CENTER FOR
MACHINE PERCEPTION

CZECH TECHNICAL
UNIVERSITY IN PRAGUE

DIPLOMOVÁ PRÁCE

ISSN 1213-2365

Video s elektronickou štěrbinovou závěrkou pro 3D rekonstrukci

Michal Polic

policmic@fel.cvut.cz

CTU–CMP–2015–01

Květen 10, 2015

Vedoucí Ing. Tomáš Pajdla, Ph.D.

Research Reports of CMP, Czech Technical University in Prague, No. 1, 2015

Published by

Centrum strojového vnímání, Katedra kybernetiky
Fakulta elektrotechnická ČVUT
Technická 2, 166 27 Praha 6
fax: (02) 2435 7385, tel: (02) 2435 7637, www: http://cmp.felk.cvut.cz

Video s elektronickou štěrbinovou závěrkou pro 3D rekonstrukci

Michal Polic

Květen 10, 2015

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Michal P o l i c

Studijní program: Otevřená informatika (magisterský)

Obor: Počítačové vidění a digitální obraz

Název tématu: Video s elektronickou štěrbinovou závěrkou pro 3D rekonstrukci

Pokyny pro vypracování:

Cílem diplomové práce je prozkoumat možnosti pořízení video záznamu na mobilních zařízeních s obrazovým senzorem s elektronickou štěrbinovou závěrkou, implementovat pořízení videa, kalibraci jeho parametrů a demonstrovat, že jej lze použít k rekonstrukci polohy kamery v prostoru.

1. Prostudujte a na experimentech demonstrujte možnosti pořízení videa na mobilním zařízení s operačním systémem Android a s obrazovým senzorem s elektronickou štěrbinovou závěrkou. Analyzujte a popište časování pořízení snímku a parametry závěrky pro jedno konkrétní zařízení.
2. Implementujte metodu pořízení videa a demonstrujte ji na úloze výpočtu polohy kamery na základě pozorování známé scény. Implementujte detekci referenční značky v obraze s využitím knihovny OpenCV [1] a výpočet polohy kamery P3P [2]. Demonstrujte funkčnost implementace na reálném zařízení.
3. Rozšiřte bod 2) na kameru s elektronickou štěrbinovou závěrkou. Použijte výsledek [3] a demonstrujte funkčnost implementace na reálném zařízení.

Seznam odborné literatury:

[1] Android. In: OPENCV: Open Source Computer Vision [online]. c2015- [cit. 2015-02-17]. Dostupné z: http://opencv.org/platforms/android.html.
[2] T. Pajdla: Elements of Geometry for Computer Vision [online]. [Praha: ČVUT], 12. 5. 2014 [cit. 2015-02-17]. Dostupné z: http://cmp.felk.cvut.cz/~pajdla/gvg/GVG-2014-Lecture.pdf.
[3] C. Albl, Z. Kukelova, T. Pajdla: R6P - Rolling Shutter Absolute Pose Problem. CVPR 2015. Boston USA.

Vedoucí diplomové práce: Ing. Tomáš Pajdla, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 17. 2. 2015

Poděkování

Děkuji Ing. Tomášovi Pajdlovi, Ph.D. za pomoc při vedení diplomové práce. Mé poděkováni patří též Ing. Čeňku Alblovi za poskytnutí implementovaných algoritmů, Tomáši Novákovi za technickou pomoc při konstrukci experimentů a Mgr. Lucii Jakubcové za pomoc při gramatické kontrole práce.

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne ....

Podpis autora práce

Abstrakt

This thesis on the possibilities of taking pictures on cell phones with the Android operating system in order to find picture capture timing and shutter parameters needed for 3D reconstruction. Three applications for camera control were created. This work contains a description of how these applications were implemented. Camera parameters, namely exposure and shutter timing, are obtained using several techniques, which are compared for accuracy and susceptibility to error. The work analyses the influence of these parameters on 3D reconstruction. The result of conducted research is a description of the possibilities of controlling a camera on a cell phone with the Android operating system and a summary of information needed for 3D reconstruction on devices with an electronic rolling shutter.

Abstrakt

Tato práce řeší možnosti pořízení fotografií na mobilním telefonu s operačním systémem Android za účelem zjištění časování pořízení snímků a parametrů závěrky pro 3D rekonstrukci. Byly vytvořeny tři aplikace k ovládání kamery. V této práci jsou popsány návody, jak byly tyto aplikace vytvořeny. Parametry kamery, konkrétně doba expozice a čas vyčítání jednoho řádku fotografie, jsou určeny několika metodami, u nichž je porovnána přesnost a náchylnost na chyby. Práce analyzuje vliv těchto parametrů na 3D rekonstrukci. Výsledkem provedeného výzkumu je popis možností ovládání fotoaparátu mobilního telefonu s operačním systémem Android a souhrn informací potřebných pro trojdimenzionální rekonstrukci na zařízení s elektronickou štěrbinovou uzávěrkou.

Obsah

Seznam zkratek, konstant a symbolů

3A označení pro expozici, ohniskovou vzdálenost a vyvážení bíle barvy
ADT *"Android Development Tools"* - doplněk do prostředí Eclipse pro vývoj aplikací na operační systém Android
AE *"Auto Exposure"* - automatická doba expozice
AF *"Auto Focus"* - automatické nastavení ohniskové vzdálenosti
AWB *"Auto White Balance"* - automatické vyvážení bílé barvy
CCD *"Charge Coupled Device"* - zařízení s vázanými náboji, druhý nejrozšířenější čip pro snímání obrázků
CMOS *"Complementary Metal Oxide Semiconductor"* - komplementární kov-oxid-polovodič, nejrozšířenější čip pro snímání obrázků
C-API, **C-API2** Android kamera API, API2
HAL *"Hardware Abstraction Layer"* - hardwarová vrstva pro ovládání kamery v C++ pod vrstvou os. Android
JNI *"Java Native Interface"* - rozhraní jazyku Java pro spouštění nativních C/C++ kódů
K kalibrační matice kamery
LD *"Line Delay"* - doba vyčítání jedné řádky snímku
RANSAC *"RAndom Sample Consensus"* - iterativní metoda pro odhad parametrů matematického modelu
RS *"Rolling Shutter"* efekt
SfM *"Structure from Motion"* - rekonstrukce prostředí z pohybu kamery
SLERP *"Spherical Linear intERPolation"* - lineární interpolace rotace

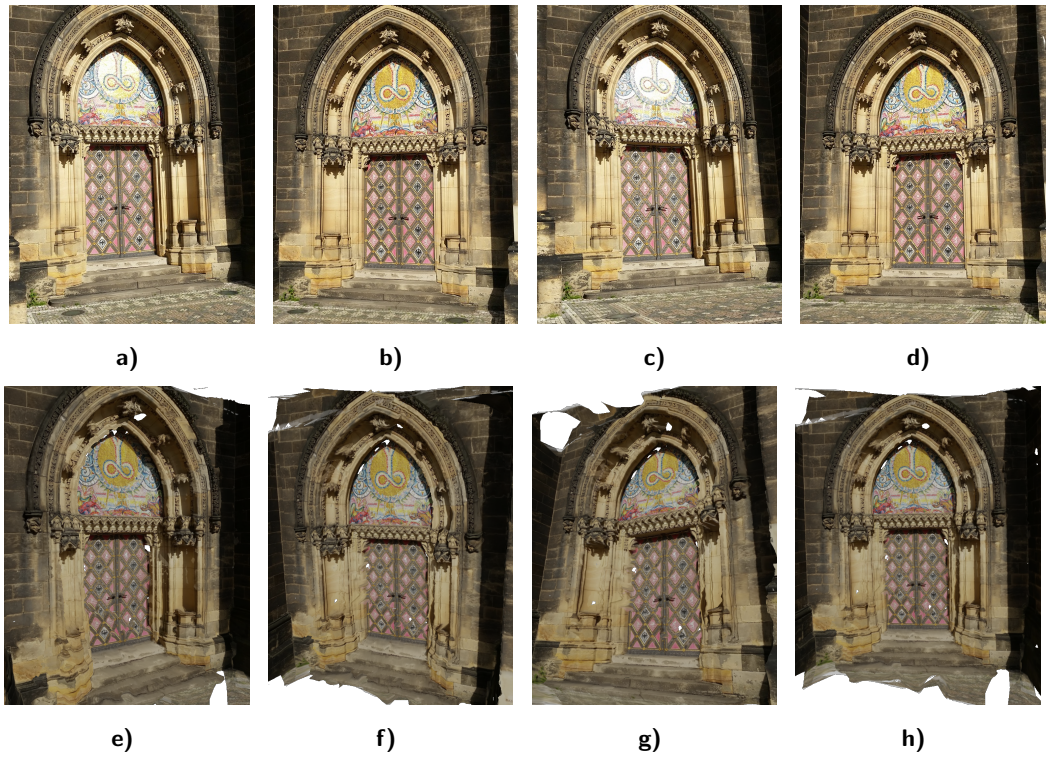
Δt doba snímání jednoho snímku od počátku expozice první řádky po konec vyčítání poslední řádky.
$\Delta\theta$ úhel zkosení čar.
exp čas expozice v sekundách
K kalibrační matice.
ω počet sekund potřebných na jednu otáčku kamery.
R_z rotace kolem osy z
s_c šířka čáry c.
sv konstanta *"ShuterValue"* z EXIF.
τ_ω úhel rotace o jeden pixel při rychlosti otáčení ω.
$t(\theta)$ funkce udávající čas, za který kamera rotovala o úhel θ.
u, v horní, spodní středy čar v souřadné soustavě kamery.
u_1, u_2 horní středy čar v souřadné soustavě snímku.
v_1, v_2 spodní středy čar v souřadné soustavě snímku.

1. Úvod

Diplomová práce se zabývá pořízením fotografií pomocí zařízení s elektronickou štěrbinovou uzávěrkou, konkrétně odhadem perspektivní geometrie při pohybu zařízení. Základním východiskem výzkumu je srovnání metod využívaných pro zjištění časování pořízení snímku a parametrů závěrky kamery, na které navazuje implementace metody využívající při výpočtu pohyb kamery. Jedná se o metodu R6P. Cílem práce je srovnání uvedeného algoritmu s konvenčními metodami obsaženými v knihovně OpenCV. Metody z knihovny OpenCV pracují s předpokladem, že zařízení se během snímání nepohybují. Algoritmus implementovaný v této práci naopak dokáže polohu kamery vypočítat i když byla poloha kamery během snímání změněna. V této práci jsou algoritmy implementovány na zařízení s operačním systémem Android.

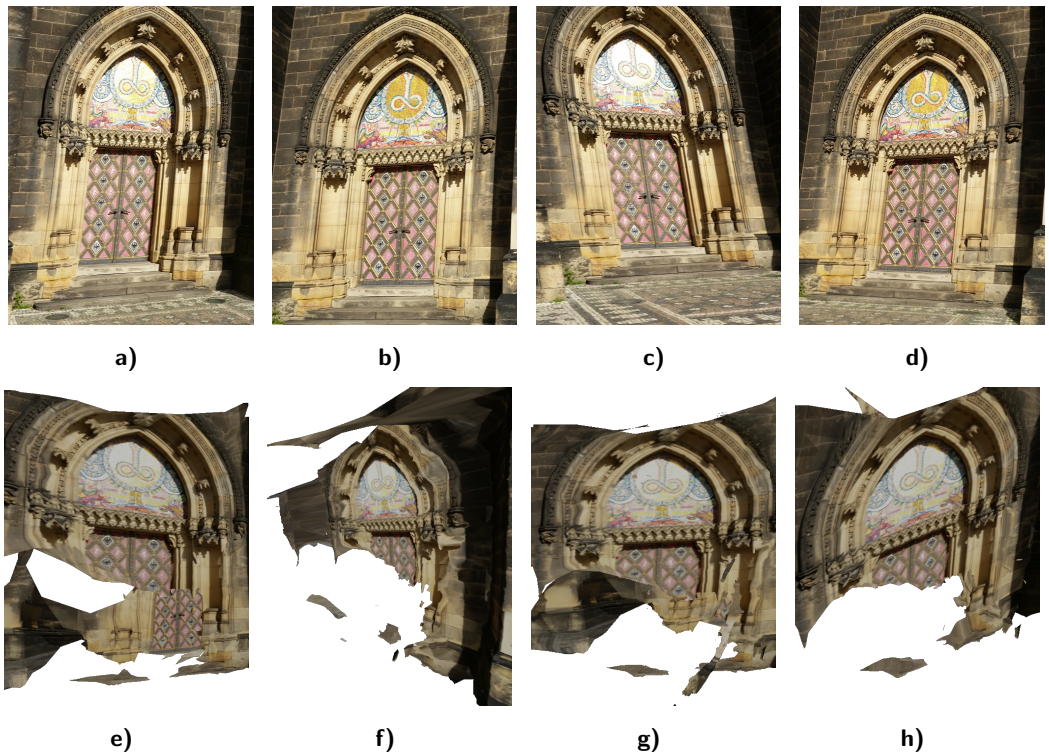
1.1. Motivace

Typickou úlohou počítačového vidění je získání informací o trojdimenzionální scéně pomocí geometrického popisu vztahů mezi scénou a kamerami. Oblast, která se tímto geometrickým popisem zabývá, se nazývá vícepohledová geometrie *"Multiple view geometry"* [10]. Pomocí ní lze řešit řadu úloh, například úlohu rekonstrukce prostředí z pohybu (SfM *"Structure from motion"*) [27] nebo rekonstrukce prostředí ze dvou snímků (*"Stereo vision"*) [22]. Výsledky těchto úloh lze využít například pro 3D rekonstrukci scény nebo k určení pozice kamery (*"Simultaneous localization and mapping"*) [6]. Demonstrace využití uvedených metod je zachycena na o obrázku 1.1. Jedná se o 3D rekonstrukci, která vznikla ze čtyř snímků.



Obrázek 1.1. Rekonstrukce z fotografií, které byly pořízeny bez pohybu kamery během vyčítání snímků. Obrázky **e)**, **f)**, **g)**, **h)** zobrazují rekonstruovanou třídimenzionální scénu z obrázků **a)**, **b)**, **c)**, **d)**. Scéna byla rekonstruována pomocí VizualSfM a CMPMVS

V současné době je většina zařízení využívaných k zachycení snímků vybavena čipem CMOS (komplementární kov-oxid-polovodič, *"Complementary Metal Oxide Semiconductor"*). Mezi hlavní výhody tohoto čipu patří především delší expozice, menší spotřeba energie a levnější výroba než u čipů CCD (zařízení s vázanými náboji *"Charge Coupled Device"*), které jsou v těchto zařízeních druhými nejčastěji využívanými čipy. Nevýhodou je naopak nepatrné zvýšení šumu. Z ekonomických důvodů je ve většině zařízeních s tímto čipem prováděno vyčítání snímků po řádcích, což má v případě pohybujícího se zařízení za následek deformaci snímané scény. Tento jev je nazýván *"Rolling shutter"* efekt. Existují sice některá zařízení s čipem CMOS, která dokáží vyčíst snímek v jeden okamžik. Jejich výroba je však značně nákladná a tudíž nerentabilní [3]. Zmiňovaný *"Rolling shutter"* efekt je patrný i na obrázcích 1.2. V porovnání s



Obrázek 1.2. Rekonstrukce z fotografií, které byly pořízeny při pohybu kamery během vyčítání snímků. Obrázky **e)**, **f)**, **g)**, **h)** zobrazují rekonstruovanou třídimenzionální scénu z obrázků **a)**, **b)**, **c)**, **d)**. Scéna byla rekonstruována pomocí VizualSfM a CMPMVS

rekonstrukcí provedenou z obrázků, při jejichž pořízení nebyla kamera v pohybu (viz obrázek 1.1), je rekonstrukce ze snímků, při nichž se kamera pohybovala, značně deformovaná a nepřesná (viz obrázek 1.2). Výsledná přesnost vypočteného modelu závisí na počtu nalezených korespondencích mezi dvojicemi snímků. Nalezené korespondence slouží k optimalizaci počítané geometrie. Konvenční metody nepředpokládají pohyb a z tohoto důvodu je nezanedbatelná část správných korespondencí při pohybu odstraněna. Abychom byli schopni rekonstruovat 3D prostředí i ze snímků, při nichž se kamera pohybovala, byl vyvinut algoritmus R6P [4], který doposud nebyl implementován a nebylo provedeno jeho srovnání s konvenčními metodami. Tato práce je tedy motivována snahou o implementaci a ozkoušení funkčnosti tohoto algoritmu.

1.2. Přínosy práce

Přínosy této práce jsou,

- popis deformací vzniklých *Rolling Shutter* (RS) efektem,
- ověření možností vyčítání a popis kamer s čipem CCD a CMOS,
- návod pro pořízení videa na telefonu s operačním systémem Android,
- analýza systémových časů při snímání,
- analýza času vyčítání jedné řádky fotografie (LD),
- popis používaných metod pro určení LD a porovnání jejich přesností,
- vylepšená detekce Aruco značek,
- implementace P3P a R6P algoritmu a porovnání přesností a časů výpočtu s algoritmy P5P, P6P a PnP z kihovny OpenCv,

- porovnání přesnosti reprojekce a časové náročnosti nového algoritmu, který uvažuje RS zkreslení.

Přiložené DVD obsahuje: a) vyvinuté aplikace, b) skripty vyhodnocující experimenty, c) simulace zkoumaných jevů a d) testovací data. Všechny tyto kódy a data jsou popsány v příloze této práce (A. Testovací data, B. Knihovny pro vyhodnocení experimentů).

1.3. Struktura práce

Práce sestává ze čtyř hlavních částí, úvodu (Kap. 1) a závěru (Kap. 7).

První část práce, která zahrnuje dvě kapitoly (Kap. 2; Kap. 3), je teoretickým uvedením do zkoumané problematiky. V kapitole 2 *"Rolling shutter" efekt* je popsán mechanismus snímání za použití čipů CCD a CMOS. Je popsán jejich vývoj a čipy jsou zhodnoceny z hlediska jejich výhod a nevýhod. V této kapitole jsou také popsány druhy zkreslení, k nimž při snímání pomocí těchto čipů dochází. Kapitola 3, *Současná řešení "rolling shutter" zkreslení*, shrnuje současné vědecké poznatky řešení RS efektu a popisuje způsoby modelování, odstranění a využití vlivu RS efektu. Pozornost je podrobněji věnována řešení pomocí algoritmu R6P.

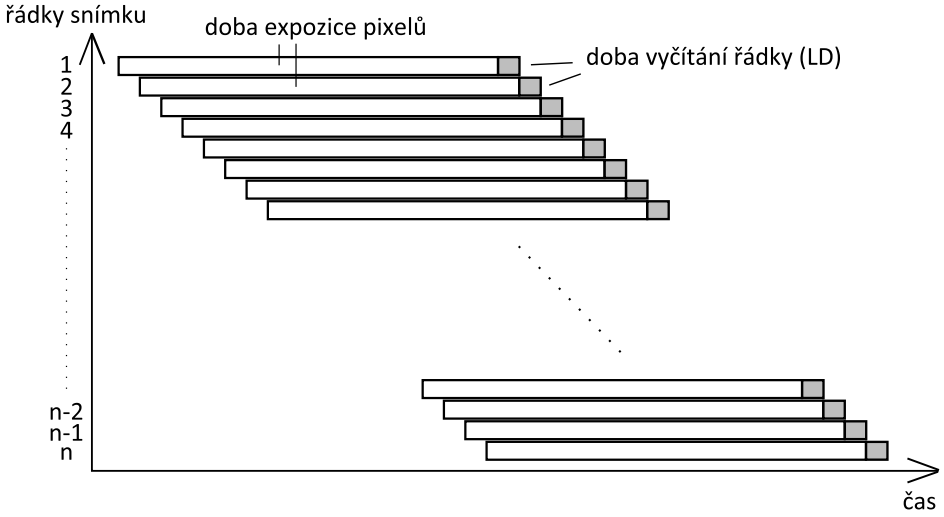
Druhá část práce (Kap. 4) je zaměřena na tvorbu aplikací pro operační systém Android, pomocí nichž je ovládána kamera zařízení. Pojednány jsou tři postupy pro ovládání kamery. První postup je zaměřen na ovládání kamery na Androidu do verze 4.x, druhý na ovládání kamery od Androidu 5.0 a třetí shrnuje jakým způsobem pracovat s ovládáním kamery při využití knihovny OpenCv.

Třetí část (Kap. 5; Kap. 6) je zaměřena na popis samotného provedení a průběhu experimentů a na zhodnocení výsledků měření.

