stupňů, aby bylo simulováno reálné použití aplikace, kdy je v zachycené scéně průběžně vyhledávána registrační značka a uživatel nedrží zařízení zcela bez třesu ruky. Pokud není registrační značka nalezena ani po sedmém pokusu, je výsledek aktuálního scénáře označen jako negativní. Pokud je během nějakého pokusu nalezena registrační značka, ale její identifikátor nebyl správně rozpoznán, je aktuální scénář označen také jako negativní a další pokusy již nejsou prováděny.

Po dokončení testování vybraných snímků se v aplikaci zobrazí dialog, který obsahuje výsledky daného testu. Je zobrazen počet správně rozpoznaných značek, špatně rozpoznaných značek a celková přesnost rozpoznávání. Dále je uveden celkový čas rozpoznávání, průměrný čas potřebný k dokončení jednoho scénáře a počet pokusů o rozpoznání registrační značky včetně průměrného počtu pokusů na jeden test. Následuje seznam, ve kterém jsou zobrazeny výsledky jednotlivých testů. Při kliknutí na řádek zobrazující výsledek testu je zobrazen snímek použitý v daném testu.



Obrázek 8.5: Dialog zobrazující výsledek testování.

8.3 Výsledky testování

Výsledky testování jsou zaneseny do následující tabulky, kde jednotlivé sloupečky mají následující význam:

- **Galerie** – název galerie, jejíž snímky byly testovány;
- **P** – počet testovacích scénářů, ve kterých byla správně rozpoznána RZ;
- **FN** – počet testovacích scénářů, ve kterých nebyla rozpoznána žádná RZ;
- **FP** – počet testovacích scénářů, ve kterých byla nalezena nesprávná RZ;
- a – celkový počet pokusů, které byly provedeny napříč testovacími scénáři;
- $\overline{a}$ – průměrný počet pokusů na jeden testovací scénář;
- t – celkový čas rozpoznávání během celého testu;
- $\overline{t}$ – průměrný čas rozpoznávání během jednoho testovacího scénáře;
- **Přesnost** [%] – podíl celkového počtu testovacích scénářů ku scénářům, ve kterých byla správně rozpoznaná RZ, vyjádřený v procentech.

Galerie	P	FN	FP	a	$\overline{a}$	t [ms]	$\overline{t}$ [ms]	Přesnost [%]
slunecno	21	0	0	33	1,57	6769	322	100
zatazeno	5	0	0	7	1,40	1288	258	100
podvecer	17	1	0	30	1,67	6184	344	94,45
vecer	19	3	1	52	2,26	10093	439	82,60

Tabulka 8.1: Výsledky testování provedené na zařízení Oneplus 6.

8.4 Zhodnocení výsledků testů

Na základě získaných výsledků testování lze pozorovat, že implementovaná metoda dosahuje dobrých výsledků, a to i za zhoršených světelných podmínek. Při zhoršení světelných podmínek přesnost rozpoznávání klesá a počet potřebných pokusů k rozpoznání RZ stoupá, avšak testovanou metodu lze stále považovat za použitelnou.

Hlavním důvodem, proč nejsou RZ při zhoršených podmínkách rozpoznávány, je špatné určení prahu při prahování oblasti registrační značky, jenž je určován automaticky pomocí metody vyváženého histogramu. Při důkladnějším pohledu na to, co se děje při automatickém určování prahu, lze pozorovat, že určený práh je příliš vysoký a následně tak dojde k nevhodné segmentaci. Následný výpočet poměru černé a bílé barvy tak v oblasti udává příliš velkou převahu černé barvy nad bílou, a tak je registrační značka zamítnuta, i přesto, že OCR správně určí identifikátor registrační značky.

Z výsledků je také možné konstatovat, že implementovaná metoda obecně potřebuje krátký čas k rozpoznání registrační značky ve snímku. V následující tabulce jsou zaneseny časy rozpoznávání v testech spuštěných na zařízení Xiaomi Mi A1, jež svými hardwarovými parametry spadá do střední třídy mobilních telefonů a na trh bylo uvedené v třetím čtvrtletí roku 2017.

Galerie	t [ms]	$\overline{t}$ [ms]
slunecno	19119	911
zatazeno	3800	760
podvecer	16662	926
vecer	27682	1204

Tabulka 8.2: Časy testování provedené na zařízení Xiaomi Mi A1.

9 Závěr

Během vypracovávání této práce se její autor seznámil s problematikou návrhu a implementace metod k rozpoznávání registračních značek motorových vozidel.

S využitím frameworku Flutter byla vytvořena rozšiřitelná multiplatformní mobilní aplikace určená pro operační systémy Android a iOS, jež umožňuje v reálném čase pomocí fotoaparátu vyhledávat ve scéně české registrační značky několika typů, které lze následně uložit do lokální databáze. Ke každé registrační značce má uživatel možnost přidat vlastní textovou nebo hlasovou poznámku. Pokud si to uživatel přeje, tak je dále při pořízení záznamu možné automaticky získat pomocí služby GPS polohu zařízení, která je uložena spolu s registrační značkou.

Kvalita implementované metody k rozpoznávání registračních značek byla důkladně otestována snímáním scén zachycujících odlišná vozidla za různých světelných podmínek. Výsledky jednotlivých testů jsou uvedeny v kapitole 8 a na jejich základě lze konstatovat, že i navzdory své jednoduchosti dokáže tato metoda relativně s velkou přesností rozpoznávat registrační značky i za zhoršených světelných podmínek v krátkém čase.

