Knihovna pro automatickou kalibraci prostoru v obrazu z kamery

Bc. Richard Fous

Diplomová práce
2021

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav počítačových a komunikačních systémů

Akademický rok: 2020/2021

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Richard Fous**
Osobní číslo:	**A19444**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Počítačové a komunikační systémy**
Forma studia:	**Prezenční**
Téma práce:	**Knihovna pro automatickou kalibraci prostoru v obrazu z kamery**
Téma práce anglicky:	**An Automatic Space Calibration Library Based on Camera View**

Zásady pro vypracování

1. Seznamte se s problematikou počítačového zpracování obrazu se zaměřením na knihovnu OpenCV
2. Prostudujte možnosti kalibrace obrazu z kamery
3. Vhodným způsobem definujte požadavky na výslednou knihovnu
4. Sestavte architekturu navrhované knihovny a definujte proces zpracování obrazu v rámci knihovny
5. Implementujte knihovnu na základě definovaných požadavků a sestavené struktury
6. Sestavte testovací aplikaci a ověřte funkčnost knihovny

Forma zpracování diplomové práce: **Tištěná/elektronická**

Seznam doporučené literatury:

1. KAEHLER, Adrian a Gary R. BRADSKI. Learning OpenCV 3: computer vision in C++ with the OpenCV library. Sebastopol: O'Reilly, 2016. ISBN 9781491937990.
2. LISCHNER, Ray. Exploring C++20: The Programmer's Introduction to C++. New York: Apress, 2020. ISBN 9781484259603.
3. BAGGIO, Shervin EMAMI, David Millán ESCRIVÁ a Khvedchenia IEVGEN. Mastering OpenCV with Practical Computer Vision Projects. Birmingham: Packt Publishing, 2012. ISBN 9781849517829.
4. DMITROVIĆ, Slobodan. Modern C++ for Absolute Beginners: A Friendly Introduction to C++ Programming Language and C++11 to C++20 Standards. New York: Apress, 2020. ISBN 9781484260463.
5. LAFORE, Robert. Object-Oriented Programming in C++. 4th ed. Carmel, Indiana: CourseSams Publishing, 2001. ISBN 9780672323089.
6. ENG, Lee Zhi. Hands-On GUI Programming with C++ and Qt5 by Lee Zhi Eng. Birmingham: Packt Publishing, 2018. ISBN 9781788397827.

Vedoucí diplomové práce: **Ing. Peter Janků, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **15. ledna 2021**
Termín odevzdání diplomové práce: **17. května 2021**

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

Ing. Miroslav Matýsek, Ph.D. v.r.
ředitel ústavu

Ve Zlíně dne 15. ledna 2021

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 25. 5. 2021

RICHARD FOUS, v.r.
podpis studenta

ABSTRAKT

Cílem této diplomové práce je navrhnout, vytvořit a implementovat knihovnu pro kalibraci prostoru v obrazu z kamery, zejména v rámci sportovních utkání. Pomocí kalibrace prostoru je definována oblast daného sportovního hřiště v obraze. Zkalibrovaný prostor je poté možné využít pro následnou práci s obrazem, jako například perspektivní kreslení. Po vytvoření této knihovny je požadováno vytvoření aplikace, která bude sloužit pro testování a předvádění funkčnosti vytvořené knihovny. Knihovna bude postavena na základě volně dostupné knihovny OpenCV, která patří mezi hojně využívané knihovny určené pro zpracování obrazu. V teoretické části této práce jsou popsány jednotlivé prvky, které byly využity pro vývoj této knihovny. V praktické části se nachází popis vytvořené knihovny spolu s popisem testovací aplikace.

Klíčová slova: Kalibrace, OpenCV, C++, Qt, Open source

ABSTRACT

The aim of this thesis is to design, develop and implement a library for a space calibration based on camera view, particularly in the context of sports events. With the help of space calibration, the area of the given sports field in the image is defined. The calibrated space can then be used for subsequent work with the image, such as perspective drawing. After creating this library, it is required to create an application that will be used to test and demonstrate the functionality of the created library. The library will be built on the basis of freely available OpenCV libraries, which are among the widely used libraries designed for image processing. The theoretical part of this work describes the individual elements that were used for the development of this library. In the practical part there is a description of the created library together with a description of the testing application.

Keywords: Calibration, OpenCV, C++, Qt, Open source

Chtěl bych zde poděkovat vedoucímu práce panu Ing. Peteru Janků, Ph.D. za cenné rady a za odbornou pomoc při kompletaci této práce. Děkuji také své rodině za podporu a veškerou pomoc v průběhu celého studia a při psaní této diplomové práce. Zároveň děkuji také všem svým kolegům a vedení z firmy Daite s.r.o za odborné rady při vytváření knihovny a testovací aplikace.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Pořizování fotografií či videí pomocí digitálních zařízení je v dnešní době velmi populární a dostupné téměř pro všechny. Výhodou takto pořízeného záznamu je, že může být do nekonečna upravován a pozměňován, díky různým metodám zpracování obrazu. V posledních letech vzniklo obrovské množství nových technologií určených pro zpracování obrazu a to díky neustále se rozšiřujícím schopnostem dostupné techniky. S digitálním obrazem pracují i jiná odvětví informačních technologií, jako například oblast počítačového vidění, které se zabývá porozuměním obsahu digitálního obrazu. Tato oblast čím dál více nabírá na popularitě, jelikož produkty, které pracují s počítačovým viděním se v dnešní době nacházejí v mnoha zařízeních, které se běžně vyskytují v našem okolí.

V dnešní digitální době je problematické zvládnutí techniky kreslení objektů do výstupního obrazu kamery. Techniku kreslení lze uplatnit v širokém spektru odvětví, ale velice důležitá je při různých sportovních utkáních, kde toto kreslení může pomoci při odhalování prohřešků v příslušném zápase. Aby bylo kreslení do obrazu přesné a spolehlivé, je nejdříve nutné provést prostorovou kalibraci, při které je ve vstupním obraze definována pozice daného sportovního hřiště.