Čtvrtou část tvoří dodatek, ve kterém je popis všech přiložených kódů a návod k jejich použití. Konkrétně se jedná o tři aplikace na operační systém Android a kódy k vyhodnocení všech experimentů.

2. "Rolling shutter" efekt

Při postupném vyčítání řádek během pohybu kamery vzniká významná deformace obrazových dat, která je vidět na obrázcích 1.2. Každý řádek se začíná exponovat v jiný čas. Schéma 2.1 znázorňuje průběh snímání jedné fotografie.

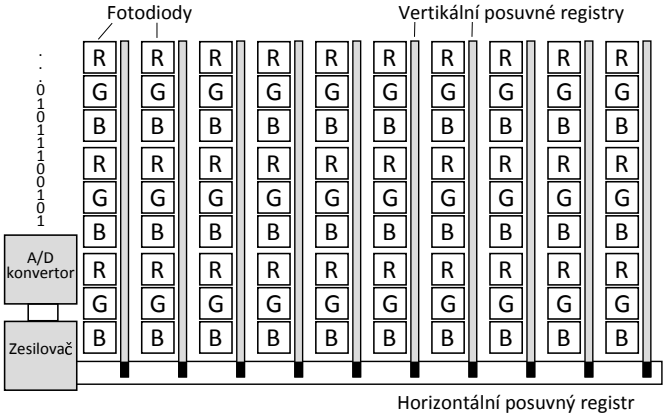


Obrázek 2.1. Schéma nejčastějšího způsobu vyčítání obrázku z CMOS čipu. Osa y odpovídá řádkům obrázku a na ose x je zobrazen čas.

Čip CMOS obsahuje složitější struktury, než konkurenční čip CCD. Výroba čipu CMOS je levnější především kvůli použití stejné technologie, jako při výrobě počítačových čipů. Konkurenční čip CCD byl vynalezen už v roce 1969 (Boyle a Smith) a na trhu převládal přibližně do roku 2011 a to především díky větší světlocitlivé ploše a znatelnému odstupu signálu od šumu. Roku 2009 firma Sony uvedla na trh první BSI CMOS čip, který měl kovové části posunuty pod fotocitlivou vrstvu. Tato technologie výrazně zvětšila plochu absorbující fotony a společně s využitím miniaturních čoček pro lepší směrování světla odstranila nedostatky CMOS čipů. V současné době se CCD čipy používají pouze v levných zařízeních s nižším rozlišením nebo u průmyslových kamer, protože mají lepší odstup signálu od šumu.

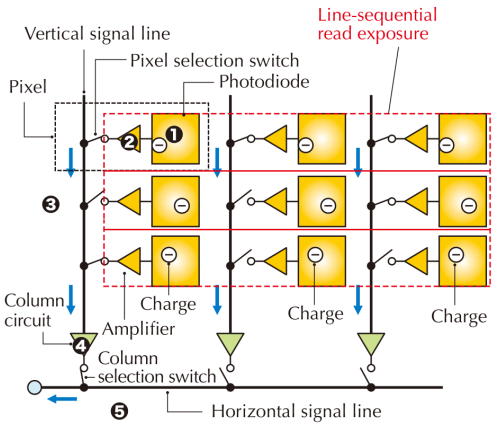
2.1. Architektury kamer

Základním principem fungování čipu CCD je, posun elektronů od fotodiod k zesilovači a analog-digitálnímu převodníku střídáním třech různě velkých napětí. Vzniklý posuvný registr musí mít minimální ztrátovost, kvůli několikanásobnému posunu uvolněného náboje před jeho zesílením. Střídání různých napětí má za následek řádově vyšší spotřebu, než které dosahují čipy CMOS. Během posouvání náboje nesmí fotocitlivé části absorbovat další fotony. Tento problém je řešen elektronickou závěrkou, která je jedním z důvodů proč je výroba čipů tohoto druhu dražší. Zjednodušené schéma CCD senzoru je na obrázku 2.2. Čip CCD exponuje všechny fotocitlivé části najednou, díky



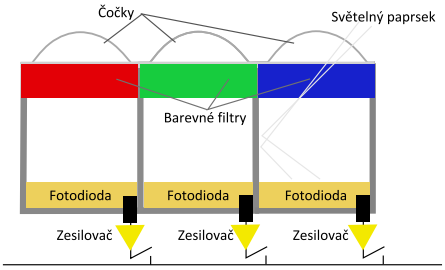
Obrázek 2.2. Schéma senzoru CCD

čemuž lze geometrii promítání modelovat perspektivní projekcí. Každý snímek je pořízen z jedné pozice. Čip CMOS má oproti čipu CCD převaděč náboje na napětí u každé fotodiody [3]. Každý pixel má u sebe tranzistor, který slouží k řízení stavu vyčítání a tranzistor sloužící k odstranění náboje [25]. Před expozicí je náboj z fotodiod vybit pomocí resetovacího tranzistoru. Po dokončení expozice je spuštěno vyčítání přivedením řídící napětí na vyčítací tranzistor. K přesunu napětí dochází pomocí vertikální sběrnice, na jejímž konci je umístěn analog-digitální převodník (viz obrázek 2.3). Každý sloupec má přiřazen jeden analog-digitální převodník, který pro odstranění šumu převede napětí na digitální signál dvakrát. Průměrem těchto dvou hodnot, je výsledná intenzita jasu, která je přenášena pomocí horizontální sběrnice. Vyčtení celého obrázku najednou vyžaduje dodatečnou paměť u každého pixelu. U testovaného Samsungu Galaxy S5 je obrázek vyčítán po řádcích, díky čemu není dodatečná paměť potřeba. Zjednodušené schéma vyčítání snímku po řádcích je na obrázku 2.1.



Obrázek 2.3. Obrázek zobrazuje detail CMOS senzoru z webových stránek CCTV System.

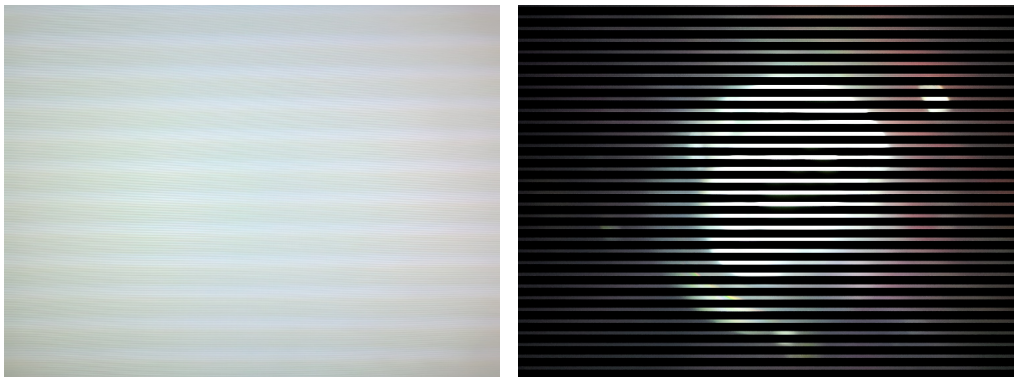
Na testovaném mobilním telefonu Samsung Galaxy S5 je čip S5K2P8, typu *CMOS ISOCELL*.Technologie *ISOCELL* spočívá v oddělení jednotlivých pixelů pomocí bariéry, která odráží světlo, viz obrázek 2.4. Výsledkem je, vyšší kvantová efektivita.



Obrázek 2.4. Na obrázku je znázorněna technologie CMOS ISOCELL, od výrobce Samsung.

2.2. Druhy zkreslení

Vlivem snímání vzniká několik druhů deformací. Cílem experimentů v části 5 je pořídit fotografie některých z nich. Následně pomocí modelů a změn ve snímaných scénách ověřit, jak probíhá časování vyčítání snímků pod operačním systémem Android. Všechny druhy deformací nastávají při změnách snímané scény. Deformace lze rozčlenit na ty, jež vznikly změnou osvětlení a na ty, které byly způsobeny změnou pozice snímaných objektů nebo kamery. Při změně osvětlení během vyčítání snímku je na výstupním obrázku pozorovatelná skoková změna intenzity jasu. Při dostatečně rychlé a intenzivní periodické změně vznikly obrázky 2.5 na kterých je tento efekt zachycen.



a) **b)**

Obrázek 2.5. Na obrázku **a)** je IPS monitor. Obrázek **b)** byl pořízen pohledem do baterky s pulzně šířkovou modulací. Na obou obrázcích je vidět střídání jasů, které je způsobeno postupným vyčítáním snímku při periodickém rozsvěcení a zhasínáním světelného zdroje.

Efekt, který je známý v anglické terminologii, jako *„jello effect"* a je způsoben vibracemi kamery. Nejlépe je pozorovatelný na videu, nebo sekvenci střídajících se snímků. Obrázek 2.6 vznikl složením dvou následující obrázků ze sekvenčního snímání, při vibracích kamery. Na přiloženém DVD jsou ve složce *"Fotografie práce"* přiloženy jejich originály (tres1.jpg a tres2.jpg). Při posunu mezi nimi, pomocí prohlížeče je zřetelná nepřirozenost obrazu a jeho deformace.

Obrázek 2.6. Snímek je složen ze dvou následujících, při vibraci pořízených snímků.

Posledním, a zároveň nejčastějším druhem deformací, je přímá změna polohy objektu ve scéně nebo pozice kamery. Příklady jsou vidět na obrázcích 1.2 a 2.7. Tohoto efektu se dá využít. Ait-aider a kol. používají v práci [1] tyto deformace k určení rychlosti pohybu známých objektů ve scéně z jednoho snímku. Výrazně častější je ovšem snaha tento efekt odstranit, viz kapitola 3. Ukázkový obrázek 2.7 vznikl střídáním rotace kamery a nehybného stavu, během vyčítání jednoho obrázku.

Obrázek 2.7. Fotografie byla pořízena při střídání pohybu, konkrétně rotace kamery a neměnnou pozicí

3. Známé přístupy k odstranění "Rolling shutter" efektu

V současné době existuje několik přístupů jak naložit se zkreslením, které vzniká postupným vyčítáním řádků. Tyto přístupy se dělí podle výstupu, kterého chtějí autoři dosáhnout. Výstupem může být například upravené video bez zkreslení, fotografie bez rozmazání, určení rychlosti objektů nebo řídká a hustá rekonstrukce bodů ve trojdimenzionálním prostoru. Další roli hraje časová náročnost a přesnost. Aplikace kalibrující čas vyčítání řádky [19] musejí být co nejpřesnější a aplikace běžící na mobilním telefonu (například [17] [12] [13]) musejí zvládat zpracování snímků v reálném čase. Autoři prací [15] [1] [13] [5] [7] [20] [19] vycházejí z předpokladu, že perspektivní geometrie platí pro jednotlivé řádky. Cílem těchto prací je modelovat pohyb a rotaci kamery. Následně tyto informace použít pro trojdimenzionální rekonstrukci a rektifikaci obrázků ve videu. Druhý přístup se zaměřuje přímo na rektifikaci videa, bez nutnosti rekonstrukce prostředí. Například Grundmann a kol. [8] mapuje změny mezi dvěma obrázky homografiemi, Baker a kol. [2] na základě optického toku, Karpenko a kol. [12] pomocí gyroskopů, Saurer a kol. [21] transformací pixelů mezi následujícími snímky a Šindelář a kol. [17] pomocí rozdělení snímku na části.

3.1. Vývoj současných řešení

Zkreslení vznikající rolling shutter efektem je problém, který se začal objevovat ve velké míře od uvedení technologie BSI CMOS roku 2009. Geyer a kol. ve své práci [15] otevírají tento problém už roku 2005. Popisují obecné rovnice pro RS kameru a jak jsou ovlivněny různými pohyby kamery. Ukazují, jak modelovat RS kameru několika GS kamerami při lineárním pohybu, který je paralelní se snímanou rovinou a jak odhadnout projekci při konstantní rotaci kolem jedné z os. Při modelování RS kamery je potřeba přesně určit čas vyčítání jedné řádky. Geyer ukazuje, že lze pro tuto kalibraci použít intenzivní světelný zdroj blikající přímo do senzoru. Následnou FFT analýzou [26] je určena frekvence střídání jasů ve snímku. Ukázka tohoto efektu je na obrázku 2.5. O rok později Ait-Aider a kol. [1] popisují, jak využít RS pro dohad rychlosti známého 3D objektu pomocí šesti korespondencí nelineární metodou minimálních čtverců a planárního objektu pomocí devíti korespondencí lineární metodou. Na nelineární metodu může být pohlíženo jako na optimalizaci reprojekční chyby (*"bundle adjustment"*). Rotace je v této práci vyjadřována pomocí quaternionu uvedeného v [23]. V roce 2009 Klein a Murray [13] optimalizovali SLAM pro použití na mobilních zařízeních pomocí paralelního sledování bodů a výpočtu pozice zařízení. Pro redukci RS efektu aproximovali lineárně pohyb mezi následujícími snímky a upravili pozice bodů před jejich optimalizací. Klein a Murray na rozdíl Karpenko a kol. neodstraňují rotaci, která hraje při RS efektu zásadní roli. Karpenko a kol. [12] v roce 2011 využili ke stabilizaci videa gyroskopy. Jedná se o jednu z prvních prací, kdy jsou snímky upraveny bez přímého výpočtu epipolární geometrie. Před prací od Karpenka a kol. je zde uveden pouze Baker a kol. [2], který v roce 2010 odhadoval velký pohyb mezi snímky, které jsou ve velkém rozlišení a přicházejí v dlouhých časových intervalech na základě afinního modelu po-

hybu. Tento přístup umožňuje redukci více pohybujících se objektů ve scéně, ovšem přesnost odstranění RS efektu byla záhy překonána. Karpenko sjednocuje rotaci a RS tím, že pomocí údajů z gyroskopu definuje funkci obalující dva snímky. Tato funkce transformuje bod z jednoho snímku do druhého a jejím vypočtením pro skupiny řádek je vytvořena syntetická kamera, která má plynulý pohyb a skládá se z množiny rotací GS kamer. Odstranění RS efektu je popsáno pouze pro rotace a posun není uvažován. GS kamery jsou vytvořeny pomocí interpolace quaternionu (SLERP [23]). Dále v práci definuje, jak kalibrovat rotaci získanou z gyroskopů. Kalibrace vůči snímkům probíhá minimalizací chyby reprojekce pomocí metody nejmenších čtverců. Další řešení, které počítá s obrazovými daty bez výpočtu pozice kamery, je řešení Grundmanna a kol. [8]. Grundmann popisuje vztah mezi následujícími snímky pomocí homografií pro několik planárních objektů. Následnou QR dekompozicí a optimalizací transformace se minimalizují chyby vzniklé RS efektem. Tato metoda umožňuje úpravu několika pohybujících se objektů ve scéně. Téhož roku vyšel článek od Jia a kol. [5], který se zaměřil na pravděpodobnostní odhad rotace kamery za pomoci vyžití gyroskopů, Kalmanova filtru, úběžníků a dalších vizuálních interních měření. Jako pravděpodobnostní model je použita dynamická Bayesovská síť. Jia odhaduje RS rotaci, kterou následně interpoluje a použije pro odstranění zkreslení. Na spojitý popis pozice kamery se zaměřuje článek od Furgale a kol. [7]. Je v něm popsán pohyb pomocí množiny funkcí $\Phi = [\Phi_1, \Phi_2, ..., \Phi_n]$, kterým jsou přiřazovány váhy $\mathbf{c} = [c_1, c_2, ..., c_n]$. Těmito funkcemi je prokládán pohyb kamery na ohraničených časových úsecích a díky tomu není potřeba optimalizovat pohyb kamer pro celé video najednou. Tomuto prokládání se říká *B-spline*. Tato reprezentace je využita i v dalším článku Oth a kol. [19] z roku 2013. Oth řeší problém přesného stanovení LD bez osciloskopu, blikajícího světelného zdroje a dat z gyroskopů. Pro zjištění LD je využita kalibrační mřížka. Rotace a pohyb jsou aproximovány v čase nejen mezi snímky videa, ale i během jednoho snímku. Algoritmus je inicializován jako GS kamera s vyjádřením chyby v závislosti na nepřesnosti reprojekce. Pokud je chyba větší než stanovený práh, je oblast s chybou (konkrétní snímek) rozdělena na části. Takto se oblasti dělí a dokud není chyba dostatečně malá. Pohyb i rotace mezi částmi jsou aproximovány opět váženou množinou funkcí. V roce 2012 se objevila rozsáhlá dizertační práce Ringabiho a kol. [20], která se zaměřuje na aproximaci libovolné rotace i pohybu kamery z videa. Saurer a kol. [21] v roce 2013 upravili *Stereo vision* pro RS kamery. Hustá rekonstrukce je popsaná pro planární scény a efekt vzniklý RS je odstraněn pomocí mapování deformací RS množinou funkcí. Pro opravu je použita obalující funkce mezi dvěma sousedními obrázky videa. Jeden z posledních přístupů, kterým odstranili autoři Šindelář a kol. [17] RS efekt, je rozdělení snímku na oblasti a práce s každou oblastí zvlášť. Jejich práce se zaměřuje na opravu fotek při pohybu. Odstraňují rozmazání pomocí dekonvoluce s Wienerovým filtrem, který vytvoří pomocí sledování pohybu gyroskopů. Obraz je rozdělen na 48 částí s 25% překryvem a transformován pomocí FFT. Následně je dekonvolucí odstraněno rozmazání a snímek je transformován zpět, kde se jednotlivé části obrázku skládají pomocí přiřazení vah Hammingovými okny.

3.2. R6P algoritmus

Algoritmus R6P [4] funguje na scénách, které neleží v jedné rovině a RS efekt kompenzuje v rámci jednoho snímku. Vychází z upravené projekční funkce [10] $\lambda_i x_i = R(r_i)X_i + C(r_i)$, kde $R(r_i)$ je funkce rotační matice a $C(r_i)$ funkce vektoru posunu od středu souřadné soustavy. Bod X_i je v globální souřadné soustavě a promítá se na vek-

tor v souřadnicích kamery definovaný trojicí $x_i = [r_i, c_i, 1]^T$ a jejími násobky skalárem λ_i. Funkce rotace a posunu je definovaná v závislosti na řádku r_i. Pro zjednodušení je rotace vyjádřená pomocí quaternionu a následný výpočet je bez použití goniometrických funkcí. Quaternion lze transformovat, až na rotaci o 180^o, do rotační matice polynomiálním zápisem nazývaným *Cayleyova transformace* [11]. Rotace se dá rozdělit podle jednotlivých os a $R(r_i)$ se dá přepsat na rotaci kolem osy x, z a rotaci $R(v)$ násobenou rotací kolem osy y jako $R(r_i, w)$. Princip polynomiálního řešení spočívá ve využití Gröbnerových bází. Jelikož je řešení soustavy polynomů pátého stupně o 18 proměnných časově náročné a nestabilní, je rotace linearizována pomocí Tylorova rozvoje podle iniciální rotace $R(v)$.

$$\lambda_i \begin{bmatrix} r_i \\ c_i \\ 1 \end{bmatrix} = (I + (r_i - r_0)[w]_x)R(v)X_i + C + (r_i - r_0)t \tag{3.1}$$

Kde t je posun v globálních souřadnicích vlivem RS a $[w]_x$ je antisymetrická matice tvořená trojicí $[w_1, w_2, w_3]^T$, která reprezentuje RS rotaci. Ani tento model není dostatečně zjednodušený pro běh v *RANSAC* algoritmu a tak je linearizovaná ještě inicializační rotace $R(v)$ nahrazením $I - [v]_x$. Iniciální hodnoty matice $[v]_x$ jsou získány pomocí P3P algoritmu [9]. Minimální počet korespondencí pro řešení pomocí Gröbnerových bází je 6. Další zjednodušení vzniká eliminací λ_i pomocí vynásobení antisymetrickou maticí z trojice $[r_i, c_i, 1]^T$. Řešením této soustavy 12 lineárně nezávislých rovnic je vypočteno až 20 řešení $\mathbf{v}$ a $\mathbf{w}$. Z těch je dosazením do 3.1 získáno řešení $\mathbf{C}$ a $\mathbf{t}$. Průběh vyhodnocení je generován automatickým generátorem minimálních řešení [14].