Z důvodu absence podpory knihovny OpenCV pro framework Flutter nebylo použito mnoha technik z oblasti počítačového vidění, a pro vyhledávání RZ ve snímku je tak primárně využíváno optického rozpoznávání znaků pomocí knihovny Firebase ML Kit. Jelikož je framework Flutter poměrně mladý a stále se aktivně vyvíjí, lze v budoucnosti očekávat, že bude přidána podpora pro knihovnu OpenCV, pomocí které by bylo možné nad jednotlivými snímky provádět pokročilejší techniky rozpoznávání, a dosáhnout tak vyšší přesnosti i za zhoršených světelných podmínek.

V rámci vývoje této aplikace autor narazil na chybu v knihovně pro používání fotoaparátu zařízení a její opravu pomocí služby GitHub navrhnul vývojářům této knihovny, čímž přispěl do vývoje otevřeného software.

Tato práce splňuje všechny body zadání a výsledná aplikace je vhodná k osobnímu použití jednotlivci či k následnému rozšíření za účelem konkrétního použití v nějaké firmě či instituci.

Přehled použitých zkratek

- **OS** – operační systém
- **RZ** – registrační značka
- **OOS** – Open Source Software (software s volně dostupným zdrojovým kódem)
- **SDK** – Software Development Kit (balíček pro vývoj software)
- **NDK** – Native Development Kit (balíček pro vývoj software pro Android v jazycích C a C++)
- **API** – Application Programming Interface (rozhraní pro programování aplikací)
- **HDR** – High Dynamic Range (technika vylepšení fotografií s využitím několika expozicí při pořizování fotografie)
- **BloC** – Business Logic Component (komponenty programové logiky)

Literatura

[1] *Firebase* [online]. Google, 2019. [cit. 3.3.2020]. Dostupné z: https://firebase.google.com/.

[2] *The State of the Octoverse* [online]. GitHub, 2019. [cit. 5.1.2020]. Dostupné z: https://octoverse.github.com/.

[3] Anjos, A. – Shahbazkia, H. Bi-Level Image Thresholding - A Fast Method. *BIOSIGNALS*. 01 2008, 2, s. 70–76.

[4] Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, 2016. ISBN 0-387-31073-8.

[5] *CameraX Overview | Android Developers* [online]. Google, 2019. [cit. 17.12.2019]. Dostupné z: https://developer.android.com/training/camerax.

[6] *FAQ - Flutter* [online]. Google, 2019. [cit. 2.1.2020]. Dostupné z: https://flutter.dev/docs/resources/faq#what-is-flutter.

[7] *Operating System Market Share Worldwide* [online]. StatCounter Global Stats, 2019. [cit. 15.12.2019]. Dostupné z: https://gs.statcounter.com/os-market-share.

[8] Phan, K. *Get to know different JavaScript environments in React Native* [online]. Free Code Camp, 2019. [cit. 5.1.2020]. Dostupné z: https://www.freecodecamp.org/news/get-to-know-different-javascript-environments-in-react-native-4951c15d61f5/.

[9] Sinha, U. *Introduction the Canny edge detector* [online]. AI Shac, 2020. [cit. 23.03.2020]. Dostupné z: https://aishack.in/tutorials/canny-edge-detector/.

[10] Smith, R. *Tesseract OCR* [online]. GitHub, 2020. [cit. 8.3.2020]. Dostupné z: https://github.com/tesseract-ocr/tesseract.

[11] *What is Xamarin? - Xamarin | Microsoft* [online]. Microsoft, 2019. [cit. 31.12.2019]. Dostupné z: https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin.

A Uživatelská příručka

A.1 Sestavení a instalace aplikace

V této sekci bude popsáno sestavení a instalace aplikace pro operační systémy Android verze 5.0 (úroveň API 21) a novější.

K sestavení instalačního balíčku APK, který se používá při instalaci aplikací na Android OS, je třeba nainstalovat Android SDK[1]. Dále je potřeba mít na zařízení, kde chceme sestavit balíček APK, nainstalované Flutter SDK[2]. Po instalaci výše zmíněného softwaru se přesuneme do kořenového adresáře přiloženého projektu a spustíme příkaz `flutter packages get`, čímž se stáhnou všechny potřebné závislosti projektu. Po získání závislostí provedeme příkaz `flutter build apk`, po jehož dokončení vznikne v adresáři `build/app/outputs/apk/release` instalační balíček `app-release.apk`. Pro vytvoření balíčku s ladící verzí aplikace spusťte v kořenovém adresáři projektu příkaz `flutter build -debug`, po jehož dokončení vznikne v adresáři `build/app/outputs/apk/debug` balíček `app-debug.apk`. Instalace balíčku na fyzické zařízení můžeme dosáhnout dvěma způsoby:

- Přesunutím instalačního balíčku na své fyzické zařízení s operačním systémem Android a následnou instalací tohoto balíčku tak, že ho v zařízení otevřeme pomocí prohlížeče souborů nebo instalátoru balíčků APK.
- Připojením fyzického zařízení k počítači pomocí kabelu USB a využitím softwaru `adb`, jež je součástí Android SDK. Následným spuštěním příkazu `adb install app-release.apk` a jeho úspěšným dokončením je aplikace na fyzickém zařízení nainstalována.

A.2 Ovládání aplikace

Ovládání aplikace je stejné jako u běžných mobilních aplikacích. Změnu jazyku a motivu aplikace lze provést v nastavení aplikace, jež se zobrazí po kliknutí na ikonu ozubeného kola v levém horní rohu úvodní obrazovky. Popis jednotlivých obrazovek se nachází v kapitole 6.

[1] dostupné na `https://developer.android.com/studio/`

[2] návod dostupný na `https://flutter.dev/docs/get-started/install`