Tato diplomová práce se zabývá návrhem a implementací knihovny pro automatickou kalibraci prostoru v obrazu z kamery a pro kreslení objektů do zkalibrovaného obrazu, zejména v rámci sportovních utkání. Výsledná knihovna bude postavena na základech volně dostupné knihovny OpenCV, která obsahuje velké množství různých funkcí. Tyto funkce napomáhají běžným uživatelům okusit oblast počítačového vidění a usnadňují jim práci s jednotlivými aspekty této oblasti. Pro kontrolu funkčnosti této knihovny bude dále v rámci této diplomové práce navrhnuta a implementována testovací aplikace. V teoretické části práce je nastíněna problematika jednotlivých částí, které jsou potřebné k realizaci knihovny. V praktické části je poté popsán postup návrhu a implementace knihovny spolu s popisem vytvořené testovací aplikace.

I. TEORETICKÁ ČÁST

1 Počítačové vidění

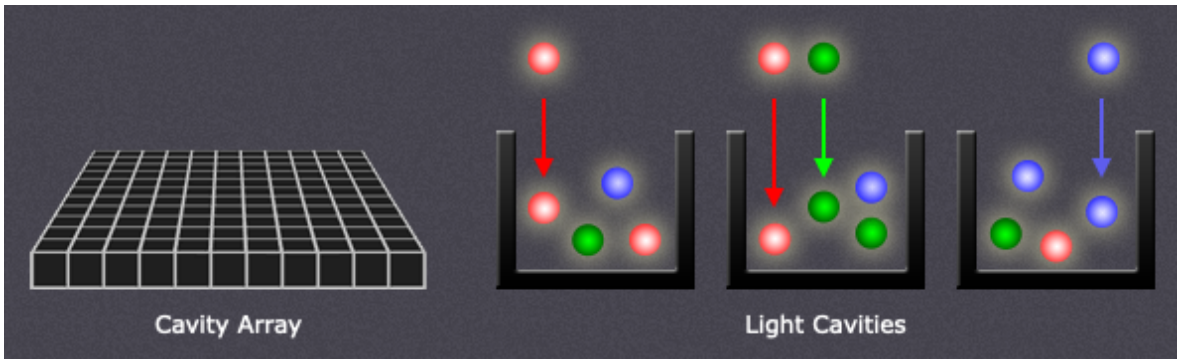
Pojmem počítačové vidění je označován obor výpočetní techniky, který se zabývá vysoko úrovňovým porozuměním digitálního obrazu či videa. Výzkumníci v oboru počítačového vidění se intenzivně zabývají výzkumem matematických modelů, které by sloužily pro rozpoznání trojrozměrných objektů v digitálním obraze. V dnešní době již existují technologie, pomocí kterých jsme schopni přesně vypočítat částečný 3D model prostředí z několika částečně se překrývajících fotografií. Tyto technologie dále umožňují například analyzovat pohyb člověka v obraze a také částečně analyzovat a identifikovat jednotlivé osoby, například pomocí rozpoznání obličeje. I když jsou tyto technologie na velmi vysoké úrovni, stále nejsou dostatečně rozšířené na to, aby byly schopny samy rozpoznat všechny prvky v obrázku tak, jak to dokáží lidé. Řešení tohoto problému je velice složité, jelikož se jedná o inverzní problém, při kterém se snažíme z obrazu získat nějaké neznámé prvky na základě neúplných vstupních informacích. Je tedy nutné zvolit metody, které jsou založené na fyzikálních a pravděpodobnostních modelech, pomocí kterých je možné zjistit několik přibližných řešení problému a z těchto řešení je poté vybráno jedno nejlepší. [1]

Jelikož je v dnešní době počítačové vidění velice rozšířeným oborem, existuje plno technologií, které se zabývají specifickými částmi tohoto oboru a jsou využívány v reálných aplikacích, jako například [1]:

- **Optické rozpoznávání znaků (OCR) :** slouží pro rozpoznávání ručně psaných znaků, ale své využití nalézá také např. u čtení poznávacích značek automobilů (ANPR)
- **Výrobní kontrola :** technologie, které jsou schopny rozpoznat a měřit rozdíly mezi vyprodukovanými díly ve výrobě. Slouží například pro kontrolu defektů při výrobě součástek pro letadla či automobily.
- **3D modelování budov :** tuto technologii využívají například Google mapy, které jsou schopny ze satelitních fotografií vytvořit 3D modely budov.
- **Využití v automobilech :** existují zde technologie, které slouží ke zvýšení bezpečnosti provozu, jako například detekce objektů a chodců v případech, kdy ostatní technologie, jako například radary, které nejsou z důvodu špatných podmínek schopny vyhodnotit situaci. Tato detekce poté slouží například pro systém automatického brždění. Počítačové vidění zde nachází uplatnění i u autonomního řízení.