4. Tvorba aplikace

Pro účely zkoumání vlivu *Rolling Sutter* efektu na mobilním telefonu s operačním systémem Android byly vytvořeny 3 aplikace. První aplikace obstarává ukládání fotografií různými způsoby za pomoci rozhraní s názvem *Camera API* (C-API). Slouží k porovnání délek snímání náhledu a fotografií. Dále ukládá doplňující údaje jako např. data ze senzorů. Druhá aplikace také snímá a ukládá fotografie i data ze senzorů. Umožňuje navíc výběr rozlišení a potřebná nastavení, např. zapínání macro módu přes uživatelské rozhraní. Je napsaná pomocí novějšího rozhraní *Camera API2* (C-API2). Poslední, třetí aplikace využívá pro komunikaci s kamerou OpenCV knihovnu a zaměřuje se na zpracování snímků a dat ze senzorů, vylepšuje detekci ARUCO značek (jak po časové stránce, tak i přidáním dodatečných funkcí pro získání vnitřních bodů z libovolné značky), umožňuje ukládání všech důležitých údajů dostupných při průběhu zpracování snímků. Přes grafické uživatelské rozhraní lze nastavit ukládání senzorů v požadované frekvenci, ukládání informací o pozicích kamery, vypočtených geometriích, pozicích referenčních bodů ve snímku, v prostoru a jejich reprojekčních chybách.

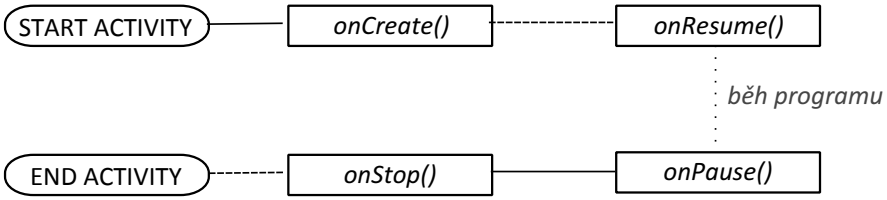
4.1. Základy tvorby aplikací pro Android

Struktura projektu se v jednotlivých vývojových prostředích a jejich verzích mění, ale důležité složky mají pořád stejný název. Názvy těchto složek jsou shrnuty v následujícím výčtu:

- **bin** obsahuje zkompilovaný projekt a spustitelnou aplikaci s koncovkou *.apk*,
- **src** obsahuje zdrojové balíčky a kódy aplikace,
- **res** obsahuje několik podsložek se zdroji,
  - *drawable* obsahuje uložené ikony,
  - *values/strings.xml* obsahuje popisky v různých jazycích definovaných příponou,
  - *menu* obsahuje definice položek menu jednotlivých obrazovek aplikace,
  - *layout* obsahuje grafické rozvržení jednotlivých obrazovek aplikace,
- **libs** obsahuje Java knihovny, nebo předkompilované C/C++ kódy,
- **jni** obsahuje nastavení pro kompilaci nativních kódů a samotné C/C++ soubory.

V projektu jsou vždy první tři složky, zbylé dvě je potřeba vytvořit při využívání knihoven a Java Native Interface (JNI). Důležitým souborem v projektu je *Mainfest.xml*, který shrnuje základní nastavení, registrované aktivity a práva aplikace. Je v něm například uvedeno, jaká aktivita se má zobrazovat v menu. K ovládání běhu aplikace existuje několik variant. Nejjednodušší je podědění objektu *Activity* a implementace vyžadovaných metod. Zjednodušený diagram popisující hlavní stavy objektu *Activity* pomocí metod vyvolaných při tomto stavu je na obrázku 4.1. Metoda *onCreate()* je volána při vytváření objektu, *onResume()* těsně před předáním řízení programu a zobrazení uživatelského prostředí (UI), například při změně orientace zařízení, *onPause()*

je první volání při předání řízení jinému potomku *Activity*, což je například ukončení aplikace a *onStop()* když UI přestává být viditelné. Objekt není ihned smazán, ale ponechán v paměti pro rychlý start. Informace o mazání je předána metodě *onDestroy()*, která společně s metodami *onRestart()* a *onStart()* není pro tvorbu aplikace spravující kameru důležitá.



Obrázek 4.1. Diagram stavů objektu *Activity*

Při tvorbě je potřeba nezapomenout registrovat aktivity v souboru *Mainfest.xml*. Neregistrované aktivity nepůjdou spustit. Registrace se provádí přidáním elementu *<activity>* s atributem *android:name* a *android:label*. Atribut *name* obsahuje ve zkráceném zápise pouze tečku a název potomka *Activity*, například *.OvladaniKamery*. Atribut *label* slouží k zadání popisku zobrazeného v menu a horní stavové liště aplikace s využitím jazykových mutací například *@string/id_zobrazeneho_popisku*. Typický postup při vytváření potomka *Activity* je v metodě *onCreate()* uložit odkaz na předcházející stav aplikace voláním *super.onCreate(savedInstanceState)*, přičemž objekt *savedInstanceState* je vstupním parametrem metody *onCreate()* a inicializovat objekty UI. Inicializace UI je nejjednodušší pomocí načtení předpřipravené grafiky uložené v XML dokumentu *res/layout/nazev.xml* voláním *setContentView(R.layout.nazev)*. XML dokument s grafikou lze vytvořit a editovat pomocí vývojového prostředí *Android Studio* nebo *Eclipse*. Elementy v tomto dokumentu tvoří objekty a jejich atributy vlastnosti, které lze obdobně nastavit i v kódu za běhu aplikace. Přístup ke kameře a její nastavení je vhodné provádět v metodě *onResume()*, která je na rozdíl od funkce *onCreate()* volána při každém zobrazení UI. Ukázka struktury potomka *Activity* je na následujícím kódu.

```
1  // inicializace potomka aktivity
2  public class CameraActivity extends Activity{
3
4          // funkce vyvolana pri startu aktivity
5          protected void onCreate(Bundle savedInstanceState) {
6                  super.onCreate(savedInstanceState);
7                  // načtení připraveného grafického rozhraní "res/layout/main.xml"
8                  setContentView(R.layout.main);
9                  ...
10         }
11
12         // funkce vyvolaná těsně před zobrazením UI
13         public void onResume() {
14                 super.onResume();
15                 // načtení knihovny OpenCV / kamery
16                 ...
17         }
18
19         // funkce vyvolaná těsně před ukončením aktivity
20         public void onPause(){
21               super.onPause();
22                 // odhlášení kamery
23                 ...
24         }
25           ...
26 }
```

4.2. Android do verze 4.x

Na těchto verzích Androidu se používá C-API, které funguje jako black-box a umožňuje nastavit parametry pro kameru. Neumožňuje nastavení pro jednotlivé snímky. Stručný návod k dosažení náhledu a pořízení fotografií je shrnut v 4.2. Na telefonech s Androidem 5.0 a vyšším je využíváno nové C-API2, které vylepšuje nedostatky první verze a umožňuje lepší kontrolu nad snímáním, proto je v části 4.3 rozebráno podrobněji.

Postup práce s kamerou na Androidu do verze 4.x

1. Inicializace. Získání objektu, který reprezentuje a spravuje kameru voláním funkce *Camera.open(id_kamery)*.
2. Nastavení. Pomocí objektu s parametry lze nastavit vlastnosti kamery jako například mód ostření, expozice a clony. Objekt *Camera.Parameters* obsahuje, při jeho vyvolání funkcí *objKamery.getParameters()*, standardní nastavení. Po úpravách se předá zpět kameře voláním *objKamery.setParameters(parametryKamery)*.
3. Vytvoření náhledu. Nedílnou součástí snímání fotografií je v C-API vytvoření náhledu. Náhled vznikne rozšířením třídy *SurfaceView*, *TextureView* nebo jejich potomků a předáním náhledu kameře *objKamery.setPreviewDisplay(objSurfaceView.getHolder())*. Náhled se musí následně spustit voláním *objKamery.startPreview()*.
4. Snímání snímků. Během zobrazování náhledu lze vyvolat snímání fotografie pomocí funkce *objKamery.takePicture(shutter,raw,postview,jpeg)*. Parametr *shutter* je posluchač dokončení snímání. Ostatní parametry jsou posluchače výstupních snímků v různých formátech.
5. Ukončení. Po dokončení práce s kamerou je náhled ukončen voláním *objKamery.stopPreview()* a kamera uvolněna funkcí *objKamery.release()*.

Objekt *shutter* ve čtvrtém bodě postupu je typu *Camera.ShutterCallback*. Tento posluchač je vyvolán podle dokumentace ihned po sejmutí snímku. Na začátku metody *onShutter()*, kterou tento posluchač musí implementovat, lze získávat informace o konci snímání. První aplikace implementuje výše popsaný postup pro získání snímků z kamery a doplňuje ho o získávání dat ze senzorů a různé varianty náhledu. Například náhled pomocí *TextureView*, který je otevřen v hardwarově akcelerovaném okně, s možností specifického vykreslování. *TextureView* umožňuje například průhlednost, kterou standardní okna neumožňují. Tyto části lze přepínat v kódu, odkomentováním a zakomentováním jednotlivých pasáží. Další částí aplikace je snímání dat ze senzorů, které je pro přehlednost zapsáno formou ukázky kódu. Následující příklad ukazuje jak získat vektor dat z akcelerometru.

```
1  // inicializace systémové služby spravující senzory
2  SensorManager sm = (SensorManager) getSystemService(SENSOR_SERVICE);
3
4  // inicializace snímaného senzoru
5  Senzor senzor = sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
6  Akcelerometr objOdchytavajiciUdalosti = new Akcelerometr();
7  ...
8
9  // registrace a odhlášení posluchače, na který bude ...
10 // ... posílán stav senzoru každých 1000ns
11 mSensorManager.registerListener(objOdchytavajiciUdalosti,
12         senzor, 1000);
```

```
13  mSensorManager.unregisterListener(objOdchytavajiciUdalosti);
14  ...
15
16  // třída přijímající data ze senzoru, jako události...
17  // ...v proměnné event
18  class Akcelerometr implements SensorEventListener {
19          @Override
20      public void onSensorChanged(SensorEvent event) {
21          // zpracování dat senzoru
22          ...
23      }
24
25      @Override
26      public void onAccuracyChanged(Sensor sensor, int i) {}
27  }
```

Data ze senzoru jsou dostupná na řádce 20 v objektu *event*, proměnné *values*. Pro snímání ostatních senzorů stačí nahradit konstantu *Sensor.TYPE_ACCELEROMETER* na řádku 5 jiným typem senzoru. V aplikaci je kostra senzoru napsána jako abstraktní třída a její potomci pouze definují typ senzoru. Aplikace snímá akcelerometr, rotaci, gravitaci a gyroskop. Využití senzorů je popsáno v sekci 6.1.

4.3. Android od verze 5

Zdrojový kód Androidu 5 byl uvolněn 3. listopadu roku 2014 a v současné době ho obsahují pouze prémiové telefony. Jením z vlastností tohoto operačního systému je nové rozhraní pro komunikaci s kamerou C-API2. Toto rozhraní je podmíněno implementací *Hardware Abstraction Layer* (HAL) třetí verze, kterou dodává výrobce. U telefonu Samsung Galaxy S5 jsem při testování objevil několik chyb v této implementaci, jejíž dokumentace je na oficiálních webových stránkách [24]. Při vytváření aplikace pracující s kamerou je potřeba dát pozor na obrácení významů *CONTROL_AF_MODE_OFF* a *CONTROL_AF_MODE_AUTO*, kdy automatické ostření zajišťuje mód s koncovkou *OFF* a manuální ostření mód s koncovkou *AUTO*. Bohužel manuální ostření není podporováno. Další chyba spočívá v tom, že by měla být při vypnutém ostření ohnisková vzdálenost nastavena na nekonečno. Bohužel je podle ostrosti snímků mezi 30 - 40 cm před senzorem.

4.3.1. Rozhraní C-API2

Hlavní změnou oproti C-API je možnost ovládat nastavení, například ohniskovou vzdálenost, nebo dobu expozice u jednotlivých snímků. Díky tomuto nastavení lze manuálně vyfotit množinu fotografií pro následné složení HRD snímku, nebo složit několik snímků s různou ohniskovou vzdáleností a dosáhnout tak ostrosti požadovaném rozmezí. Toho se využívá u snímků s malou ohniskovou vzdáleností. Popsané nastavení není bohužel podporováno u všech telefonů a záleží na výrobci, do jaké úrovně zpracuje HAL3 a jaké vlastnosti podporuje samotná kamera.

Postup práce s kamerou na Androidu 5 a vyšším

1. Inicializace. Instance objektu, který spravuje kamery, je dostupná stejným způsobem jako instance objektu *SenzorManager*, který je popsaný v předchozí části. Inicializace se provádí voláním funkce *(CameraManager) getSystemService(CAMERA_SERVICE)*. Objekt *CameraManager* na rozdíl od C-API poskytuje informace o kameře a jejích možných nastaveních, aniž by při tom blokoval kameru. Samotné otevření kamery probíhá voláním metody *objCameraManager.openCamera(id_kamery, callback, handler)*. Parametr *callback* je objekt, který dědí *CameraDevice.StateCallback* a *handler* je je objekt typu *Handler*, který slouží k posílání zpráv do vlákna. Když tyto zprávy implementují interface *Runnable*, je jejich metoda *run* vyhodnocena. V opačném případě se tyto zprávy používají pro mezi vláknovou komunikaci.

2. Správa kamery. V předchozím bodě bylo potřeba předávat objekt, který dědí *CameraDevice.StateCallback*. Tento objekt musí implementovat tři metody, které jsou volány při změně stavu kamery. První metoda se jmenuje *onOpened(kamera)* a předává v parametru objekt kamery *CameraDevice*. Je volána při zpřístupnění kamery aplikaci. V této metodě se inicializují výstupy, na které budou při pořízení snímku přicházet data, což je rozebráno v bodě číslo 3. Další povinná metoda *onDisconnected(kamera)* je volána těsně před tím, než je řízení kamery aplikací ukončeno. Poslední povinnou metodou je *onError(kamera,id_chyby)*, která slouží pro odchycení chyb.

3. Správa výstupů. Po otevření přístupu ke kameře je k dispozici objekt třídy *CameraDevice*. Pomocí jeho metody *createCaptureSession(vystupy, session_callback, handler)* se vytvoří vyrovnávací paměť dopravující výstup z kamery do registrovaných posluchačů. Proměnná *vystupy* je seznam posluchačů *List<Surface>*, které budou přijímat snímky. Tyto objekty vznikají voláním funkce *getSurface()*. Pro základní aplikaci stačí dva výstupy. Jedná se o náhled třídy *SurfaceView* a objekt uchovávající snímky za účelem jejich uložení *ImageReader*. *SurfaceView* se definuje v XML souboru spolu s dalšími elementy UI. V kódu lze získat instanci voláním funkce *this.findViewById(R.id.cameraSurfaceId)* a *Surface*, který se přidá do seznamu *vystupy*, je získán zavoláním metod *objektSurfaceView.getHolder().getSurface()*. Druhý objekt spravující výstupy se inicializuje voláním *ImageReader.newInstance(sirka, vyska, ImageFormat.JPEG, 10)*, ve kterém *sirka* a *vyska* popisuje rozměry obrázku, po kterých následuje formát ve kterém budou výstupní snímky a počet fotografií, které se uchovávají v jeho vyrovnávací paměti. Aby *objImageReader* umožňoval ukládání, je potřeba definovat posluchač *dostupne_foto* třídy *ImageReader.OnImageAvailableListener* voláním jeho metody *setOnImageAvailableListener(dostupne_foto, handler)*.

4. Pořízení snímku. Třída obstarávající vyrovnávací paměť *CameraCaptureSession.StateCallback* musí implementovat dvě metody. První *onConfigured(session)* je volána při bezchybném vytvoření a druhá *onConfigureFailed(session)* pokud nastane chyba. V první metodě je vše správně nakonfigurováno pro pořízení snímku, který se začne snímat po odeslání žádosti o jeho pořízení voláním *capture(request, listener, handler)*. Parametr *request* je typu *CaptureRequest* a obsahuje nastavení při kterém se má snímek pořídit, *listener* je třídy *CameraCaptureSession.CaptureCallback*. Opakovaného snímání se docílí voláním *session.setRepeatingRequest(captureBuilder,listener,handler)* s parametrem třídy *CaptureRequest.Builder*. Ostatní parametry jsou stejné jako dříve popsané proměnné se stejným názvem.

5. Metadata snímku. Objekt s nastavením *CaptureRequest*, který je předán kameře se vrací při startu snímání spolu s číslem pořizovaného snímku do metody onCaptureStarted(session, request, timestamp, poradi_snimku) posluchače *CameraCaptureSession.CaptureCallback*. Po dokončení všech operací souvisejících se snímáním je volána druhá metoda onCaptureCompleted(session, request, result), jejíž parametr *result* třídy *TotalCaptureResult* obsahuje všechny mezivýsledky snímání. Na mobilním telefonu Samsung Galaxy S5 obsahuje v současné době pouze jeden výsledek, který je zároveň uložen v proměnné *request*.

6. Ukládání. Bod číslo 3 končí definicí posluchače *dostupne_foto*. Ten musí implementovat metodu *onImageAvailable(reader)*, kde *reader* je objekt třídy *ImageReader* a vrací snímek (proměnná *foto*) třídy *Image* voláním *reader.acquireNextImage()*. Tento snímek je potřeba ukládat v jiném vlákně, aby neblokovalo UI. V praxi se používá metoda popsaná v bodu číslo jedna. Vytvoří se objekt implementující *Runnable* a odešle se jako zpráva voláním *handler.post(new PotomekRunable(foto,soubor))* do procesoru. Druhý parametr *soubor* je třídy *File* a obsahuje cestu, kam se obrázek uložit.

7. Ukončení. Všechny posluchače spojené s snímáním je potřeba odhlásit v opačném pořadí, než v jakém vznikly - konkrétně *session.stopRepeating()*, pak *session.close()* a na závěr *kamera.close()*. Pokud aktivita končí ve stejnou dobu jako snímání, je vhodné místo pro ohlášení posluchačů funkce *onPause()*.

Možnosti nastavení kamery

Nastavení, ve kterém se mají jednotlivé snímky pořídit je předáváno kameře v objektu *CaptureRequest*. Rozsah možných nastavení je závislý na úrovni, kterou zpřístupní výrobce. Existují tři základní varianty *FULL*, *LIMITED* a *LEGACY*. Na variantu nastavení se lze zeptat dotazem *objCameraCharacteristics.get(id_vlastnosti)*, kde *id_vlastnosti* je *CameraCharacteristics.REQUEST_AVAILABLE_CAPABILITIES*. Samsung Galaxy S5 vrací střední variantu nastavení *LIMITED*, ale při dotazu jaké konkrétní vlastnosti podporuje, stejným dotazem s proměnnou *id_vlastnosti* nastavenou na *CameraCharacteristics.REQUEST_AVAILABLE_CAPABILITIES* vrací pouze konstantu *BACKWARD_COMPATIBLE*, kterou musí podporovat všechny telefony. Tři základní oblasti vlastností, které telefon může podporovat, jsou v následujícím výčtu.

- *BACKWARD_COMPATIBLE* - minimální podpora, kterou musí obsahovat všechny zařízení. Umožňuje snímání náhledu a fotografií, programátor má možnost přepínat mezi automatickým ostřením (AF), expozicí (AE) a vyvážením bíle (AWB). Tato trojice nastavení se nazývá 3A.
- *MANUAL_SENSOR* - podpora manuálního nastavení 3A, ISO, blesku a rychlost snímání
- *MANUAL_POST_PROCESSING* - podpora nastavení úprav po obdržení snímku, například mapování barev, úprava křivek, jasu, kontrastu a stínování

Pokud telefon zprostředkovává úroveň *FULL*, umožňuje všechny výše uvedená nastavení a navíc informuje o mezivýsledcích snímání. V základní podpoře všech telefonů je zapínání a vypínání 3A, které se nastavuje předáním parametru *hodnota* jednotlivým vlastnostem, při volání *CaptureRequest.Builder.set(vlastnost, hodnota)*. Při nejvyšší úrovni podpory parametr *vlastnost* může být jedna z přibližně 20 hodnot a ke každé vlastnosti existují přibližně tři varianty, nebo možnost nastavit číselnou hodnotu. Zapnutí automatického ostření lze nastavit uvedeným voláním s parametry *vlastnost* rovná se *CaptureRequest.CONTROL_AF_MODE* a *hodnota* rovná se *CaptureRequest.CONTROL_AF_MODE_OFF*, což je výjimka Samsungu Galaxy S5 oproti dokumentaci. Nastavení expozice se dělá dosazením *CaptureRequest.CONTROL_AE_MODE* a vyvážení bílé *aptureRequest.CONTROL_AWB_MODE*, obojí do parametru *vlastnost*. Další užitečná nastavení jsou popsána na stránce [16]. Za zmínku ještě stojí vlastnost *CaptureRequest.CONTROL_MODE*, která může nabývat hodnoty s koncovkou *_AUTO* nebo *_OFF*. Tato vlastnost, podle dokumentace, přepisuje dříve nastavené manuální ovládání a spouští v módu s koncovkou *_AUTO* automatické ovládání 3A.