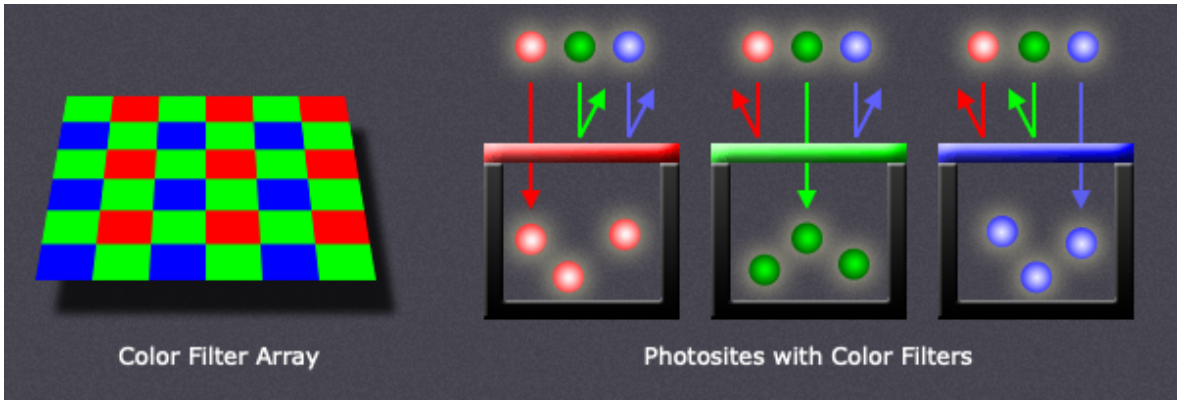
1.1 Princip digitální kamery

Základním stavebním kamenem počítačového vidění jsou digitální kamery, které umožňují zaznamenávat obraz tak, aby byl čitelný pomocí počítače. Tyto moderní kamery využívají snímač, který má ve většině případů tvar obdélníku, na který dopadají paprsky světla. Tyto snímače se skládají z několika miliónů tzv. dutin (Obr. 1.1), které pohlcují paprsky světla. V momentě, kdy dojde k otevření uzávěrky kamery, začíná tzv. expozice, při které jsou jednotlivé dutiny otevřeny a začínají pohlcovat fotony, které jsou ukládány ve formě elektrického signálu. Po dokončení expozice jsou všechny tyto dutiny uzavřeny a kamera poté pomocí měření intenzity elektrického signálu zjišťuje, kolik fotonů spadlo do každé dutiny. Tyto elektrické signály jsou poté kvantifikovány jako digitální hodnoty, jejichž rozsah je určen bitovou hloubkou. Bitová hloubka určuje, kolik bitů má barva jednoho pixelu. Jednotlivé dutiny se na senzoru nenachází v těsné blízkosti vedle sebe, ale jsou mezi nimi určité malé mezery, které jsou využívány pro ostatní elektroniku, která je v kameře potřebná. Tyto dutiny mohou například pokrývat pouze polovinu celkového prostoru senzoru. Aby nedocházelo k odrážení fotonů a tím pádem ke ztrátě informace, je každá mezera pokryta mikročočkou, která slouží k nasměrování fotonů do dané dutiny. Každá z těchto mikročoček je schopna zvýšit intenzitu fotonu v dané dutině a tím je možné dosáhnout lepší kvality obrazu pro stejně dlouhou dobu expozice. [2]



Obr. 1.1 Kamerový senzor složený z dutin [2]

Na obrázku 1.1 je možné vidět, jsou do dutin zachycovány jednotlivé fotony. Při využití takového senzoru by ale byl výsledný obrázek černobílý, jelikož jednotlivé dutiny nejsou schopny rozpoznat, jaké množství daného barevného spektra se v nich nachází. Aby bylo možné zachytit jednotlivé barvy, využívá se zde pole barevných filtrů, které propouští pouze danou barvu a ostatní jsou odraženy. Pro každou dutinu je tedy nutné aproximovat zbývající dvě barvy, aby každý pixel obsahoval kompletní barevné spektrum. Nejznámější a zároveň nejvyužívanějším typem tohoto filtrového pole je tzv. Bayerovo pole (Obr. 1.2). [2]



Obr. 1.2 Bayerovo pole [2]

V Bayerově poli se na jednotlivých řádcích střídají zeleno-červené a zeleno modré filtry. Nejvíce zastoupeným filtrem v tomto poli je zelený, který zabírá dvakrát více plochy než ostatní barvy dohromady. Toto je dáno faktem, že lidské oko je více senzitivní na zelené světlo, což ve výsledném obrázku snižuje velikost šumu. [2]

K získání výsledného obrázku z daného pole filtrů se využívá proces demosaicing. V dnešní době je dostupné nespočetné množství těchto procesů, všechny ale vychází z původního Bayerova procesu. Jelikož má každá dutina pouze jednu barvu filtru, není kamera schopna změřit hodnoty ostatních barev. K zjištění výsledné barvy daného pixelu se tedy používají hodnoty ostatních barev z vedlejších dutin. Celé pole je rozděleno na menší pole o velikosti 2x2 dutin. Tímto rozdělením je poté možné získat výslednou plnou barvu pixelu, ale zároveň tím dojde ke dvojnásobnému snížení počtu pixelů, jelikož nyní každý pixel výsledného obrazu využívá 4 dutiny. Tento problém je vyřešen překrýváním jednotlivých menších polí. [2]

1.2 Barevné modely

Barevné modely slouží pro matematickou reprezentaci barev, kde každý model definuje barvy dle přesně specifikovaných barevných nebo jiných komponent. Barvy, které jsou definované pomocí stejného barevného modelu, nesou název barevný prostor. Jednotlivé barevné modely mohou být dle jejich vlastností rozděleny do dvou základních kategorií [3].

- **Modely závislé na zařízení:** u těchto modelů je výsledná barva závislá na daném zpracovávacím a zobrazovacím zařízení. Každé zařízení obsahuje svou vlastní reprezentaci barev, a proto se na různých zařízeních mohou barvy s totožnými hodnotami lišit. Mezi tyto modely patří např. model RGB nebo HSB [3].
- **Modely nezávislé na zařízení:** tyto modely nejsou závislé na výstupním nebo

zobrazovacím zařízení, a tedy na různých zařízeních se budou barvy se stejnými parametry zobrazovat stejným způsobem. Tento barevný model je vhodný například pro přenos obrazových dat pomocí internetu, jelikož při cestě musí procházet přes různá zařízení s různými parametry. Popis barev u těchto modelů probíhá pomocí 3 atributů, které odpovídají lidskému vnímání barev. Mezi tyto metody patří například CIELAB [3].

1.2.1 RGB

Jedná se o barevný model, který využívá komponenty červené (R), zelené (G) a modré (B) k definování množství daných základních barev ve výsledné barvě. Barevný rozsah je zde specifikován pomocí jednotlivých bitů. Například u 24-bitového obrázku může mít každá komponenta RGB hodnotu od 0 do 255. Se zvyšujícím se počtem bitů roste rozsah možných hodnot jednotlivých komponent. Tyto hodnoty značí, kolik světla prochází v daném barevném spektru. Kombinacemi různých hodnot těchto komponent poté vznikají různé barvy. V případě, že jsou všechny 3 hodnoty nastaveny na maximum, jeví se výsledná barva jako bílá. Teoreticky se zde stále nacházejí všechny 3 barvy, ale pixely zobrazující dané barvy se nacházejí v takové blízkosti, že lidské oko není schopno tyto 3 barvy rozpoznat. Pokud jsou všechny 3 komponenty RGB nataveny na hodnotu 0, jeví se výsledný pixel jako černý, jelikož světlo neprochází žádnou z komponent [3] [4].

1.2.2 Grayscale

Grayscale model využívá pouze jednu komponentu a to světlost, která je znázorněna hodnotami 0 až 255. Hodnota 0 značí černou barvu a hodnota 255 bílou, mezi těmito hodnotami se poté nachází různé odstíny šedi [4].

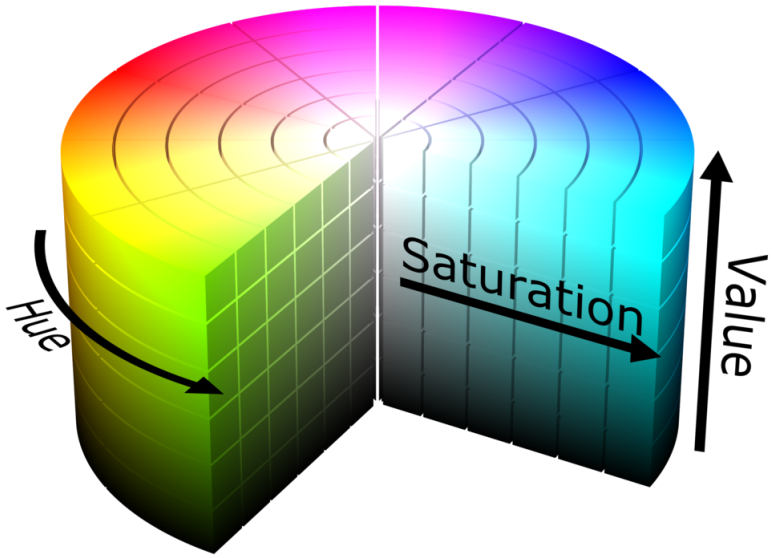
1.2.3 RGBA

RGBA obsahuje stejné komponenty jako RGB model s tím rozdílem, že je zde navíc obsažena i čtvrtá komponenta, která obsahuje informace o průhlednosti daného pixelu. Pokud má tato komponenta hodnotu 0, znamená to, že daný pixel je plně průhledný, naopak má-li hodnotu 255, je pixel plně viditelný [5].

Z RGBA obrázku je možné získat tzv. alfa kanál, což je ve své podstatě grayscale obrázek, který kopíruje tvar neprůhledných pixelů z původního obrázku. Tento grayscale obrázek je také nazýván jako mapa, která může sloužit například při spojování RGBA obrázku s RGB obrázkem [5].

1.2.4 HSB

Tento barevný model využívá odstín (H), sytost (S) a jas (B) jako komponenty pro popis jednotlivých barev. Model HSB (Obr. 1.3) je také známý pod názvem HSV, kdy komponenta V značí hodnotu (value). Sytost zde popisuje pigment barvy a je vyjádřena pomocí stupňů, které reprezentují pozici na standardním barevném kole. Například červená barva je 0° , zelená na 120° a modrá na 240°. Sytost barvy značí příměs jiné barvy, bývá také označována jako čistota barvy a představuje množství šedé příměsi v poměru k odstínu. Sytost je popisována v procentech, kde 100% představuje plně sytou barvu a 0% představuje šedou barvu. Komponenta jasu určuje množství bílého světla v barvě a stejně jako u sytosti je jas popsán v procentech a se zvyšující se hodnotou se zvyšuje jasnost barvy. Pokud je jas nastaven na 0% je výsledná barva černá [3] [6].



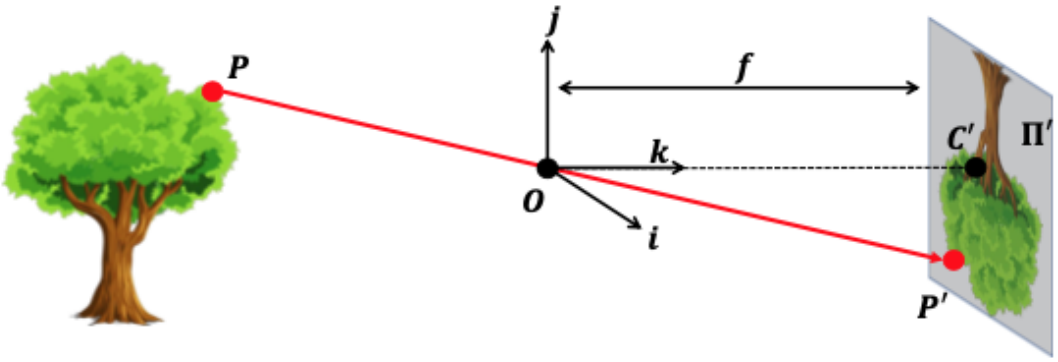
Obr. 1.3 Grafické zobrazení HSV [6]

1.3 Modelace obrazu

Modelace obrazu je proces, při kterém dochází k zobrazování 3D scény na 2D plochu. Odražené světlo od 3D modelu je zachycováno pomocí snímacího zařízení na 2D plochu. Za 2D plochu se zde považuje fotografický film, nebo kamerový senzor. Každý bod daného 3D objektu odráží světelné paprsky do několika směrů, což způsobuje, že každý bod na záznamovém zařízení je tedy ovlivňován více body ve 3D prostoru, což má za následek rozostřený obraz, který bude těžce čitelný. Je tedy nutné nějakým způsobem omezit většinu odražených paprsků a zachytit pouze požadované paprsky. Tohoto omezení je možno dosáhnout několika způsoby [8].