4.4. Využití knihovny OpenCV

Struktura celé aplikace je popsána v dodatku B.1.2. V této části je uvedeno jak zprovoznit knihovny OpenCV a Aruco. Dále jakým způsobem je vylepšena detekce Aruco značek, jak ji spustit a jak spustit výpočet geometrie. Pro implementaci funkční knihovny, obzvláště nativních C/C++ kódů, je potřeba postupovat přesně podle návodu, protože při jakékoliv změně dojde ke zhroucení vývojového prostředí Eclipse. Například, pokud je zdrojový kód C/C++ v tomto editoru otevřen, projekt už nepůjde zkompilovat, nebo pokud je vytvořena složka *jni* jinak než proklikáním se přes vlastnosti projektu, celé prostředí přestane fungovat a projekt opět nepůjde zkompilovat. Bohužel oficiálně doporučené prostředí nepodporuje nativní projekty vůbec a tak je potřeba pracovat v Eclipse s doinstalovaným doplňkem *Android Development Tools* (ADT).

Zprovoznění projektu

1. Import knihoven. Před importem OpenCV knihovny je potřeba vytvořit novou pracovní složku v dialogovém okně *File > Switch Workspace > Other...*. Knihovna se zdrojovými kódy OpenCV se přidá volbou okně *File > Import > Existing Android Code Into Workspace > Browse..* zadáním cesty k *složka_OpenCV/sdk/Java* a následným potvrzením přidávaného projektu. Stejným postupem se přidají knihovny Aruco značek, které lze stáhnout pouze přes *SVN* nebo *GitHub* z této stránky. Ze stažené složky stačí přidat projekty *slozka_Aruco/Aruco* a *slozka_Aruco/min3d-rotation*.

2. Nastavení a oprava knihoven. Prvním krokem je nastavení knihoven použitých v *Aruco* knihovně. V *Properties* projektu *Aruco* je v záložce *Android* po srolování dolů vidět okno s použitými knihovnami. Je potřeba vše odstranit a přidat jako první *OpenCV Library* a jako druhou *min3d-rotation* knihovnu. Dalším krokem je oprava *OpenCV*. V *OpenCV* je chyba v složce *src* balíčku *org.opencv.android* třídě *AsyncServiceHelper* funkci *initOpenCV* jejíž obsah musí být nahrazen následujícím kódem.

```
AsyncServiceHelper helper =
        new AsyncServiceHelper(Version, AppContext, Callback);
Intent intent = new Intent("org.opencv.engine.BIND");
intent.setPackage("org.opencv.engine");
if (AppContext.bindService(intent,
        helper.mServiceConnection,Context.BIND_AUTO_CREATE)){
        return true;
} else {
        AppContext.unbindService(helper.mServiceConnection);
        InstallService(AppContext, Callback);
        return false;
}
```

3. Vytvoření projektu. Vytvoření nového projektu s podporou knihoven a nativní podpory musí být prováděno v následujícím sledu. Projekt lze vytvořit přes menu *File > New > Android Application Project*. Pokud je to možné *Minimum Required SDK* by nemělo být příliš malé, aby se do projektu nepřidávaly knihovny pro zpětnou kompatibilitu. Dalším krokem je přidání nativní podpory pomocí kontextové nabídky projektu *Properties > Android Tools > Add Native Support...* a zadání názvu neexistujícího C/C++ souboru. Eclipse vytvoří v projektu složku *jni* s C++ souborem. Tento soubor lze následně smazat a použít vlastní zdrojové kódy. Popis, jak spustit vlastní C/C++ kódy, je v následujícím bodu 4. Používané knihovny se přidají obdobně jako v bodě 2. Je potřeba přidat *OpenCV Library*, *min3d-rotation* a *Aruco*.

4. Nastavení JNI. Pozor, pokud je otevřen soubor C/C++ ve složce *jni*, podtrhá se většina kódu a projekt bude potřeba vytvořit znovu. Částečně se podtrhání dá zabránit přidáním cest ke hlavičkám funkcí v *Properties > C/C++ General > Paths and Symbols > Includes*. Častou chybou je, že v *Properties* neexistuje varianta *C/C++ General* nebo *C/C++ Build*, v tom případě je potřeba vytvořit projekt znovu. Pro kompilaci je C/C++ kódů nutné v *Properties > C/C++ Build* zadat vlastní kompilátor *$NDKROOT/ndk-build.cmd* do políčka *Build command*. *NDKROOT* je proměnná prostředí, kterou musíte vytvořit a ukazuje na složku *Android NDK*, která se po stažení a extrahování vytvoří. Kompilace je definovaná v souboru *jni/Android.mk* a *jni/Application.mk*, které je potřeba vytvořit. Pokud už existují, stačí je editovat. Soubor *jni/Android.mk* definuje kódy ke kompilaci podle následující ukázky.

```
LOCAL_PATH := $(call my-dir)

# následující blok zkompiluje kód hello.cpp ...
# ... do knihovny s názvem hello
include $(CLEAR_VARS)

LOCAL_MODULE    := hello
LOCAL_SRC_FILES := hello.cpp
LOCAL_LDLIBS += -llog -ldl

include $(BUILD_SHARED_LIBRARY)
```

Pro získání podpory více knihoven C/C++ a zrychlení kompilace je dobré v *jni/Application.mk* použít následující kód.

```
APP_STL := gnustl_static
APP_CPPFLAGS := -frtti -fexceptions
APP_ABI := armeabi-v7a
```

Aby šlo kód C/C++ spustit, musí implementovat JNI interface. Z programovacího jazyku Java půjdou pustit C/C++ kódy používající knihovnu *jni.h* a funkce, které jsou definované v bloku *extern "C"*. Každá takto definovaná funkce musí být uvedena ve tvaru jako hlavička funkce i volání funkce. Ukázka je na následujícím kódu.

```
#include <jni.h>
#include <android/log.h>

extern "C" {
  JNIEXPORT void JNICALL Java_com_example_algorithm_R6P_HelloWorld(JNIEnv *env);

  JNIEXPORT void JNICALL Java_com_example_algorithm_R6P_HelloWorld(JNIEnv *env){
    __android_log_print(ANDROID_LOG_INFO, "JNI_LOG", "Hello world");
}
```

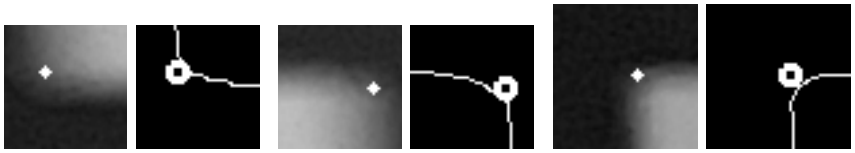
Název funkce se skládá z posloupnosti *programovací-jazyk_balíček_objekt_funkce*, přičemž *balíček* obsahuje místo teček podtržítka. Tuto funkci půjde z Java kódu pustit po načtení knihovny *System.loadLibrary("hello")* a definici nativní funkce *public native static void HelloWorld()*.

5. Inicializace a snímání. Inicializace OpenCV zároveň inicializuje snímání. OpenCV načítá *OpenCVLoader* voláním *initAsync(verze, posluchač_snímků, posluchač_inicializace)*. Parametr *verze* je v současné době *OpenCVLoader.OPENCV_VERSION_2_4_9*, *posluchač_snímků* je objekt implementující *CvCameraViewListener2* a *posluchač_inicializace* je objekt třídy *BaseLoaderCallback*. *CvCameraViewListener2* vrací do povinně implementované metody *onManagerConnected(status)* jestli se povedlo knihovnu OpenCV načíst. Toto je vhodné místo pro začátek snímání voláním *objektOpenCvView.enableView()*. Proměnná *objektOpenCvView* je třídy *CameraBridgeViewBase* a načítá se z XML souboru s uživatelským rozhraním. *CvCameraViewListener2* musí implementovat tři metody, které poskytují informace o startu snímání *onCameraViewStarted(int šířka_snímku,int výška_snímku)*, konci snímání *onCameraViewStopped()* a o přijetí nového snímku *onCameraFrame(CvCameraViewFrame inputFrame)*. Funkce *onCameraFrame(CvCameraViewFrame inputFrame)* musí vracet obrázek vykreslovaný na displeji. Ten lze získat ze vstupní proměnné voláním *inputFrame.rgba()* nebo *inputFrame.gray()*.

Detekce Aruco značek

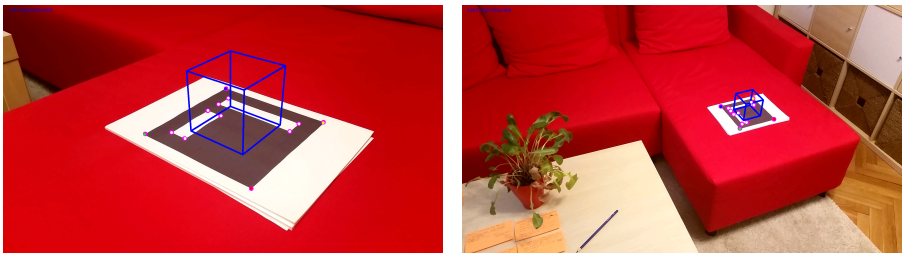
Prvním krokem k detekci značek je jejich vytvoření. Zdrojový kód Aruco značek lze stáhnout z oficiálních stránek. Kód se následně transformuje do projektu programu Visual Studio pomocí aplikace *Cmake*. Po kompilaci ve Visual Studiu lze příkazem *aruco_create_marker [id_značky][výstup.jpg][velikost_px]* vytvořit marker. Následná detekce značky se volá v aplikaci kódem *detect(obrazek_in, detekovane_znacky, parametry_kamery, hrana_v_metrech, mat)* u objektu třídy *MarkerDetector*. Parametr *obrazek_in* je třídy *Mat* z OpenCV knihovny a obsahuje sejmutý obrázek v RGB, *detekovane_znacky* je třídy *Vector<Marker>* a obsahuje výstup po dokončení funkce, *parametry_kamery* je třídy *CameraParameters* a obsahuje kalibrační matici a koeficienty funkce odstraňující radiální zkreslení. *CameraParameters* lze načíst pouze z XML souboru, který je ve formátu OpenCV knihovny. Parametr *hrana_v_metrech* udává délku hrany značky v metrech a je typu *float*. Poslední parametr je objekt třídy *Mat* a není nikde v detektoru použit. Tato detekce běží průměrně 2s na snímcích o rozlišení 1920x1080px na telefonu Samsungu Galaxy S5. Rychlost detekce a časté nedetekování značky byly důvody, proč jsem napsal vlastní detektor. Tento detektor reprezentuje třída *MyDetector* a využívá části kódu původního detektoru. Nejvíce času při detekci zabíralo nalezení seznamu obrysů a jejich filtrování. Zároveň se při pohybu obrys snadno narušil a detektor značku nenašel. Základní vylepšení, které zvyšuje robustnost a zrychluje běh, je

hledání obrysů na zmenšeném obrázku. Následné filtrování je kvůli vypuštění velkého množství malých obrysů také rychlejší. Filtrování probíhá stejně jako v původním kódu. Jsou odstraněny obrysy, které po proložení polynomem neobsahují 4 rohy a nebo nejsou konvexní. Po filtrování zůstane 2 - 6 návrhů značek. Jejich rohy jsou detekovány na původním obrázku, ve výřezech definovaných odhadnutou pozicí ze zmenšeného snímku. Detekce probíhá pomocí proložení nejdelších detekovaných hran přímkami u nichž je pak nalezen průsečík, nebo pokud tato varianta selže, tak Harrisovým detektorem rohů.



Obrázek 4.2. Dvojce obrázků **a)**,**b)**,**c)** ukazují výřezy detekce rohů. Velikost výřezu je určena dynamicky, proto nejsou obrázky stejně velké. První obrázek z dvojce ukazuje výřez okolí rohu, který přichází do zpřesňující funkce a pozici detekovaného rohu v něm. Druhé snímky ve dvojicích vznikly detekcí hran. Na hranách byly detekovány části zachovávající směr. Body těchto částí byli proloženy přímkami. Na obrázcích je vyznačen průsečík těchto přímek kolečkem.

Po upřesnění rohů je výřez oblasti převeden do matice o velikosti 7x7, ve které jsou světlá místa označena 1 a tmavá 0. Zde je doplněno další vylepšení, které rotuje matici, aby byla invariantní a z kódu odhadne pozice vnitřních rohů. Ty jsou upřesněny stejným způsobem jako rohy krajní. Vše je napsáno obecně, pro libovolnou značku, ale na konci je vrácena pouze značka s id = 1. Další změna oproti původní variantě nastává, pokud nebyla značka nalezena, například porušením jejího obrysu. V tom případě jsou pozice jejích rohů odhadnuty z pohybu mezi posledními dvěma snímky. Pak je opět provedeno zpřesnění a kontrola kódu uvnitř značky. Tyto odhady přináší přibližně 30% zmenšení výpadků detekce. Na závěr je každému rohu je přiřazena 2D a 3D souřadnice. Ty jsou výstupem statické metody *MyDetector.detect(obrázek)*. Původní detektor vrací pouze pozici čtyřech krajních rohů v obrázku. Rychlost detekce závisí na snímané scéně a počtu obrysů v ní, stejně jako u původního detektoru. Průměrná rychlost snímání při použití vylepšené varianty je okolo 17 snímků/s při použití přesné detekce rohů a 23 snímků/s při využití Harrisova detektoru.



Obrázek 4.3. Ukázka detekce Aruco značky pomocí třídy *MyDetector*.

5. Experimenty

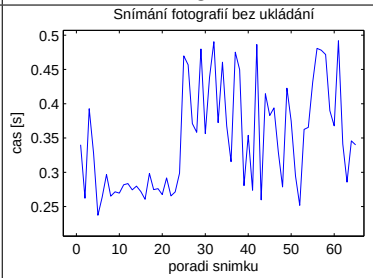
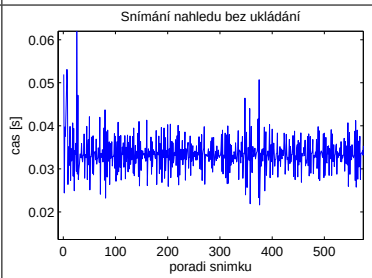
Experimenty byly prováděny za účelem ověření předpokládaného modelu *Rolling Shutter* kamery a zjištění jeho parametrů, kterými je doba expozice a čas vyčítání jedné řádky. První část je zaměřena na přímé získání informací z operačního systému Android. Ve druhé části je měření efektu částečné expozice 2.5. Poslední, třetí část je zaměřena na měření z deformací vznikajících pohybem kamery 2.7.

5.1. Systémové časy

Model kamery, který byl ověřen pomocí následujících experimentů, se vyčítají řádky se zpožděním LD a doba expozice je uvedena v EXIF datech u pořízených fotografií. Pokud by byl známý přesný start a konec snímání, dalo by se LD vyjádřit ze vzorce 5.1.

$$\Delta t = exp + nLD \tag{5.1}$$

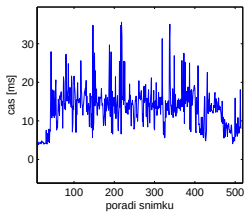
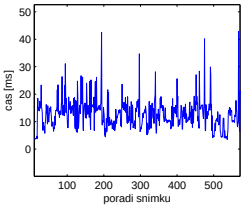
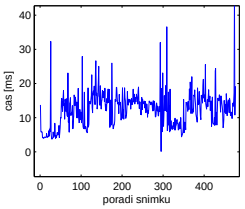
Přičemž Δt udává dobu snímání fotografie od začátku expozice prvního řádku po vyčtení posledního řádku, exp udává dobu expozice jedné řádky, n počet řádků a LD dobu vyčítání jedné řádky. Různé verze Androidu zprostředkovávají různé informace, ale žádná neposkytuje přesný začátek a zároveň konec snímání. Pokud je kamera ovládána pomocí OpenCv, je informací o časech snímání ještě méně. Zároveň nastavení kamery obsahuje parametry ovlivňující rychlost snímání. Nejdůležitějším z nich je, zda jsou snímky pořízeny pro náhled na displeji, video, nebo vyfotografování jedné fotografie.

Rozlišení snímku:	1920x1080px	1920x1080px
Mód snímání	fotografie	náhled
Čas snímání:		
Průměrný čas:	0.348 s	0.035 s
Odchylka času:	0.078 s	0.025 s

Tabulka 5.1. Délky snímání, změřené při použití C-API. Rozlišení jsou nastaveny na maximální hodnotu, kterou podporuje náhled telefonu Samsung Galaxy S5. Při experimentu nebyly snímky ukládány do paměti a délka snímání trvala přibližně minutu. Odchylka času značí směrodatnou odchylku měřené veličiny.

Pro experiment uvedený v tabulce 5.1 byla použita aplikace využívající C-API. Toto rozhraní slouží pro ovládání kamery na zařízeních s operačním systémem Android do verze 4.x. API obsahuje funkci *onShutter()*, která je volána ihned po dokončení snímání fotografie. Je tedy přesně určen konec snímání, ale není dostupná žádná informace o startu snímání. Jako start je u focení fotografií počítán řádek po volání funkce *takePicture()*, vyvolávající pořízení snímku. Čas pořízení fotografie je řádově vyšší, než čas potřebný pro pořízení náhledu. U náhledu není přesně definován začátek, ani konec snímání. Při pořizování náhledu je za konec snímání považován první řádek ve funkci *onPreviewFrame()*, která oznamuje pořízení nového náhledu. Čas začátku snímání nového náhledu je měřen na posledním řádku téže funkce. Průměrný čas pořízení náhledu je 35ms, z čehož vyplývá frekvence 28,6 snímků/s. Bohužel je směrodatná odchylka 25 ms, což dělá variační koeficient 71%. Proto je tato informace nepoužitelná pro určení doby expozice, ověření modelu snímání a výpočtu LD. Zároveň jsou v čase snímání započteny další operace, jelikož snímání obdobné scény bez ukládání je na stejném telefonu při použití C-API2 přibližně 3x rychlejší.

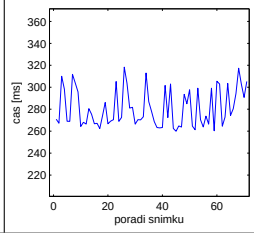
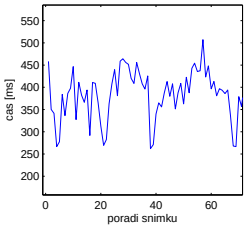
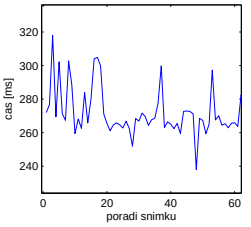
C-API2 umožňuje podrobnější správu kamery a poskytuje podle dokumentace přesnou informaci o počátku snímání a to při focení snímků i při náhledu. Použitím rozhraní, které se nachází pouze na telefonech s Androidem 5 a vyšším, se časy pořízení snímku na stejném telefonu zrychlí, což je vidět v následující tabulce 5.2.

Rozlišení:	1920x1080px	1280x720px	640x480px
Mód snímání	náhled	náhled	náhled
Čas snímání:			
Průměrný čas:	13.304 ms	12.667 ms	12.444 ms
Odchylka času:	5.406 ms	5.257 ms	5.093 ms

Tabulka 5.2. Délky snímání, změřené při použití C-API2 na telefonu Samsung Galaxy S5. Při experimentu nebyly snímky ukládány do paměti a délka snímání trvala přibližně minutu. Odchylka času značí směrodatnou odchylku měřené veličiny.