1.3.1 Dírková komora (Pinhole camera)

Jedná se o historicky první způsob modelace obrazu. K omezení odražených paprsků ze 3D objektu je zde využita deska s velmi malou dírkou, také nazývaná jako clona. Díky této cloně dochází k zachycení pouze některých světelných paprsků a ostatní jsou odraženy. V ideálním případě má clona velikost jednoho bodu, což umožňuje propustit pouze jeden světelný paprsek daného 3D bodu na 2D plochu. V reálném světě není možné takovou ideální clonu vyrobit, a proto i ta nejmenší clona propustí více paprsků v daném úhlu [8] [7] .



Obr. 1.4 Parametry dírkové komory [8]

Na obrázku 1.4 je možné pozorovat jednotlivé parametry dírkové komory. Clona je zde označována jako bod O nebo také jako střed kamery. Vzdálenost mezi clonou a snímacím zařízením je označována jako ohnisková vzdálenost f. Jednotlivé body 3D objektu jsou zde označeny jako $P = \begin{bmatrix} x & y & z \end{bmatrix}^T$. Tyto body jsou mapovány na obrazovou rovinu Π', kde vznikají body $P' = \begin{bmatrix} x' & y' \end{bmatrix}^T$. Stejně tak samotná clona může být namapována na obrazovou rovinu, kde vznikne nový bod C' [8].

Na středu dírkové komory je definován souřadnicový systém $\begin{bmatrix} i & j & k \end{bmatrix}$, který bývá označován jako referenční systém kamery nebo také souřadnicový systém kamery. V tomto souřadnicovém systému je osa k kolmá k obrazové rovině, osy i a j slouží k určení pozice clony vůči obrazové rovině. Ve většině případů se clona nachází uprostřed obrazové roviny. Úsečka, která vznikne spojením bodů C' a O se nazývá optická osa. Díky této úsečce vznikne v souřadnicovém systému trojúhelník s vrcholy P' C' O. Tento trojúhelník je podobný trojúhelníku, který je složený z bodů P, O a (0,0,z). U těchto trojúhelníků je možné využít věty o podobnosti trojúhelníků, která je zde využita pro zjištění pozice obrazových bodů [8]:

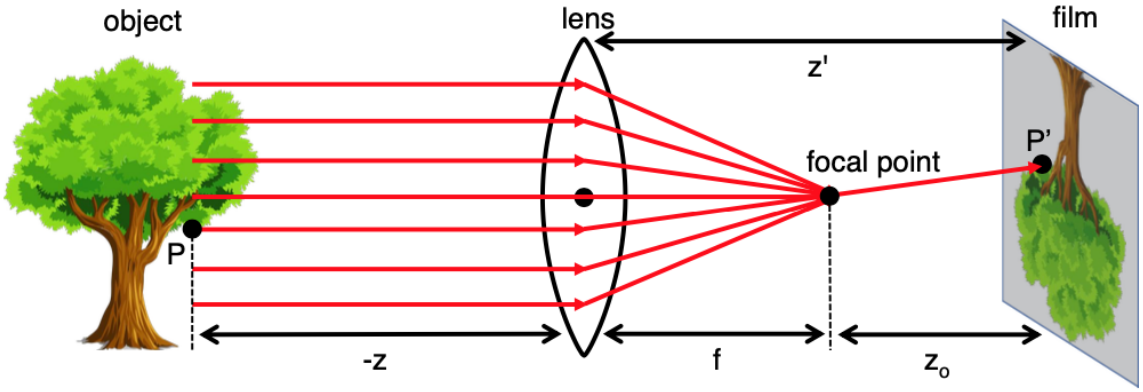
$$P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} f\frac{x}{z} \\ f\frac{y}{z} \end{bmatrix} \tag{1.1}$$

Koncept dírkové kamery vychází z faktu, že ideální clona má velikost pouze jednoho bodu, který je nekonečně malý. Při reálném využití není možné takovou velikost clony využít. Je tedy potřeba počítat s různými velikostmi této clony. Při zvětšování velikosti clony dopadají paprsky na snímací zařízení z více 3D bodů, což způsobuje, že výsledný obrázek je poté rozostřený. Naopak při zmenšení velikosti clony dopadá na snímací zařízení méně paprsků a výsledný obrázek je ostřejší, což má ale za následek, že výsledný obrázek je tmavý a nemusí být čitelný. Toto je jeden ze zásadních nedostatků tohoto kamerového modelu [8].

1.3.2 Optické čočky

U moderních kamer jsou problémy s ostrostí a jasem, které se nacházely u dírkové komory vyřešeny pomocí optické čočky. V případě, že je čočka správně umístěna a je zvolena správná velikost, tak všechny 3D body označeny jako P, jsou všechny světelné paprsky z tohoto bodu lomeny tak, že všechny směřují pouze do jednoho bodu P', který se nachází na obrazové rovině. Tím je tedy vyřešen problém s jasem a ostrostí, jelikož zde oproti dírkové komoře, do jednoho bodu na obrazové rovině, směřuje více odražených paprsků. Ovšem tento fakt platí pouze pro 3D body, které jsou ve stejné vzdálenosti od čočky. Pokud se jiný bod ve 3D prostoru nachází v jiné vzdálenosti od obrazové roviny něž původní bod, pro který byla čočka nastavena, dochází zde opět k rozostření obrazu. Paprsky tohoto bodu jsou totiž kvůli jiné vzdálenosti od čočky lomeny jinak a místo toho, aby dopadaly na obrazovou rovinu do jednoho bodu, dopadají na více bodů. Tomuto fenoménu se říká hloubka ostrosti, což je efektivní vzdálenost, na které jsou kamery schopny pořizovat čisté fotografie [8] [1] [7].

Na obrázku 1.5 je znázorněna funkčnost jednoduché optické čočky. Paprsky světla, které jsou odražené od jednotlivých 3D bodů P, jsou pomocí čočky lámány. Optické čočky využívané v kamerách mají jednu zásadní vlastnost, a to takovou, že směřují všechny paprsky, které jsou paralelní k optické ose, do jednoho bodu, kterému se říká ohnisko. Vzdálenost tohoto bodu od středu čočky se nazývá ohnisková vzdálenost čočky f. S rostoucí ohniskovou vzdáleností se zmenšuje zorné pole objektivu, naopak se snižující se ohniskovou vzdáleností se zorné pole rozšiřuje. Světelné paprsky, které prochází středem optické čočky nejsou lámány ani odchylovány, díky tomu je možné pro popis mapování jednotlivých bodů využít poupravenou rovnici dírkové komory [8]:



Obr. 1.5 Parametry optické čočky [8]

$$P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} z' \frac{x}{z} \\ z' \frac{y}{z} \end{bmatrix} \tag{1.2}$$

Celková ohnisková vzdálenost je zde vypočítána jako součet vzdálenosti středu čočky k ohnisku a vzdálenosti ohniska od obrazové roviny $z' = f + z_0$ [8].

Výše uvedený popis je založen na nejjednodušším modelu jedné tenké čočky. Při výrobě dnešních objektivů je využíváno větší množství čoček, což umožňuje např. vyrobit objektiv s velkou ohniskovou vzdáleností, ale o malých rozměrech. Jelikož každá čočka láme vstupní paprsky jiným způsobem a směrem, jsou jednotlivé parametry těchto složitějších objektivů vždy vypočítávány v závislosti na počtu a typu použitých čoček [8].

1.4 Převod do digitálního obrazového prostoru

Každý P bod ve 3D obraze může být namapován na 2D bod P', který se nachází na obrazové rovině Π'. Tomuto mapování z R^3 na R^2 se říká projektivní transformace. Tato projekce 3D bodů na obrazovou rovinu přímo neodpovídá tomu, co je možné pozorovat na výsledném obrázku, a to z několika důvodů. Prvním z těchto důvodů je, že ve většině případů jsou body digitálního obrazu popsány v jiném referenčním systému, než obrazové body. Druhým důvodem je, že digitální obrázky jsou rozděleny do několika diskrétních pixelů, ale body v obrazové rovině jsou spojité. Posledním důvodem je fakt, že samotné senzory mohou do mapovacího procesu vložit nějakou nelineární chybu, jako například zkreslení mapování. Aby bylo možné tyto problémy eliminovat, je nutné do procesu mapování vložit několik dalších transformací, které poté umožní mapování jakéhokoliv bodu ze 3D do pixelových souřadnic [8].

1.4.1 Matice kamery

Matice kamery popisuje sadu parametrů, které ovlivňují, jakým způsobem jsou 3D body obrazu P mapovány na body P' v obrazové rovině. Tyto jednotlivé parametry se zde nacházejí ve formě matice. Mezi tyto nové parametry patří c_x a c_y, které popisují, jakým způsobem jsou jednotlivé 3D a obrazové body vůči sobě posunuty. Obrazové body mají počátek v bode C', který se většinou nachází uprostřed obrazové roviny. 3D body mají ve většině případů počátek v levém spodním bodě obrazu. Tedy 2D body v obrazové rovině a body v obraze mají mezí sebou daný posun, který je definován pomocí translačního vektoru $\begin{bmatrix} c_x & c_y \end{bmatrix}^T$. Aby byl tento posun vykompenzován, je zapotřebí upravit mapovací matici [8] [1]:

$$P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} f\frac{x}{z} + c_x \\ f\frac{y}{z} + c_y \end{bmatrix} \tag{1.3}$$

Další problém u mapování představují rozdílné jednotky bodů. Zatímco body v digitálním obraze jsou definovány pomocí pixelů, body v mapovacím obraze jsou definovány pomocí fyzikálních jednotek (např. centimetry). Aby bylo možné zakomponovat tyto rozdílné jednotky do mapovací matice, je zapotřebí využít nových parametrů k a l. Tyto parametry, jejichž jednotka bude například $\frac{pixely}{cm}$, korespondují se změnou jednotek u obou os na obrazové rovině. Oba parametry mohou mít rozdílné hodnoty, jelikož je možné, že poměr stran daného pixelu nemusí být roven jedné. Pokud se $k = l$, tak se tomuto pixelu říká čtvercový pixel. Na základě těchto nových koeficientů je nutné upravit stávající mapovací matici [8]:

$$P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} fk\frac{x}{z} + c_x \\ fl\frac{y}{z} + c_y \end{bmatrix} = \begin{bmatrix} \alpha\frac{x}{z} + c_x \\ \beta\frac{y}{z} + c_y \end{bmatrix} \tag{1.4}$$

V případě, že by výše uvedená projekce byla lineární transformací, bylo by možné ji přepsat jako součin matice a vstupního vektoru. Tato projekce je ale nelineární, jelikož se tato operace dělí pomocí jednoho ze vstupních parametrů (pomocí parametru z). Pro budoucí výpočty je vhodnější, aby tato projekce byla uváděna jako součin matice a vektoru, čehož je možné dosáhnout při využití homogenních souřadnic [8].

Při využití homogenních souřadnic se mění systém souřadnic. Tato změna se provádí pomocí přidáním nového bodu do již existující souřadnice, takže z původního bodu *P = (x, y, z)* vznikne bod *P = (x, y, z, 1)* a stejným způsobem z namapovaného bodu *P' = (x', y')* vznikne *P' = (x', y', 1)*. Tedy při převodu z Euklidovského vektoru do homogenních souřadnic se na konec původního vektoru přidá nová dimenze s hodnotou 1. Rovnost mezi tímto vektorem je možná pouze v případě, že poslední hodnota

homogenních souřadnic je 1. Pokud je tedy homogenní souřadnice převáděna zpět do Euklidovského vektoru a hodnota posledního prvku souřadnice není rovná 1, je zapotřebí každou hodnotu vydělit hodnotou posledního prvku souřadnice. Po převedení do homogenních souřadnic je možné upravit mapovací vektor následovně [8]:

$$P' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & 0 & c_x & 0 \\ 0 & \beta & c_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & 0 & c_x & 0 \\ 0 & \beta & c_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} P \tag{1.5}$$

Matice obsahující parametry c_x, c_y, α, β se nazývá matice kamery K. Tyto parametry jsou označovány jako vnitřní parametry a jsou jedinečné pro každou kameru. Tyto parametry jsou neměnné a je možné je znovu využít pro mapování při využití stejné kamery [8].