Experiment uvedený v tabulce 5.2 byl proveden pouze s registrovaným náhledem na výstupu z kamery. V tomto nastavení jsou fotografie snímané scény dostupné ve frekvenci přibližně 80 snímků/s bez závislosti na výběru jednoho z povolených rozlišení. C-API2 na rozdíl od C-API rozlišuje jaké výstupy jsou zaregistrované. Pokud je vyžadováno focení, ale není registrovaný žádný výstup, k focení vůbec nedochází. Náhled má stejně jako v C-API povolené velikosti do 1920x1080px a předává se pro vykreslení ve formátu NV21 potomkům objektů *Surface*. Velké rychlosti je dosaženo díky tomu, že snímání probíhá v rozměrem omezeném módu a odpadá konverze snímků do formátu JPEG. Díky CMOS čipu lze snímat jednotlivé pixely zvlášť a mód náhledu má nejspíše přednastaven výběr pixelů, které exponuje a nesnímá celý snímek. Tuto teorii potvrzuje fakt, že se se změnou rozlišení nemění doba snímání. Další drobnější časové rozdíly tvoří nastavení šablony snímání a následné operace se snímky. Např. pokud je

snímán obraz v šabloně náhledu *TEMPLATE_PREVIEW* a pak jsou snímky ukládány, je čas všech operací pro pořízení jedné fotografie přibližně o 1/3 větší, s 4x větší směrodatnou odchylkou než při použití šablony pro natáčení videa *TEMPLATE_RECORD*. Tyto časové rozdíly jsou pozorovatelné v tabulce 5.3.

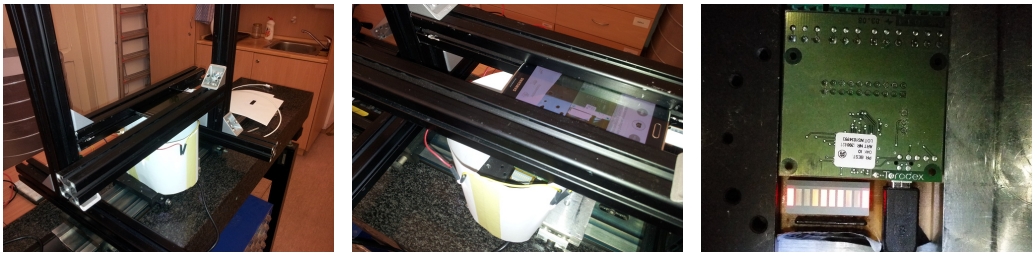
Rozlišení snímku:	5312x2988px	5312x2988px	5312x2988px
Šablona snímání	fotografie	náhled	video
Čas snímání:			
Průměrný čas:	280.333 ms	383.885 ms	271.802 ms
Odchylka času:	17.090 ms	56.393 ms	14.350 ms

Tabulka 5.3. Délky snímání, změřené při použití C-API2 na telefonu Samsung Galaxy S5. Při experimentu byly snímky ukládány do paměti a délka snímání trvala přibližně minutu. Odchylka času značí směrodatnou odchylku měřené veličiny.

Při ukládání snímků v módu *TEMPLATE_RECORD* trvá z celkového času 272 ms potřebného pro pořízení snímku přibližně 38 ms snímání scény, což je vidět z experimentů v sekci 5.3.1. Zbylý čas, přibližně 234 ms, slouží ke konverzi snímku a jeho uložení. Výsledkem měření systémových časů jsou dvě zjištění. Pomocí systémových časů nelze určit dobu vyčítání jednoho řádku kvůli nepřesnému startu nebo konci snímání. Pro běh aplikace v reálném čase je lepší neukládat obrázky do paměti telefonu.

5.2. Rozsvěcení LED

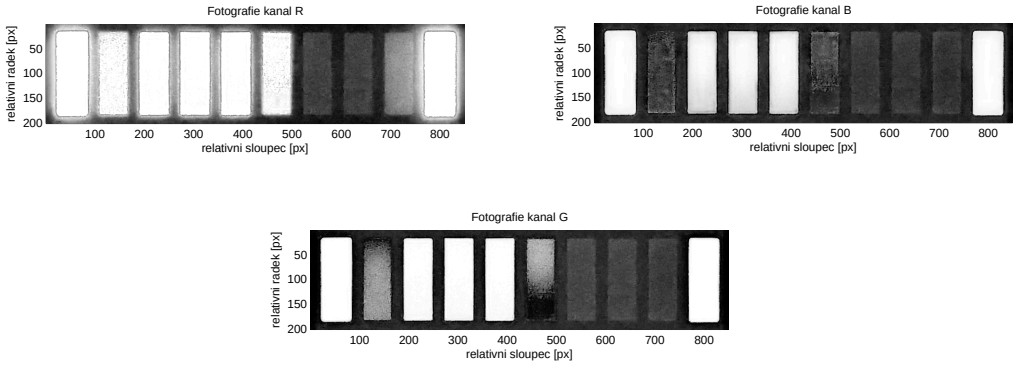
Cílem tohoto experimentu bylo co nejpřesněji odhadnout dobu vyčítání jedné řádky (LD). Experiment probíhal ve dvou částech. V první části byly data měřeny na panelu s deseti led diodami, který byl řízen z operačního systému Linux přes USB. Měření probíhalo při různých rychlostech a změnách na 8 vnitřních diodách. Aparatura použitá při měření je na obrázcích 5.1.



a) **b)** **c)**

Obrázek 5.1. Obrázky **a)** a **b)** zobrazují konstrukci, která byla použita pro měření. Na snímku **c)** je panel s LED diodami.

Pro měření doby LD byl vybrán posun jedné rozsvícené diody. Předpokladem bylo, že rozsvěcení a zhasínání bude ovlivněno RS efektem. Jelikož se řádky snímku začínají exponovat se zpožděním, měl by v být na diodě pozorovatelný přechod zvyšujícího nebo snižujícího se jasu. Tento efekt je pozorovatelný na zeleném kanálu jednoho z měřených snímků 5.2.

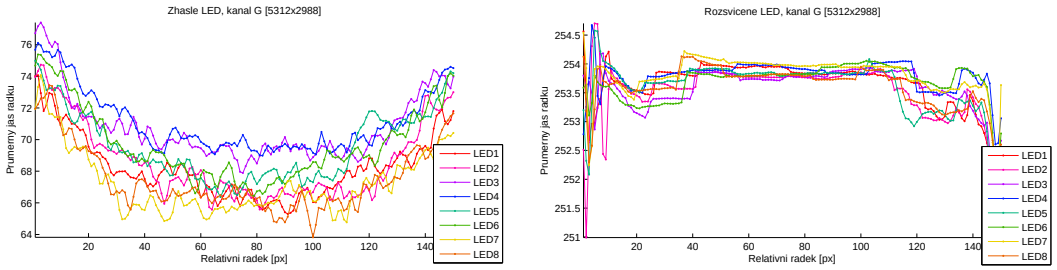


Obrázek 5.2. Naměřené hodnoty jasu na panelu s diodami. Obrázek vznikl posouváním svítící diody po 10 ms. Čísla řádků a sloupců jsou určeny vzhledem k panelu s diodami.

V nasnímaných fotografiích není panel paralelní s osou x, ale obsahuje drobnou odchylku. Snímky jsou před vyčítáním hodnot rotovány homografií H podle osy z, pomocí kubické interpolace.

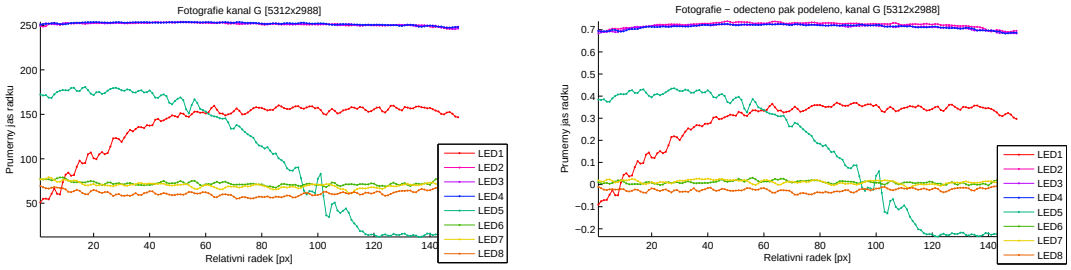
$$H = KR_zK^{-1};\ R_z = \begin{bmatrix} cos(\gamma) & -sin(\gamma) & 0 \\ sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.2}$$

Přičemž K značí kalibrační matici, R_z rotaci podle osy z o úhel γ. Po interpolaci je fotografie oříznuta do podoby snímků 5.2 a jsou extrahovány pozice jednotlivých diod. Pro každý řádek diody je určen aritmetický průměr sloupců, nebo-li každá dioda je reprezentována vektorem průměrného jasu jejích řádků. Výsledné intenzity pro zelený kanál snímku 5.2 jsou zobrazeny na grafech 5.4. Referenční grafy 5.3 slouží k odečtení okolního osvětlení a normalizaci jasu jednotlivých diod. Experiment byl prováděn se stejnosměrně napájeným osvětlením, aby nedocházelo k vnějšímu ovlivňování naměřených hodnot.



a) **b)**

Obrázek 5.3. Grafy referenčních hodnot jasu. Na obrázku **a)**/**b)** jsou zobrazeny vektory jasu zhaslých/rozsvícených diod. Hodnoty zhaslých a rozsvícených diod jsou vyčteny z průměrných snímků. Průměrný snímek zhaslých diod je vypočten z 28 obrázků a rozsvícených z 25 obrázků.



a) **b)**

Obrázek 5.4. Grafy intenzit jasů na diodách při posunu svítící diody po 10 ms. Graf **a)** zobrazuje naměřené hodnoty bez úprav. Na grafu **b)** jsou intenzity po odečtení jasu zhasnutých diod a vydělení rozsvícenými diodami.

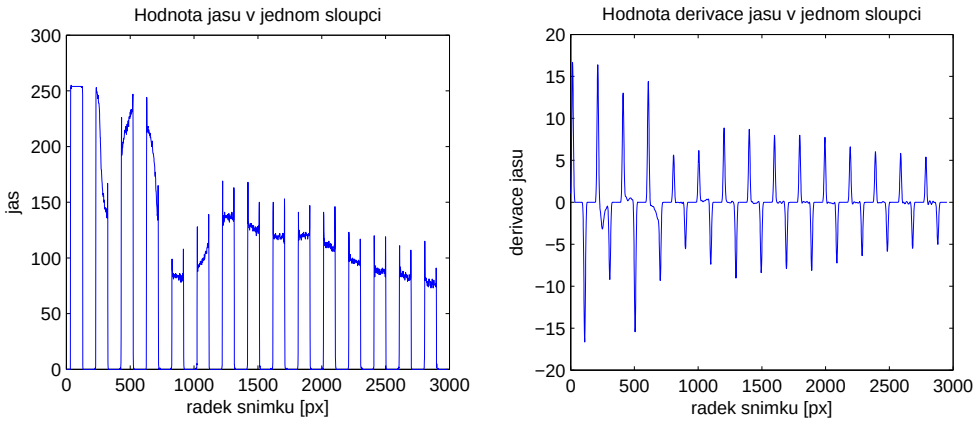
Na výsledných křivkách je po odečtení referenčního osvětlení vidět posun *LED6*, *LED7* a *LED8* na hodnotu blízkou nule. Pro rozbor LED5, která se dostává do záporných hodnot, je potřeba shrnout nastavení kamery. Jelikož nelze měnit ohniskovou vzdálenost manuálně, bylo zapnuté automatické ostření. Tento konkrétní snímek byl focen s podle dokumentace manuálním vyvážením bílé a manuálně nastavenou dobou expozice. Z průběhu jasu na LED5 vyplývá, že dochází ke změnám vyvážení barev. Bohužel se situace nezmění ani při (dle dokumentace) zapnutém vyvážení bílé a automatickém určení doby expozice. Tyto hodnoty se regulují automaticky, nehledě na nastavení. Při vyhodnocení ostatních kanálů se objevují další nedostatky. Jas červené barvy je ve snímcích saturován v maximální hodnotě 255, kvůli čemuž průběh nepopisuje reálné změny jasu. Kanál obsahující modrou barvu neobsahuje žádnou informaci o rozsvěcení a zhasínání. Všechny hodnoty jsou po odečtení a podělení referenčními průběhy jasu buď v okolí konstantní hodnoty odpovídající rozsvíceným diodám, nebo v okolí nuly.

Druhá fáze se zaměřuje na zpracování snímků pořízených z jednoho intenzivního světelného zdroje, který střídá periodicky rozsvícený a zhasnutý stav. Aparatura použitá pro měření je na obrázku 5.5.

a) **b)** **c)**

Obrázek 5.5. Experiment je založený na efektu částečné expozice. Fotografie **a)** a **b)** vykreslují pozice baterky s pulzně šířkovou modulací (PWM) a fotodiody. Měření doby rozsvícení a zhasnutí je prováděno na osciloskopu **c)**.

Příklad pořízeného snímku je 2.5 **b)**. Střídání tmy a světla je způsobeno krátkou dobou expozice, která se pohybuje v testovacích sadách mezi 40μs až 120μs. Pro každý sloupec snímku je spočtena jeho derivace. Hodnoty jasu jednoho sloupce jsou vidět na obrázku 5.6 **a)** a jeho derivace na 5.6 **b)**.



Obrázek 5.6. Na grafu **a)** je znázorněna hodnota jasu 1500 sloupce na 15 snímku testovací sady s rozlišením 5312x2988 px. Graf **b)** odpovídá derivaci **a)** a vznikl pomocí konvoluce s derivací Gaussovi funkce o rozměru 1x37 hodnot.

Vzdálenost mezi vrcholy derivace jasu udává počet řádků, které jsou po podělení časem rozsvíceného případně zhasnutého zdroje rovny LD. Průměrné počty řádků pro různá rozlišení a jim odpovídající časy jsou v následující tabulce 5.4.

Rozlišení snímku:	5312x2988px	3264x1836	1920x1080px	640x480
Délka svícení	1102μs	1102μs	1102μs	1102μs
Délka tmy	972μs	972μs	972μs	972μs
Prům. osvícených řádků:	104.491 px	63.867 px	36.945 px	15.871 px
Odchylka osvícených řádků:	1.155 px	0.562 px	0.603 px	0.364 px
Prům. neosvícených řádků:	93.581 px	57.943 px	34.607 px	15.915 px
Odchylka neosvícených řádků:	0.995 px	0.571 px	0.625 px	0.441 px
Prům. LD osvícených řádků:	11.776μs	19.019μs	31.843μs	69.244μs
Prům. LD neosvícených řádků:	9.302μs	15.219μs	26.309μs	61.243μs
Prům. LD:	10.539μs	17.118μs	29.076μs	65.243μs
Prům. LD při 2988 řádcích:	**10.539μs**	**10.519μs**	**10.509μs**	**10.481μs**

Tabulka 5.4. Tabulka s naměřenými časy při periodickém rozsvěcení a zhasínání světelného zdroje. Frekvence světelného zdroje byla 482,3 Hz. Odchylka času značí směrodatnou odchylku měřené veličiny.

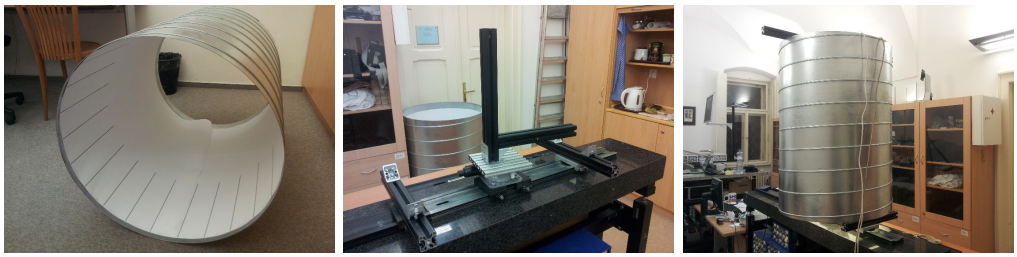
Z hodnot v tabulce vyplývá, že jsou vždy vyčítány všechny řádky snímku a menší rozlišení jsou získány následnou interpolací. To samé platí u různých rozlišení náhledu 5.2. Při vyčítání všech řádků, je doba LD rovna přibližně 10,5μs. Počet pixelů přechodu ze světlého pruhu do tmavého pruhu ve snímku odpovídá době expozice vydělené LD, konkrétně je mezi 4 px až 12 px.

5.3. Rotace kamery

Pomocí tohoto experimentu byla změřena LD a modelována doba expozice jednoho řádku. Průběh snímání byl zkoumán pro různá rozlišení a různé rychlosti pohybu.

Při experimentu byl telefon umístěn středem kamery do osy válce o průměru 35 cm, který měl na vnitřní ploše vertikální čáry o tloušťce 2 mm. Celý experiment se skládal ze

dvou fází. V první fázi byly nafoceny svislé čáry bez pohybu pro ověření jejich vertikální orientace a kvůli možnosti pozdější kompenzaci nepřesností. Ve druhé fázi byl zkoumán vliv otáčení na tvar a jas čar. Pomocí rotační plošiny OPTEN a řídící jednotky MARS 2 bylo telefonem otáčeno konstantní rychlostí 2s, 4s a 8s/otáčku.



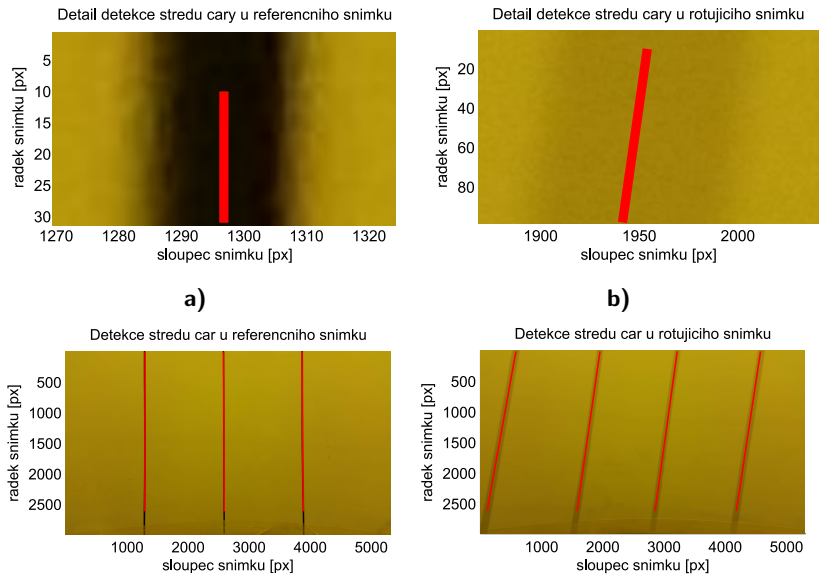
a) **b)** **c)**

Obrázek 5.7. Aparatura pro rotační experiment. Obrázek **a)** obsahuje snímaný válec, **b)** rotační plošinu OPTEN a **c)** celou aparaturu při experimentu.

Experiment byl osvětlen shora pomocí LED reflektorů, aby bylo dosaženo co nejkratší doby expozice. V důsledku rotace kamery vůči scéně se deformuje tvar čar, mění se jejich šířka a intenzita jasu. Tyto tři změny zkoumají a popisují následující tři části.

5.3.1. Zkosení čar

Pomocí rotace kamery dochází ke změně snímané scény během vyčítání jednotlivých řádků. Svislé čáry se nezanedbatelně deformují, což je vidět na obrázcích 5.8.



Obrázek 5.8. Snímky $\{a, c\}$ zobrazují detekci čar bez rotace kamery a snímky $\{b, d\}$ při rotaci $\omega = 2$s/otáčka. Horní dvojce $\{a, b\}$ znázorňuje detail subpixelové detekce středu čar. Snímky byly pořízeny v rozlišení 5312x2988px a jsou na nich pozorovatelné nezanedbatelné deformace vertikálních čar způsobené rotací.

Prvním krokem k určení LD je detekce čar. Jejich pozice jsou určeny pomocí konvoluce s derivací Gaussovy funkce. Velikost konvolučního jádra derivace Gaussovy funkce se mění pro různé rozlišení a rychlosti otáčení. Následně probíhá filtrace nalezených lokálních extrémů, na základě omezené tloušťky čar a jejich maximální počtu v jednom snímku. Po nalezení čar je jejich střed učen jako průměrná pozice ve čtverci o velikosti šířky čáry. Tyto středy jsou detekovány v horních a spodních částech snímku zvlášť. Řádek spodní detekce je určen na základě výšky snímku, jelikož se ve spodní části nachází úchyty držící snímaný válec a horizontální čára pro ověření, jestli leží kamera na vertikální ose válce. Následně jsou tyto konce párovány na základě povolených dynamicky určených rozsahů zkosení, které mohlo nastat. U detekovaných čar je určen úhel rotace $\Delta\theta$ mezi horními a spodními středy v radiánech.