1.4.2 Matice vnějších parametrů

V předchozí části byl popsán systém mapování bodů P z 3D do bodů P' na obrazové rovině s využitím vnitřních parametrů kamery, které jsou popsány pomocí matice. Jelikož ale původní obrazové body využívají jiný systém souřadnic, je nutné využít doplňující transformaci, která spojí obrazový referenční systém s kamerovým referenčním systémem. Pomocí této transformace je poté možné získat pozici a orientaci kamery vzhledem k obrazu. Tato transformace využívá rotační matice R a translačního vektoru T. Pokud je tedy dán obrazový bod P_w , je možné pomocí rovnice 1.6 zjistit jeho polohu v kamerovém systému [8] [1]:

$$P = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} P_w \tag{1.6}$$

Tuto matici je poté možné spojit s kamerovou maticí a po úpravách vznikne rovnice 1.7.

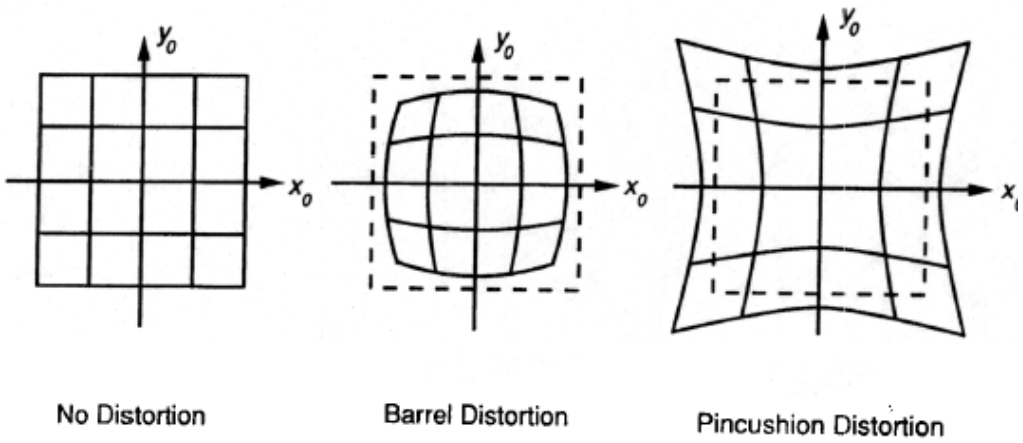
$$P = K \begin{bmatrix} R & T \end{bmatrix} P_w = M P_w \tag{1.7}$$

Matice M je spojením vnitřních a vnějších parametrů. Parametry K a T nesou název vnější parametry a nejsou závislé na dané kameře, ale vztahují se na konkrétní pohled kamery. V případě, že je změněna pozice kamery, je nutné tyto parametry znovu vypočítat [8].

1.5 Zkreslení objektivu

Výpočty uvedené v předchozích kapitolách počítaly s tím, že objektiv, který je využit u kamery má naprosto ideální vlastnosti a nezkresluje obraz. U většiny reálných objektivů se ovšem zkreslení projevuje v nějaké míře, například u širokoúhlých objektivů je toto zkreslení velice častým fenoménem. Je tedy nutné toto zkreslení nějakým způsobem kompenzovat, aby byly výpočty prováděné s obrazem co nejpřesnější. Toto zkreslení se v obraze projevuje například tak, že čáry, které jsou v reálném světě kolmé, jsou nějakým způsobem zaoblené [1] [8].

Zkreslení se může projevovat jako nepravidelné, nebo také jako nějaké vzory. Nejrozšířenějším typem zkreslení je radiální zkreslení, které vzniká díky nepřesnostem při výrobě čoček. Radiální zkreslení lze rozdělit na několik typů (Obr. 1.6), které nesou název dle vzhledu tohoto zkreslení. Mezi tyto typy se řadí soudkové zkreslení (Barrel Distortion), poduškové zkreslení (Pincushion Distortion). U soudkového zkreslení jsou souřadnice jednotlivých bodů přemístěny směrem od středu obrázku a u poduškového jsou naopak přemístěny směrem ke středu [1] [9].



Obr. 1.6 Typy radiálního zkreslení [9]

Základním modelem pro korekci zkreslení obrazu je polynomiální model, který je nejčastěji využíván pro popis radiálního zkreslení [10]:

$$r_u = r_d(1 + \lambda r_d^2 + \lambda r_d^4 + ...) \tag{1.8}$$

kde parametry r_u a r_d označují vzdálenost nezkreslených bodů (x_u, y_u) a zkreslených bodů (x_d, y_d) od středu zkresleného obrazu P a λ_i označuje jednotlivé parametry radiálního zkreslení. Polynomický model zkreslení funguje nejlépe pro objektivy s malým radiálním zkreslením. Pokud je ale zkreslení větší, je zapotřebí zakomponovat do

rovnice větší množství parametrů. S řešením přišel Andrew Fitzgibbon, který navrhnul model dělení (Divison Model), jehož hlavní výhodou oproti polynomickému modelu je, že dokáže pracovat s mnohem vyšší hodnotou zkreslení při využití menšího počtu parametrů [10]:

$$r_u = \frac{r_d}{(1 + \lambda r_d^2 + \lambda r_d^4 + ...)} \tag{1.9}$$

Při využití této metody je pro většinu existujících objektivů dostačující pouze jeden parametr [10]:

$$r_u = \frac{r_d}{(1 + \lambda r_d^2)} \tag{1.10}$$

Pro zjednodušení výpočtů u tohoto modelu je za střed zkreslení P považován samotný střed obrazu. Vzdálenost zkresleného bodu od středu je poté možné zjistit následovně: $r_d^2 = x_d^2 + y_d^2$. Pro získání pozice jednotlivých bodů jsou poté využity rovnice 1.11 [10].