Algorithm 1 **Určení rotace čar.**

1: $u \leftarrow K^{-1}[u_1, u_2, 1]'$
2: $v \leftarrow K^{-1}[v_1, v_2, 1]'$
3: $u_{1,3} \leftarrow u([1, 3])$
4: $v_{1,3} \leftarrow v([1, 3])$
5: **return** $acos((u'_{1,3}v_{1,3})/(norm(u_{1,3})norm(v'_{1,3})))$

Proměnné u, v reprezentují horní a spodní konec čáry. Čas rotace $t(\Delta\theta)$ o úhel mezi vrchním a spodním koncem čáry je popsatelný vztahem $t(\Delta\theta) = (\omega\Delta\theta)/(2\pi)$, kde ω značí počet sekund na otáčku. LD je určen poměrem $LD = t(\Delta\theta)/(v_1 - u_1)$. Výsledné úhly vycházejí pro různé rozlišení a rychlosti různě, avšak po přepočtu na čas $t(\Delta\theta)$, který je suma LD přes řádky mezi horními a spodními konci čar, dostaneme (pomineme-li nepřesnost měření) stejné hodnoty.

Rozlišení [px]:	5312x2988	5312x2988	5312x2988	3264x1836	3264x1836	3264x1836
Čas otáčky:	2s	4s	8s	2s	4s	8s
Poč. detek. čar:	679	789	666	739	615	1051
Průměrné $t(\Delta\theta)$**:**	30.347 ms	29.668 ms	28.042 ms	29.699 ms	28.287 ms	28.715 ms
Odchylka $t(\Delta\theta)$**:**	0.535 ms	1.432 ms	2.758 ms	1.141 ms	2.524 ms	4.013 ms
LD:	10.156μs	9.929μs	9.385μs	16.176μs	15.407μs	15.640μs
LD při 2988 řádcích:	10.156μs	9.929μs	9.385μs	9.940μs	9.467μs	9.610μs

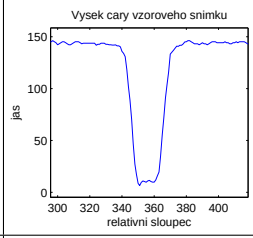
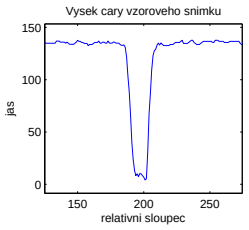
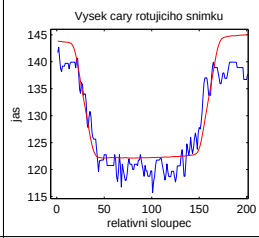
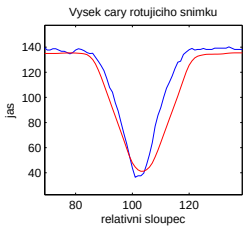
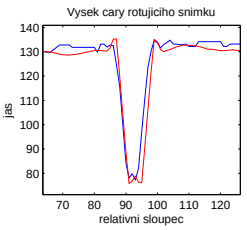
Rozlišení [px]:	1920x1080	1920x1080	1920x1080	640x480	640x480	640x480
Čas otáčky:	2s	4s	8s	2s	4s	8s
Poč. detek. čar:	743	1094	1096	623	580	634
Průměrné $t(\Delta\theta)$**:**	30.561 ms	30.009 ms	29.115 ms	29.596 ms	22.868 ms	27.020 ms
Odchylka $t(\Delta\theta)$**:**	0.446 ms	1.475 ms	3.551 ms	0.611 ms	1.524 ms	2.707 ms
LD:	28.297μs	27.786μs	26.958 μs	61.657μs	47.642μs	56.291μs
LD při 2988 řádcích:	10.228μs	10.043μs	9.744 μs	9.905μs	7.653μs	9.043μs

Tabulka 5.5. Tabulka obsahuje souhrn informací změřených při rotačním experimentu. Pro jednotlivé rozlišení a rychlosti otáčení obsahuje počet detekovaných čar, průměrnou dobu snímání fotografií a směrodatnou odchylku snímání fotografií. Dobu vyčítání jedné řádky (LD), za předpokladu, že je počet snímaných řádků definován velikostí fotografie a dobu vyčítání řádky, pokud je vyčítáno všech 2988 řádků čipu.

S přibývající rychlostí otáčení se zmenšuje směrodatná odchylka úhlů. Nepřesnosti při měření jsou dány tím, že jsou měřené čáry umístěné na válci. Při otáčení se čára umístěná na straně ležící ve směru otáčení válce vzdaluje od kamery a čára na protější straně se ke kameře přibližuje. Vzdalující se čára má menší úhel jak přibližující se čára. Tento jev je pozorovatelný na všech snímcích. Čím větší rychlostí se kamera otáčí, tím větší je úhel a relativní chyba se vůči němu snižuje. V porovnání s částečnou expozicí je tento experiment více náchylný na chyby způsobené mechanickou konstrukcí, a kvůli tomu je měření méně přesné.

5.3.2. Šířka čar

Šířka čáry závisí na času, po kterou byla čára exponována. Doba expozice se dá vyčíst z EXIF dat snímku a pro její ověření je napsána simulace rotace čar. Tato simulace se skládá ze dvou částí. První část je skript v Matlabu, který vybere jeden řádek z vzorového snímku a jeden z snímku pořízeného při rotaci kamery. Načte údaje o snímku a vytiskne potřebné data pro běh simulace, což je parametr *exif.DigitalCamera.ExposureTime*, rychlost rotace ω a úhel rotace o 1px. Zároveň exportuje vzorový řádek do csv souboru. Simulace je napsaná v jazyce Java. Načte vzorový řádek a rotuje s ním podle podle zadaných parametrů. Výstup simulace se opět předá přes pomocný csv soubor. Na závěr je v Matlabu synchronizována pozice čáry a vykreslen graf porovnávající simulaci s reálnou čárou z rotujícího snímku. Příklad simulace je v tabulce 5.6.

Rozlišení snímku:	5312x2988 px	3264x1836 px	640x480 px
Čas otáčky:	2s	8s	4s
Čas expozice:	7.194ms	7.092ms	6.536ms
Úhel rotace o 1px:	0.172ms	0.282ms	1.311ms
Graf vzor. čáry:			
Graf rot. čáry:			
Číslo vzor. snímku:	23	7	9
Číslo rot. snímku:	25	9	11
Porovnávaný řádek:	500	325	111

Tabulka 5.6. Simulace tloušťky čar při různých rozlišeních a různé rychlosti otáčení. Modrou barvou je vykreslena naměřená intenzita jasu a červeně simulovaný průběh. Červená křivka vznikla rotací čáry ze vzorového grafu po dobu expozice.

Na výstupu simulace šířka čáry přibližně odpovídá reálné šířce, která vznikla otáčením. Simulace potvrzuje, že hodnota *ExposureTime* odpovídá času expozice jednoho řádku.

5.3.3. Jas čar

Během expozice řádku se ve snímku mění pozice pixelů. Jejich jas je následně kombinací jasu čáry a jasu jejího okolí. Tento jev je vidět na obrázku 5.8b, který vznikl rotací čáry 5.8a. Ke složení jasů $I(c)$ čáry c a jasu pozadí $I(p)$ pozadí p dochází v poměru $\alpha = s(c_0)/s(c_\omega)$, kde $s(c)$ značí šířku čáry c. Spodní index u čáry značí rychlost otáčení. Výsledný jas popisuje rovnice 5.3.

$$I(c_\omega) = \alpha I(c_0) + (1 - \alpha) I(p) \tag{5.3}$$

Zároveň můžeme vyjádřit šířku čáry $s(c_\omega)$ při rotaci ω a známé době expozice jednoho řádku exp pomocí rovnice 5.4.

$$s(c_\omega) = \frac{2\pi exp}{\omega \theta_{px}} \tag{5.4}$$

Hodnota θ_{px} reprezentuje úhel otočení o jeden pixel. Z předchozího vztahu lze vyjádřit dobu expozice pomocí rovnice 5.5.

$$exp = \frac{\omega \theta_{px} s(c_0)}{2\pi} \frac{(I(c_0) - I(p))}{(I(c_\omega) - I(p))} \tag{5.5}$$

Výsledná chyba závisí na přesnosti určení šířky čáry ve vzorovém snímku, jasu pozadí a jasu čáry pořízené při rotaci. Při výpočtu pomocí tloušťky čar ze vzorce 5.4 je chyba 2 - 5x větší než při výpočtu pomocí jasů, vzorec 5.5. Průměrné chyby pro jednotlivé testovací sady jsou v tabulce 5.7.

Rozlišení [px]:	5312x2988	5312x2988	5312x2988	3264x1836	3264x1836	3264x1836
Čas otáčky:	2s	4s	8s	2s	4s	8s
Průměrná chyba 5.4:	1.496 ms	1.727 ms	1.091 ms	1.486 ms	1.572 ms	1.316 ms
Odchylka chyby 5.4:	0.137 ms	0.239 ms	0.743 ms	0.161 ms	0.156 ms	0.803 ms
Průměrná chyba 5.5:	0.313 ms	0.245 ms	0.266 ms	0.353 ms	0.604 ms	0.222 ms
Odchylka chyby 5.5:	0.237 ms	0.166 ms	0.131 ms	0.267 ms	0.195 ms	0.139 ms

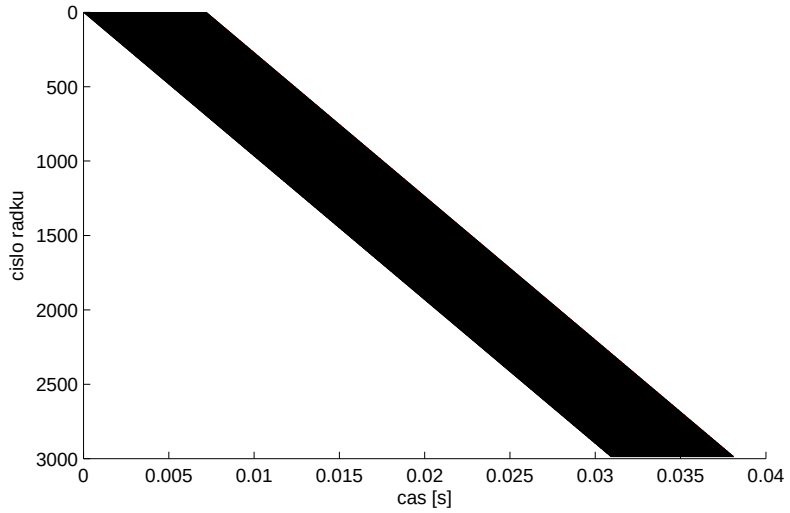
Rozlišení [px]:	1920x1080	1920x1080	1920x1080	640x480	640x480	640x480
Čas otáčky:	2s	4s	8s	2s	4s	8s
Průměrná chyba 5.4:	1.260 ms	1.396 ms	1.310 ms	0.431 ms	0.895 ms	0.260 ms
Odchylka chyby 5.4:	0.142 ms	0.217 ms	0.297 ms	0.155 ms	0.176 ms	0.080 ms
Průměrná chyba 5.5:	0.388 ms	0.271 ms	0.187 ms	0.823 ms	0.243 ms	0.061 ms
Odchylka chyby 5.5:	0.304 ms	0.207 ms	0.131 ms	0.412 ms	0.187 ms	0.048 ms

Tabulka 5.7. Chyba určení expozice při výpočtu pomocí 5.4 a 5.5.

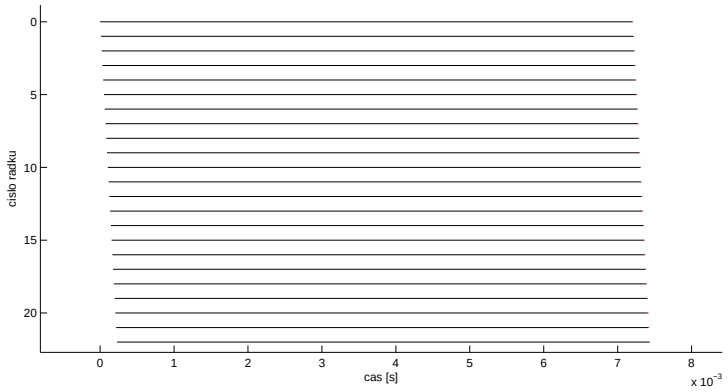
Na všech snímcích testovací sady byly detekovány čáry, vybrán výsek řádku, kterým čára prochází, a spočtena šířka čáry, jas čáry a jas okolního pozadí. Z těchto hodnot se vypočítaly průměrné hodnoty pro snímek, na kterých se určovala chyba vypočítané expozice oproti expozici uvedené v *EXIF* datech. Z průměrných hodnot pro snímek se průměrem získaly průměrné hodnoty pro testovací sady.

5.3.4. Souhrn z měření rolling shutter efektu

Pomocí dvou experimentů byla ověřena doba vyčítání jedné řádky, která je průměrně 10.35μs. Další dvě metody ověřily dobu expozice a skutečnost, že jí odpovídá hodnota uvedená v EXIF datech obrázku. Výsledný model snímání při expozici 1/132 s a průměrné době LD 10.35μs je na obrázku 5.10.



Obrázek 5.9. Expozice jednotlivých řádků je znázorněna černě a jejich doba vyčítání, viditelná při přiblížení, je vykreslena červeně. Snímek je v rozlišení 5312x2988px při trvání expozice 1/132 s a vyčítáním řádků je v čase 10.35 μs



Obrázek 5.10. Detail modelu vyčítání jednotlivých řádků na telefonu Samsung Galaxy S5. Expozice jednotlivých řádků je znázorněna černě a jejich doba vyčítání, viditelná při přiblížení, je vykreslena červeně. Snímek je v rozlišení 5312x2988px při trvání expozice 1/132 s a vyčítáním řádků je v čase 10.35 μs

6. Určení pozice kamery

Následující kapitola porovnává vlastnosti implementovaných algoritmů. První část konkrétně, časové náročnosti a reprojekční chyby algoritmů P3P [9] a P5P, P6P, PnP [18], které předpokládají pořízení obrázků bez pohybu kamery během jejich vyčítání. Výsledky jednotlivých algoritmů jsou na grafech 6.1 a v tabulce 6.1. Algoritmy P5P, P6P a PnP využívají Lambert-Markvartovu optimalizaci reprojekční chyby a jsou inicializovány pomocí geometrie nalezené P3P algoritmem. Uváděné časy výpočtů těchto algoritmů jsou složeny z času potřebného pro inicializaci a samotného času výpočtu.

Název videa:	chuze.mp4	pozice.mp4	rotace.mp4	rotacex.mp4	rotacey.mp4
Algoritmus:	P3P	P3P	P3P	P3P	P3P
Průměrná chyba:	1.84 px	16.803 px	4.569 px	15.295 px	5.536 px
Odchylka chyby:	4.468 px	21.897 px	26.484 px	19.318 px	6.980 px
Prům. čas výpočtu:	0.608 ms	0.913 ms	0.915 ms	0.769 ms	0.879 ms

Název videa:	chuze.mp4	pozice.mp4	rotace.mp4	rotacex.mp4	rotacey.mp4
Algoritmus:	P5P	P5P	P5P	P5P	P5P
Průměrná chyba:	1.340 px	6.889 px	3.309 px	6.157 px	3.445 px
Odchylka chyby:	1.682 px	4.702 px	26.217 px	4.098 px	2.483 px
Prům. čas výpočtu:	1.060 ms	2.033 ms	1.397 ms	1.796 ms	1.794 ms

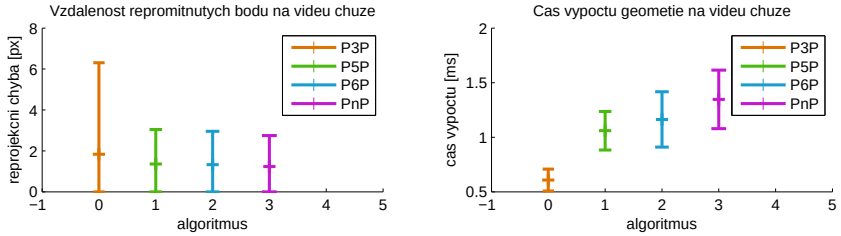
Název videa:	chuze.mp4	pozice.mp4	rotace.mp4	rotacex.mp4	rotacey.mp4
Algoritmus:	P6P	P6P	P6P	P6P	P6P
Průměrná chyba:	1.326 px	6.741 px	3.283 px	5.934 px	3.347 px
Odchylka chyby:	1.633 px	4.796 px	26.211 px	4.166 px	2.548 px
Prům. čas výpočtu:	1.163 ms	2.101 ms	1.327 ms	1.670 ms	1.517 ms

Název videa:	chuze.mp4	pozice.mp4	rotace.mp4	rotacex.mp4	rotacey.mp4
Algoritmus:	PnP	PnP	PnP	PnP	PnP
Průměrná chyba:	1.233 px	5.724 px	4.047 px	4.839 px	2.965 px
Odchylka chyby:	1.518 px	4.882 px	24.832 px	4.486 px	2.640 px
Prům. čas výpočtu:	1.347 ms	2.113 ms	1.373 ms	1.717 ms	1.756 ms

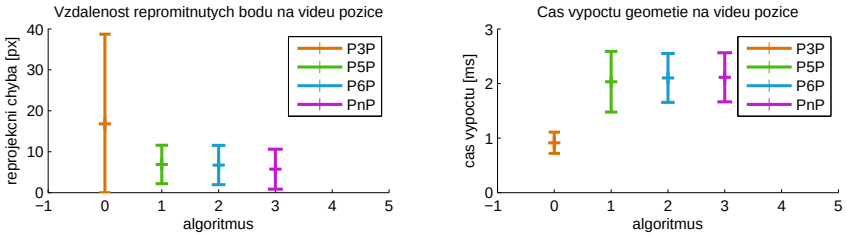
Tabulka 6.1. Tabulka shrnuje reprojekční chybu, její směrodatnou odchylku a délku trvání výpočtu geometrie kamery. Chyba udává Eukleidovskou vzdálenost 3D bodů promítnutých do snímku, od jejich 2D korespondencí.

Z naměřených dat vyplývá, že čas výpočtu uvedených algoritmů knihovny OpenCV není závislý na počtu bodů a trvá přibližně stejně dlouho jako výpočet inicializačního řešení. S přibývajícím počtem bodů klesá reprojekční chyba, ale nemění se časová náročnost. Reprojekční chyby vznikající pohybem jsou výrazně menší, než chyby způsobené rotací. Tento jev zmiňují také Ringaby aj. [20] ve své disertační práci. Následující grafy 6.1 porovnávají naměřené údaje po vizuální stránce. Testovaná videa jsou blíže popsána v příloze A.4.

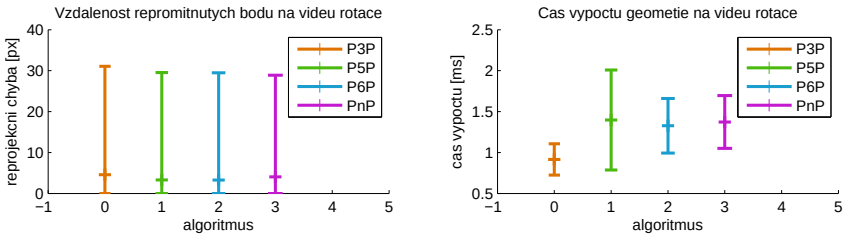
Všechny algoritmy selhávají při výrazných změnách pozice, což je pozorovatelné na směrodatné odchylce videa **c)** grafu 6.1. Při rotaci kolem osy x je velikost chyby 2x



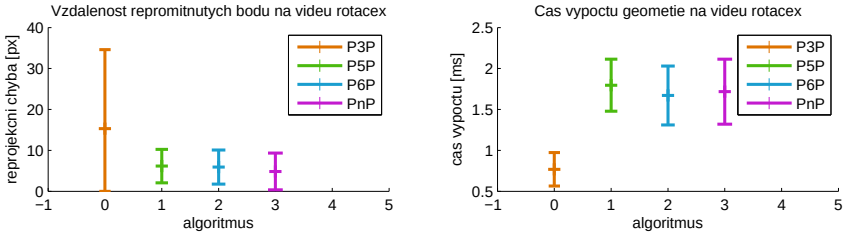
a) Vyhodnocované video bylo pořízené při chůzi okolo detekované značky. Značka byla vzdálená přibližně jeden metr od kamery.



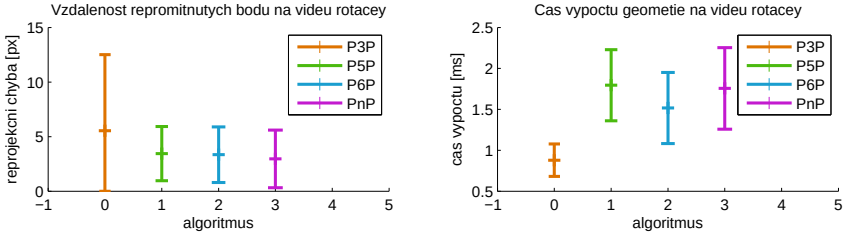
b) Kamera měnila během natáčení vyhodnocovaného videa orientaci, ale zachovávala stejnou pozici vůči referenční značce.



c) Při videu se kamera pohybovala a rotovala, aby bylo dosaženo co nejvýraznějšího zkreslení.



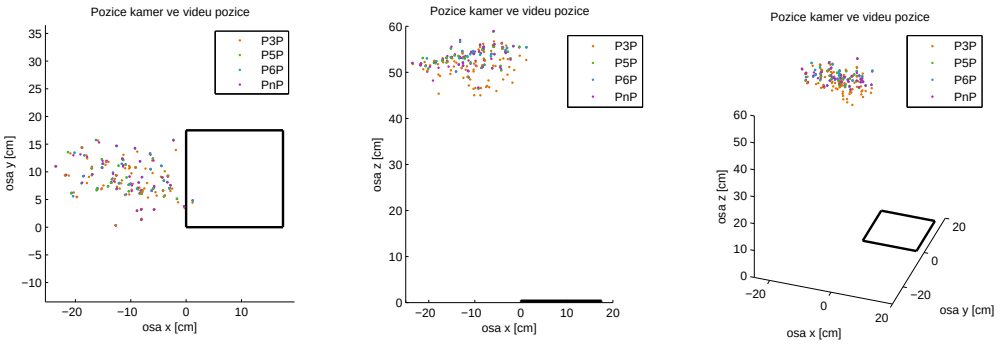
d) Během natáčení tohoto videa kamera rotovala pouze kolem osy x.



e) Během natáčení tohoto videa kamera rotovala pouze kolem osy y.

Obrázek 6.1. Grafy v levé polovině znázorňují reprojekční chybu. V pravé polovině jsou grafy znázorňující časovou složitost výpočtu. Do časů je započten i čas pro získání inicializačního řešení pomocí P3P algoritmu.

až 3x větší než u obdobné rotace kolem osy y. Ačkoliv je reprojekční chyba jedním z nejčastěji uváděných kritérií, její malá hodnota neznamená správně určenou polohu kamery. Body na obrázcích 6.2 udávají pozici kamer vůči referenční značce v centimetrech vyhodnocené z videa *pozice.mp4*.



a) **b)** **c)**

Obrázek 6.2. Obrázky zachycují vypočítanou pozici kamery. Kamera byla během snímání zapřena ihned vedle objektivu a měnila svoji pozici vůči referenční značce v řádu milimetrů až jednotek centimetrů.

Obrázek 6.2 ukazuje na špatné určení pozice až o 20% při minimalizaci chyby využitím Lambert-Markvartovy optimalizace.

6.1. R6P algoritmus

Oproti výše porovnávaným algoritmům bere algoritmus R6P v potaz možný pohyb kamery během vyčítání snímků. Algoritmus R6P v době odevzdání této diplomové práce nefungoval na planárních scénách. Tento nedostatek byl vyřešen přidáním druhé Aruco značky a rozšířením detekce na více značek najednou. Porovnání časů výpočtu a reprojekčních chyb porovnávaných algoritmů je v tabulkách 6.2, 6.3 a grafech 6.3. Algoritmus R6P byl inicializován pomocí P3P algoritmu, stejně jako algoritmy knihovny OpenCV. Vysoká reprojekční chyba R6P algoritmu je dána využitím projekce pomocí perspektivní geometrie místo vzorce 3.1.

Název videa:	rx.mp4	ry.mp4	r.mp4
Algoritmus:	P3P	P3P	P3P
Průměrná chyba:	15.418 px	8.748 px	15.801 px
Odchylka chyby:	17.894 px	11.054 px	18.750 px
Prům. čas výpočtu:	0.601 ms	0.867 ms	0.796 ms

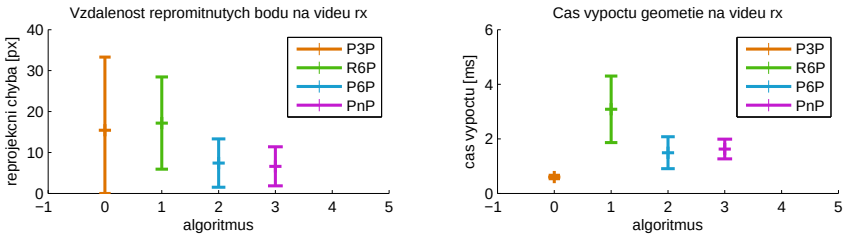
Název videa:	rx.mp4	ry.mp4	r.mp4
Algoritmus:	R6P	R6P	R6P
Průměrná chyba:	17.186 px	12.666 px	22.223 px
Odchylka chyby:	11.276 px	10.640 px	21.881 px
Prům. čas výpočtu:	3.083 ms	3.473 ms	4.278 ms

Tabulka 6.2. Tabulka shrnuje reprojekční chybu, její odchylku a délku trvání výpočtu geometrie kamery.

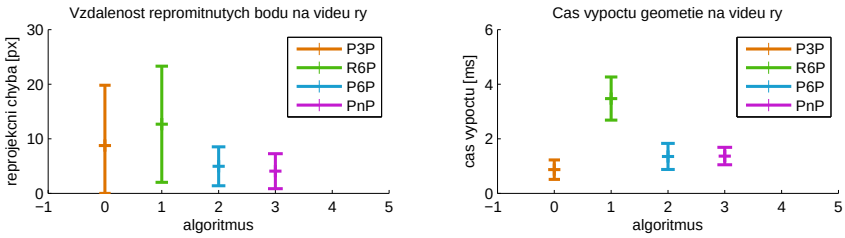
Název videa:	rx.mp4	ry.mp4	r.mp4
Algoritmus:	P6P	P6P	P6P
Průměrná chyba:	7.403 px	4.955 px	7.821 px
Odchylka chyby:	5.900 px	3.558 px	6.228 px
Prům. čas výpočtu:	1.491 ms	1.352 ms	1.650 ms

Název videa:	rx.mp4	ry.mp4	r.mp4
Algoritmus:	PnP	PnP	PnP
Průměrná chyba:	6.605 px	4.058 px	7.013 px
Odchylka chyby:	4.771 px	3.199 px	4.966 px
Prům. čas výpočtu:	1.627 ms	1.365 ms	1.762 ms

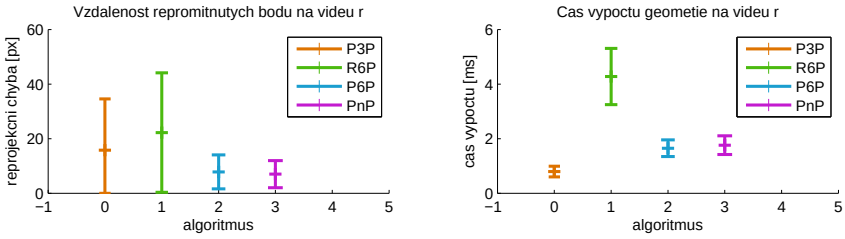
Tabulka 6.3. Tabulka shrnuje reprojekční chybu, její odchylku a délku trvání výpočtu geometrie kamery. Chyba udává Eukleidovskou vzdálenost 3D bodů promítnutých do snímku, od jejich 2D korespondencí.



a) Během natáčení tohoto videa kamera rotovala pouze kolem osy x.



b) Během natáčení tohoto videa kamera rotovala pouze kolem osy y.



c) Při videu se kamera pohybovala a rotovala, aby bylo dosaženo co nejvýraznějšího zkreslení.

Obrázek 6.3. Grafy v levé polovině znázorňují reprojekční chybu. V pravé polovině jsou grafy znázorňující časovou složitost výpočtu. Do časů je započten i čas pro získání inicializačního řešení pomocí P3P algoritmu.

7. Závěr

V diplomové práci byly zkoumány možnosti pořízení fotografií na zařízení s elektronickou štěrbinovou uzávěrkou. Hlavní pozornost byla věnován deformacím, ke kterým dochází při pohybu kamery, během vyčítání snímků. Cílem bylo:

- zjistit, jaké deformace mohou postupným vyčítáním snímků nastat,
- popsat proč k těmto deformacím dochází a změřit parametry kamery, potřebné pro jejich odstranění,
- provést implementaci algoritmu P3P, který určuje polohu kamery ze tří bodů a při kterém se předpokládá, že se vzájemná poloha kamery a snímané scény během vyčítání obrázku neměnila ,
- porovnání a implementace poskytnutého R6P algoritmu, který určuje polohu kamery ze šesti bodů a uvažuje pohyb kamery.

Výsledná práce se skládá ze dvou hlavních částí: z teoretické a z praktické části. V první, teoretické části je popsán princip snímání obrázků u dvou nejrozšířenějších čipů (CMOS a CCD), jsou porovnány jejich výhody a nevýhody a popsány deformace. Deformace, které vznikají u rozšířenějšího čipu CMOS, jsou rozčleněny do dvou skupin skupin podle jejich fyzikální podstaty a demonstrovány na příkladech. Jedním z příkladů je rekonstrukce trojdimenzionální scény za pomoci snímků bez deformací a s deformacemi. Součástí první části práce je i souhrn řešení jak redukovat zkreslení a také popis vývoje vědeckého výzkumu, který se tímto tématem zabývá.

Těžiště práce spočívá ve druhé, praktické části. Prvním krokem bylo nalezení způsobů jakými lze ovládat kameru. Za tímto účelem byly napsány tři aplikace. Jejich implementace je popsána v podrobných návodech společně s řešením problémů, které při ní mohou nastat. Kódy těchto aplikací jsou k dispozici na přiloženém DVD. Součástí jedné z aplikací je kód zvyšující výrazným způsobem spolehlivost, přesnost a rychlost detekce Aruco značek, které jsou důležité pro porovnání reprojekční chyby testovaných algoritmů. Rychlost této upravené varianty detektoru je přibližně 40x vyšší než rychlost původní knihovny, přesnost detekce je na subpixelové úrovni a počet nedetekovaných značek je snížen o více než 30%.

Druhým krokem bylo změření parametrů kamery, konkrétně doby vyčítání jedné řádky a doby expozice. Parametry kamery jsou zkoumány na celé řadě experimentů, konkrétně pět experimentů měří dobu vyčítání jednoho řádku obrázku, dva experimenty slouží k určení doby expozice a dva počítají expozici z hodnot čtených ze snímku. Všechny experimenty byly porovnány z hlediska přesnosti zjištěných údajů. Nejpřesnější se ukázala metoda periodického střídání světla a tmy. Tímto způsobem byla stanovena doba vyčítání jedné řádky 10.5μs se směrodatnou odchylkou 0.1μs.

Posledním krokem byla implementace algoritmů pro určení polohy kamery. Implementovány byly algoritmy P3P, P5P, R6P, P6P a PnP. Výsledky těchto algoritmů byly navzájem porovnány. Všeobecně lze říci, že čím více bodů je pro výpočet použito, tím nižší je reprojekční chyba. U algoritmů P3P, P5P, P6P a PnP bylo zjištěno, že zmenšující se reprojekční chyba nemusí znamenat přesnější určení polohy kamery. Algoritmus R6P vykazoval sice při projekci pomocí perspektivní geometrie vyšší reprojekční chybu,

ale vzhledem k tomu, že jako jediný uvažuje pohyb kamery při snímání, jsou výsledky, kterých dosahuje, přesnější než výsledky ostatních algoritmů.

Literatura

[1] Omar Ait-Aider, Nicolas Andreff, Jean Marc Lavest, and Philippe Martinet. Simultaneous object pose and velocity computation using a single view from a rolling shutter camera. In *Computer Vision–ECCV 2006*, pages 56–68. Springer, 2006. 12, 13

[2] Simon Baker, Eric Bennett, Sing Bing Kang, and Richard Szeliski. Removing rolling shutter wobble. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2392–2399. IEEE, 2010. 13

[3] M Bigas, Enric Cabruja, Josep Forest, and Joaquim Salvi. Review of cmos image sensors. *Microelectronics journal*, 37(5):433–451, 2006. 6, 10

[4] Tomas Pajdla Cenek Albl, Zuzana Kukelova. R6p - rolling shutter absolute pose problem. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. IEEE, 2015. 7, 14

[5] Brian L. Evans Chao Jia. Probabilistic 3-d motion estimation for rolling shutter video rectification from visual and inertial measurements. In *MMSP*, pages 203–208, 2012. 13, 14

[6] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *Robotics & Automation Magazine, IEEE*, 13(2):99–110, 2006. 6

[7] Paul Furgale, Timothy D Barfoot, and Gabe Sibley. Continuous-time batch estimation using temporal basis functions. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2088–2095. IEEE, 2012. 13, 14

[8] Matthias Grundmann, Vivek Kwatra, Daniel Castro, and Irfan Essa. Calibration-free rolling shutter removal. In *Computational Photography (ICCP), 2012 IEEE International Conference on*, pages 1–8. IEEE, 2012. 13, 14

[9] Bert M Haralick, Chung-Nan Lee, Karsten Ottenberg, and Michael Nölle. Review and analysis of solutions of the three point perspective pose estimation problem. *International journal of computer vision*, 13(3):331–356, 1994. 15, 39, 51

[10] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003. 6, 14

[11] Michiel Hazewinkel. *Encyclopaedia of mathematics, supplement III*, volume 13. Springer Science & Business Media, 2001. 15

[12] Alexandre Karpenko, David Jacobs, Jongmin Baek, and Marc Levoy. Digital video stabilization and rolling shutter correction using gyroscopes. *CSTR*, 1:2, 2011. 13

[13] Georg Klein and David Murray. Parallel tracking and mapping on a camera phone. In *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*, pages 83–86. IEEE, 2009. 13

[14] Zuzana Kukelova, Martin Bujnak, and Tomas Pajdla. Automatic generator of minimal problem solvers. In *Computer Vision–ECCV 2008*, pages 302–315. Springer, 2008. 15

[15] Marci Meingast, Christopher Geyer, and Shankar Sastry. Geometric models of rolling-shutter cameras. *arXiv preprint cs/0503076*, 2005. 13

[16] Dokumentace objektu CaptureRequest [online]. Open Handset Alliance [cit. 8.5.2015]. Dostupné z:. https://developer.android.com/reference/android/hardware/camera2/CaptureRequest.html. 23

[17] Peyman Milanfary Ondřej Šindelář, Filip Šroubek. A smartphone application for removing handshake blur and compensating rolling shutter. In *Image Processing (ICIP), 2014 IEEE International Conference on*, pages 2160–2162. IEEE, 2014. 13, 14

[18] Dokumentace OpenCV [online]. OpenCV developer team [cit. 8.5.2015]. Dostupné z:. http://docs.opencv.org/index.html. 39

[19] Luc Oth, Paul Furgale, Laurent Kneip, and Roland Siegwart. Rolling shutter camera calibration. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013. 13, 14

[20] Erik Ringaby and Per-Erik Forssén. Efficient video rectification and stabilisation for cell-phones. *International Journal of Computer Vision*, 96(3):335–352, 2012. 13, 14, 39, 53

[21] Olivier Saurer, Kevin Koser, Jean-Yves Bouguet, and Marc Pollefeys. Rolling shutter stereo. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 465–472. IEEE, 2013. 13, 14

[22] Yoshiaki Shirai. *Three-dimensional computer vision*. Springer-verlag Berlin, 1987. 6

[23] Ken Shoemake. Animating rotation with quaternion curves. In *ACM SIGGRAPH computer graphics*, volume 19, pages 245–254. ACM, 1985. 13, 14

[24] Camera HAL v3 [online]. Open Handset Alliance [cit. 1.5.2015]. Dostupné z:. https://source.android.com/devices/camera/camera3.html, 2014. 20

[25] Nick Waltham. Ccd and cmos sensors. In *Observing Photons in Space*, pages 423–442. Springer, 2013. 10

[26] Eric W Weisstein. Fast fourier transform. 2015. 13

[27] Juyang Weng, Thomas S Huang, and Narendra Ahuja. *Motion and structure from image sequences*. Springer Publishing Company, Incorporated, 2012. 6

A. Testovací data

K práci jsou přiložena testovací data, která vznikla výše popsanými experimenty 5 a 6. Nachází se na DVD u jednotlivých skriptů, které je vyhodnocují.

A.1. Systémové časy

Data použitá k testování systémových časů jsou rozdělena podle využitého rozhraní pro ovládání kamery. První složka *Experimenty/Systémové časy/C-API* obsahuje adresář *Fotografie* a *Nahled* se soubory, které obsahují časy snímání a údaje ze senzorů. Snímky, ke kterým se údaje vztahují byly pořízeny v rozlišení 1920x1080 px pomocí první verze kamera API. Druhá složka *Experimenty/Systémové časy/C-API2* obsahuje testovací sady šablon v podsložkách s předponou *template*,*template2* a náhledů s předponou *preview*. Tyto data byly získány při využití druhé generace rozhraní pro ovládání kamery. V názvu složky po předponě *template* následuje označení použité šablony snímání a rozlišení fotografií. Předpona *template* se od *template2* odlišuje tím, že při ní byly snímky ukládány.

A.2. Částečná expozice

Experiment, při kterém byl snímán panel s diodami obsahuje několik testovacích sad ve složce *Experimenty/Částečná expozice - LED*. Všechny fotografie jsou odstíněny od okolního osvětlení, které je vytvořeno led diodami se stejnosměrným zdrojem. První část testovacích sad má předponu *parametr* a zkoumá vliv průběhu při změně rozlišení a délky svícení diody. Druhá část testovacích sad s předponu *brightness* byla focena při vypnutém vyvážení bílé barvy a různých intenzitách stejnosměrného osvětlení.

Fotografie zachycující efekt částečné expozice, který vznikl pomocí baterky s pulzně šířkovou modulací, jsou ve složce *Experimenty/Částečná expozice*. Složky testovacích sad mají název skládající se z předpony *PWM* a rozlišení pořízených snímků. V každé složce je adresář *images*, který obsahuje pořízené fotografie a adresář *data*, který obsahuje soubor *casy.csv* s časy pořízení snímku v nanosekundách. Stejná hierarchie složek je použita i dalších experimentů.

A.3. Rotace kamery

Snímky pořízené při rotaci kamery ve válci se nachází ve složce *Experimenty/Rotace kamery*. Každá testovací sada má vlastní složku, jejíž název se skládá z data pořízení, počtu sekund za které se kamera otočila o úhel 360° a rozlišení fotografií. Složky testovacích sad obsahují adresář *images*, který obsahuje pořízené fotografie a *data*, který obsahuje soubor *casy.csv* s časy pořízení snímku v nanosekundách.

A.4. Určení geometrie

Pozice kamery v třídimenzionálním prostoru je vypočtena z videí, které se nacházejí na DVD ve složce *Experimenty/Algoritmy*. Každé video je zaměřeno na jiný pohyb:

- **chuze.mp4** - video je pořízeno při chůzi okolo referenční značky,
- **pozice.mp4** - na videu jsou zachyceny snímky při rotaci kamery ve všech osách. Kamera je zapřená tak, aby neměnila svojí pozici oproti referenční značce,
- **rotace.mp4** - video je pořízeno při rotaci kamery okolo všech os,
- **rotacex.mp4** - video je pořízeno při rotaci kamery kolem osy x,
- **rotacey.mp4** - video je pořízeno při rotaci kamery okolo osy y.

Složka *Experimenty/Algoritmy (dvě znacky)* obsahuje videa, které obsahují 2 značky. Tyto videa sloužily k porovnání algoritmů P3P, P6P a PnP s algoritmem R6P. Každé video je zaměřeno na jiný pohyb:

- **r.mp4** - video je pořízeno při rotaci kamery okolo všech os,
- **rx.mp4** - video je pořízeno při rotaci kamery kolem osy x,
- **ry.mp4** - video je pořízeno při rotaci kamery okolo osy y.

Všechny videa byla pořízena v rozlišení 1920x1080 px a ve frekvenci 30 snímků/s. Složky s geometrií, reprojekční chybou a korespondencemi jednotlivých bodů se nacházejí ve stejném adresáři jako videa. Tyto složky se jmenují podle názvu videa a použitého algoritmu.

B. Knihovny pro vyhodnoceni experimentů

Tento dodatek shrnuje informace o napsaných skriptech a jejich nastavení a použití.

B.1. Práce s aplikacemi pro Android

Aplikace pro operační systém Android jsou na přiloženém DVD ve složce *Aplikace*. Obsahují vždy instalovatelnou variantu a zdrojové kódy. Následující podkapitoly vždy obsahují informaci o vývojovém prostředí, ve kterém byl projekt vytvořen, protože automatické systémy pro převod, nebo otevření v jiném vývojovém prostředí skončí téměř vždy chybou. Transformace mezi vývojovými prostředími lze pomocí vytvoření nové aplikace a překopírování zdrojů, ale Android Studio ve verzi 1.1 nepodporuje nativní kódy v C/C++ a nelze v něm z tohoto důvodu spustit nejrozsáhlejší aplikaci s OpenCV knihovnami.

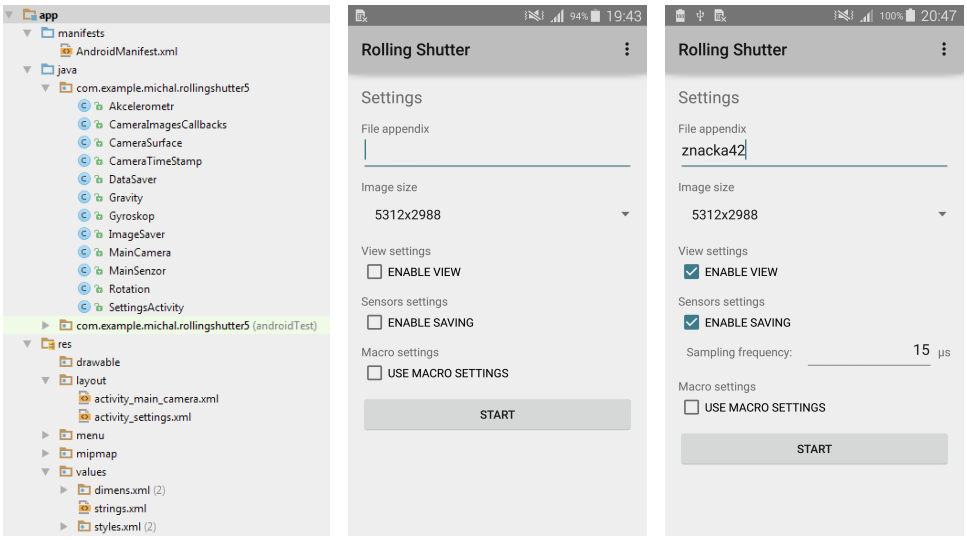
B.1.1. Android 3.0 - 4.4

První aplikace, která slouží k obstarání snímků pomocí C-API. Nachází se ve složce *Aplikace/Android C-API*. Tato aplikace běží na telefonech od Androidu 5.0 v režimu zpětné kompatibility. Obsahuje velice jednoduché grafické uživatelské rozhraní a většina funkcí se musí nastavit, případně odkomentovat, přímo v kódu. Kód je napsán ve vývojovém prostředí Android Studio verze 1.1. První položka uživatelského rozhraní je *"Start record previews"* a slouží ke spuštění náhledu. Fotografie se začnou fotit a ukládat po stisku pravé horní ikony fotoaparátu do složky *Pictures/RollingShutter/<datum>*. Pro ukončení focení pak slouží pravá spodní ikona. Časy snímání a data ze senzorů se ukládají do složky *Pictures/PreviewData/<datum>*. Druhá položka *"Start record video"* slouží pro jiný druh snímání a není v poskytnuté verzi využita. Poslední tlačítko *"Recover foto"* vyvolává kód pro převedení snímků do formátu *JPEG*. V kódu lze prohozením komentářů ukládat snímky ve formátu NV21, který poskytuje kamera a převádět je až po dokončení snímání.

B.1.2. Android 5.0 a vyšší

Aplikace využívající C-API2 je ve složce *Aplikace/Android C-API2*. Projekt je napsán ve vývojovém prostředí Android Studio verze 1.1. Aplikaci lze instalovat pouze na telefony s Androidem od verze 5.0. Tato aplikace byla optimalizována pro kapitolu 5 a umožňuje snímání fotek v různých rozlišení. Fotografie mohou být ukládány spolu s časy počátku a konce snímání a daty ze senzorů. Nastavení lze provádět přes uživatelské rozhraní, které je na obrázku B.3. Úvodní obrazovka je popsána pomocí XML formátu v souboru *res/layout/activity/settings.xml* a spravuje jí objekt *SettingsActivity*. Abstraktní třídu *MainSenzor*, která spravuje odchytávání dat z libovolného senzoru dědí třídy konkrétních senzorů *Akcelerometr*,*Gravity*,*Gyroskop* a *Rotation*. Tyto potomci slouží pouze k definici snímaného senzoru. Třída *DataSaver* implementuje rozhraní *Runnable* a obstarává ukládání dat ze senzorů a časů snímání v samostatném vlákně. Obdobně funguje i třída *ImageSaver*, která obstarává ukládání snímků ihned

po jejich pořízení. Data z jedné sekvence snímání jsou na rozdíl od předchozí aplikace uloženy do jedné složky. Ta obsahuje v názvu datum, příponu zadanou v uživatelském rozhraní a rozlišení fotografií. Časy snímání a údaje ze senzorů jsou ukládány do podsložky *data* a obrázky do podsložky *images*. Třída *CameraSurface* slouží k vykreslování náhledu a její nastavení je definované v *activity_main_camera.xml*. Informace o začátku snímání prvního řádku jsou poskytovány metodě objektu *CameraImagesCallbacks*. Posledním objektem je potomek *Activity* s názvem *MainCamera*. Tento objekt obsahuje implementaci základního postupu práce s kamerou, tak jak byl popsán v kapitole 4.3.

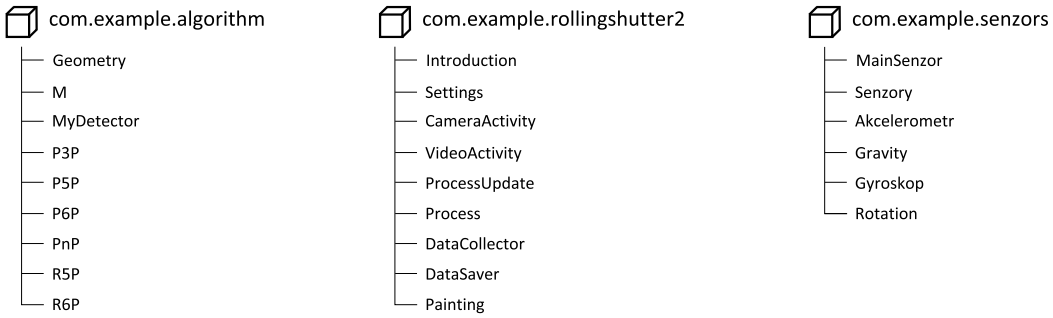


a) **b)** **c)**

Obrázek B.1. Obrázek **a)** ukazuje strukturu projektu a **b)**,**c)** grafické uživatelské rozhraní.

B.1.3. Android s využitím OpenCV

Poslední aplikace využívá OpenCV a Aruco knihovnu. Je napsaná ve vývojovém prostředí *Eclipse*. Pro spuštění projektu je potřeba doplnit podsložku *android-ndk-r10d* ve zdrojových kódech o knihovny potřebné pro běh nativního C++ kódu. Struktura projektu se dělí do třech balíčků.



Obrázek B.2. Obrázek ukazuje strukturu balíčků v aplikaci pro operační systém Android, která ovládá kameru pomocí OpenCV knihovny.

První z nich je balíček *algorithm*, který obsahuje všechny objekty obstarávající výpočty na snímcích. Konkrétně detektor *MyDetector*, pomocnou třídu pro výpočty *M*,

třídu obalující nalezenou geometrii *Geometry* a všechny použité algoritmy ve vlastních třídách. Druhý balíček *rollingshutter2* obsahuje řídící struktury uživatelského rozhraní, konkrétně třídu obstarávající focení, třídu pro extrahování snímků z videa, třídu pro vyhodnocení snímků, třídu obstarávající vykreslování korespondencí, třídu ukládající data, třídu s nastavením parametrů snímání a další pomocné třídy. Poslední balíček *senzors* obsahuje kódy spravující snímání dat ze senzorů. Tyto kódy jsou stejné jako u výše uvedené aplikace. Kód algoritmu R6P napsal v jazyce C++ kolega Čeněk Albl a nachází se ve složce *jni*. Následující výčet popisuje hlavní třídy a jejich možné nastavení pro první balíček obsahující algoritmy.

- **Geometry** je třída jejíž objekt uchovává souhrnné informace o geometrii jednoho snímku. Konkrétně kalibrační matici K, rotační matici R a posun kamery t.

- **M** je statická třída, do které jsou umístěny výpočty na maticích, vektorech a jejich převody z a do OpenCV formátu.

- **MyDetector** reprezentuje vylepšený detektor značek. V kódu jsou pod definicí třídy proměnné pro nastavení: převodu do černobílého snímku, rozlišení zmenšeného snímku a délky hrany markeru v metrech. Hlavní funkce *detect* vrací pole *double[2][n][3]*, jehož první rozměr rozděluje pole na body ve snímku a v prostoru, druhý rozměr udává jejich pořadí a třetí obsahuje jejich souřadnice x,y a z. Na zpřesnění pozic rohů je volána funkce *harrisDetector*, která určí velikost výřezů ve kterých jsou detekovány přesné pozice rohů. Na každém výřezu je detekován roh pomocí průsečíku dvou přímek, které prokládají dvě nejdelší hrany. Pokud selže tato přesná detekce, kterou reprezentuje funkce *findCorner*, je použit Harrisův detektor rohů z knihovny OpenCV.

- **P3P** reprezentuje vlastní implementaci algoritmu pro určení pozice a orientace kamery ze tří bodů. Algoritmus je napsán podle článku [9].

- **P5P,P6P,PnP** jsou třídy reprezentující algoritmy pro určení pozice a orientace kamery v prostoru, které využívají Lambert-Markvartovu iterativní optimalizační metodu, s inicializací pomocí P3P. V kódu je vybrán příslušný počet bodů a volána funkce *solvePnP* knihovny OpenCV.

- **R5P** je třída obsahující převody vstupů a výstupů do základních formátů se kterými pracuje JNI. Dále obsahuje funkce pro inicializaci a volání knihovny v jazyce C++.

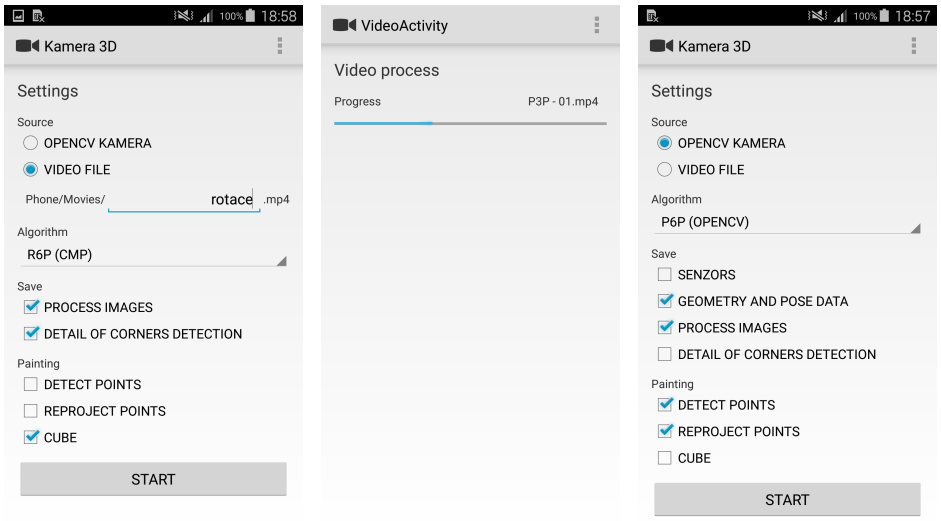
- **R6P** je třída, která obsahuje převody vstupů, výstupů a funkce pro komunikaci s nativními kódy, podobně jako třída R5P.

Následující výčet popisuje hlavní třídy a jejich možné nastavení pro druhý balíček obsahující řídící struktury.

- **Introduction** je třída, jejíž objekt spravuje úvodní grafické uživatelské rozhraní aplikace. Načítá nastavení zadané uživatelem do objektu třídy *Settings*. Dává na

výběr dvě hlavní možnosti zpracování snímků a to z kamery nebo videa.

- **Settings** reprezentuje nastavení vstupů a výstupů, vykreslování údajů do snímků, metodu získávání fotografií a použitý algoritmus při vyhodnocování.
- **CameraActivity** je třída která se stará o snímání fotografií pomocí OpenCV knihovny.
- **VideoActivity** je třída která se stará o získávání snímků z videa v samostatném vlákně.
- **ProcessUpdate** je objekt obalující informace o aktuálně prováděné operaci při vyhodnocování snímku.
- **Process** je statická třída, která vyhodnocuje jeden snímek na základě nastavení objektu třídy *Settings*.
- **DataCollector** je statická třída uchovávající informace o snímání, jako jsou například starty a konce snímání, časy trvání výpočtů, geometrie, korespondence a chyby reprojekce.
- **DataSaver** ukládá data ze senzorů a třídy *DataCollector* v samostatném vlákně.
- **Painting** je statická třída, která zajišťuje vykreslování krychle, detekovaných a promítnutých bodů ve snímku.



a) **b)** **c)**

Obrázek B.3. Obrázky **a)**,**b)** a **c)** ukazují náhled grafického uživatelské rozhraní aplikace.

Aplikace se skládá z nezávislých modulů a třídy jsou uspořádány tak, aby se kód neopakoval, ale bylo využíváno dědičnosti. Díky tomu lze jednotlivé části jednoduše měnit a nahrazovat je, bez ovlivnění chodu celé aplikace.

B.2. Částečná expozice

Pro vyhodnocení experimentů částečné expozice existují dvě podkapitoly. První se zaměřuje na zpracování dat z panelu deseti diod a druhá na zpracování fotografií pořízených při periodickém rozsvěcení a zhasínání intenzivního světelného zdroje. Pokud není uvedeno jinak, jsou skripty napsány v programu Matlab.

B.2.1. Panel s diodami

U toho experimentu jsou ve složce *Experimenty/Částečná expozice - LED* poskytnuty dva kódy. První s názvem *vyhodnoceni_detail.m* slouží pro vyhodnocení detailu jednoho snímku, ve kterém se v úvodní části nastaví složka a konkrétní snímek. Druhý kód *vyhodnoceni_vse.m* zpracovává všechny, v nastavení uvedené, testovací sady a ukládá výstup pro simulaci, která je sepsaná v programovacím jazyce Java. Simulace je provedena pomocí vygenerování událostí (rozsvícení/zhasnutí diody, začátek/konec expozice snímku a konkrétního řádku) v čase a jejich postupným vyhodnocením. Díky problémům popsaným v sekci 5.2 se nepovedlo průběhy synchronizovat. Pro synchronizaci je v simulaci zabudován algoritmus *RANSAC*. Simulace je přiložena jako projekt, vytvořený v prostředí Netbeans v podsložce *ModelLED*.

B.2.2. Periodické rozsvěcení zdroje

Skript *vyhodnoceni.m* vyhodnocující počet rozsvícených a zhaslých řádků nachází se ve složce *Experimenty/Částečná expozice*. Skládá se ze čtyřech částí. První část slouží pro nastavení složek s testovacími daty a časů, jak dlouho světelný zdroj svítil a jak dlouho byl vypnutý v jedné periodě. Druhá část slouží k vytvoření průměrných snímku podle článku [20]. Průměrné snímky nakonec nebyly použity, protože při tomto experimentu nezvyšovali přesnost. Jejich použití je zakomentováno na řádku 55. Třetí část vyčítá z každého sloupce, každého snímku, počty řádků mezi světlými a tmavými pruhy. Poslední, čtvrtá část slouží k filtraci špatně změřených údajů, jejich průměrování a výpočtu času vyčítání jedné řádky.

B.3. Rotace kamery

Pro vyhodnocení údajů ze snímků pořízených při rotaci kamery byly sepsány 4 skripty v Matlabu a jedna simulace v jazyce Java. Vše je umístěno ve složce *Experimenty/Rotace kamery*. Společné parametry k testovacím sadám načítá skript *nacti_parametry.m*, který obsahuje pro každou testovací sadu: název její složky, čas za který se kamera otočí o 360°, poslední snímek před začátkem otáčení, poslední snímek, kdy se kamera otáčela a kalibrační matici.

B.3.1. Zkosení čar

Zkosení čar vyhodnocuje skript *vyhodnoceni_zkoseni_car.m*. Jeho první část načte parametry. Druhá detekuje konce čar funkcí *detekujKonceCar* a spočte příslušný úhel, funkce *spoctiUhly2*. Třetí část filtruje špatně určené úhly odstraněním hodnot mimo směrodatnou odchylku, počítá dobu vyčítání řádky a tiskne vyhodnocené údaje.

B.3.2. Šířka čar

Skript *vyhodnoceni_sirky_cary.m* porovnává simulovanou a reálnou šířku čáry, které vznikly rotací kamery. Úvodní nastavení umožňuje výběr testovací sady, jaký řádek bude vybrán pro porovnávání a s jak velkým okolím středu čáry se bude pracovat. Následně je ze vzorového snímku testovací sady vybrána jedna čára a její vektor uložen do pomocného souboru. Na obrazovku jsou vytisknuty údaje o prvním rotujícím snímku potřebné pro běh simulace, kterými je doba expozice, čas jedné otáčky a úhel otočení o jeden pixel. Simulace je napsaná v jazyce Java a její projekt z vývojového prostředí Netbeans je ve složce *ModelCar*. Po vypsání údajů je skript pozastaven a čeká na dokončení simulace. Do kódu simulace je potřeba doplnit vytištěné údaje. Kód rotuje vektorem čáry po dobu jedné expozice a výstupní vektor uloží do druhého pomocného souboru. Skript v Matlabu pomocný soubor načte, zarovná průměrný střed obou čar a vytiskne jejich průběhy. Poslední část použije vzorce ze sekce 5.3.3 pro výpočet doby expozice.

B.3.3. Výpočet doby expozice

Stejný kód počítající dobu expozice jedné čáry v jednom snímku, který je uvedený na konci *vyhodnoceni_sirky_cary.m*, je použit pro vyhodnocení průměrných hodnot pro každý snímek ve skriptu *vyhodnoceni_expozice.m*. První část tohoto skriptu načítá parametry. V druhé části kódu jsou na každém snímku nalezeny konce čar, stejně jako u výpočtu zkosení. Konce jsou pak interpolovány a pro každý řádek mezi nimi je vybrána výseč, na které je počítán jas pozadí, jas čáry a její šířka, pomocí funkce *sirka_a_jas*. Tyto údaje jsou pro jeden snímek průměrovány v následující části, kde je zároveň počítaná, na řádku 100, odchylka času expozice pomocí obou uvedených vzorců v kapitole 5.3.3. Hodnoty pro snímky jsou v poslední, třetí části průměrovány a vypsány pro testovací sady.

C. Obsah přiloženého DVD

Kořenový adresář DVD

Diplomová práce.pdf – Výsledný PDF dokument.

Diplomová práce.zip – Zdrojové soubory výsledného PDF dokumentu.

Fotografie práce – Fotografie, na které je odkazováno v práci.

Aplikace – Složka aplikací pro operační systém Android.

Android C-API – Aplikace pracující s C-API.

Android C-API2 – Aplikace pracující s C-API2.

Android OpenCV – Aplikace využívající knihovnu OpenCV.

Experimenty – Složka s testovacími daty a vyhodnocujícími skripty.

Algoritmy – Testovací videa pro výpočet polohy kamery.

Algoritmy (dvě znacky) – Testovací videa pro výpočet polohy kamery.

Částečná expozice – Skripty a testovací data částečné expozice.

Částečná expozice - LED – Skripty a testovací data částečné expozice.

Rotace kamery – Skripty a testovací data vyhodnocující deformace obrazu.

Systémové časy – Skripty a testovací data vyhodnocující systémové časy.

Ikony použité na této stránce jsou ze stránky Gnome.org.