$$x_u = \frac{x_d}{(1 + \lambda r_d^2)}, y_u = \frac{y_d}{(1 + \lambda r_d^2)} \tag{1.11}$$

1.6 Transformace

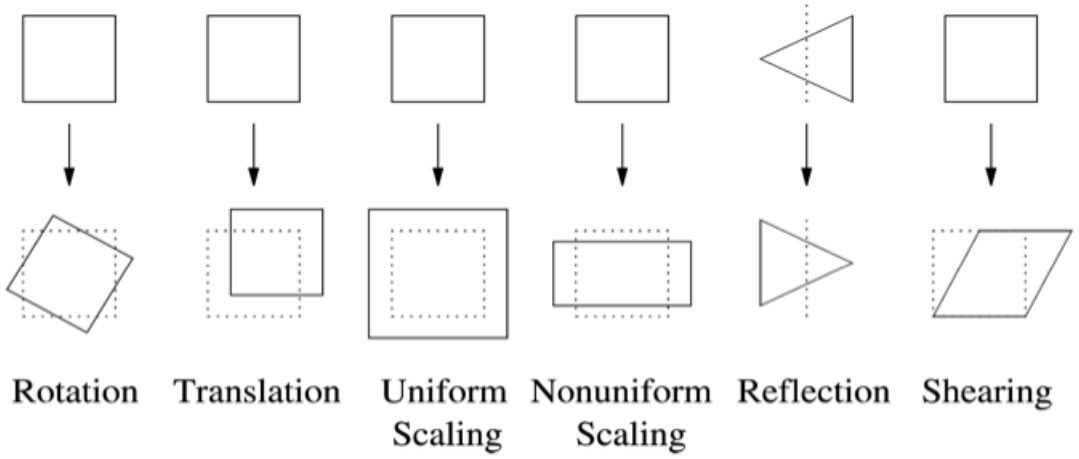
Transformace slouží k přeměně grafického útvaru do podoby jiného útvaru. Pomocí transformací (Obr. 1.7) je možné měnit polohu, velikost, rotaci či zkosení objektu. Pro jednotlivé transformace je vhodné využít homogenních souřadnic, jelikož některé transformace v euklidovských souřadnicích neumožňují násobení matic. Převod do homogenních souřadnic umožňuje řešení všech transformací jako násobení matic a tím pádem umožňuje jednotlivé transformace mezi sebou kombinovat [11] [12].

1.6.1 Translace

Translace, neboli posunutí, slouží ke změně pozice objektu na obrazovce. Ve 2D prostoru je možné posunout objekt přičtením translační souřadnice (d_x, d_y) k původní souřadnici (x, y), čímž vznikne souřadnice (x', y') [12]:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix} \tag{1.12}$$

Po převodu na homogenní souřadnice vznikne následující rovnice [11]:



Obr. 1.7 Přehled 2D transformací [11]

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{1.13}$$

Translaci je možné provést i u 3D objektů, kde přibude navíc jeden parametr z. Při využití homogenních souřadnic má translační matice u 3D objektů následující podobu [11]:

$$T(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.14}$$

1.6.2 Rotace

Operace rotace slouží pro pootočení objektu o daný úhel θ. Pokud je θ kladný, je rotace provedena proti směru hodinových ručiček, v případě záporného úhlu rotuje objekt ve směru hodinových ručiček. Rotace probíhá vzhledem ke středu dané soustavy, tedy k bodu (0,0) u 2D (Obr. 1.8) a (0,0,0) u 3D [11] [12].

Původní neupravený bod P se souřadnicemi (x, y) svírá s osou X úhel ϕ. Upravený bod P' se souřadnicemi (x', y') bude s osou x svírat úhel $\phi + \theta$. Při využití standardní trigonometrie mohou být souřadnice původního bodu P vyjádřeny následovně [12]:

$$x = rcos\phi, y = rsin\phi \tag{1.15}$$

Souřadnice upraveného bodu P' budou následující [11].

$$x' = rcos(\phi + \theta) = rcos\phi cos\theta - rsin\phi sin\theta \tag{1.16}$$

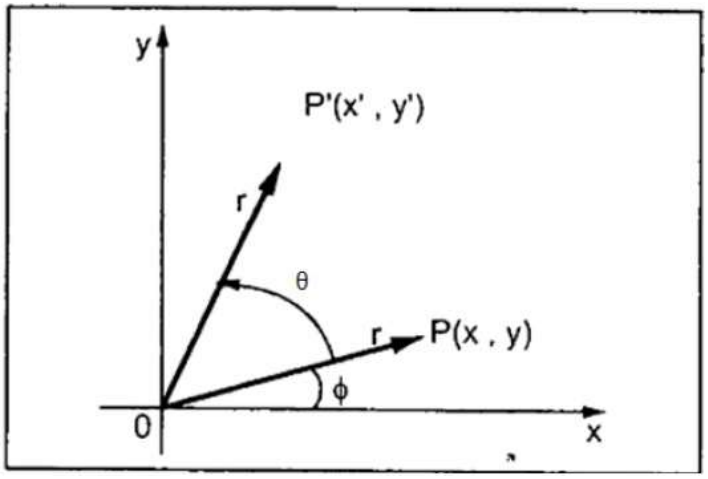
$$y' = rcos(\phi + \theta) = rcos\phi sin\theta + rsin\phi cos\theta \tag{1.17}$$

Rovnice pro původní bod je možné substituovat do rovnic pro upravený bod, z čehož poté vzniknou upravené rovnice, které je možné zapsat do maticového tvaru [11].

$$x' = xcos\theta - ysin\theta \tag{1.18}$$

$$y' = xsin\theta + ycos\theta \tag{1.19}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{1.20}$$



Obr. 1.8 Grafické zobrazení rotace [12]

Rotační matici je opět vhodné upravit do homogenních souřadnic, aby opět bylo možné využít tuto transformaci s ostatními transformacemi [11].

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{1.21}$$

Rotaci u 3D objektů je možné provést kolem všech tří os [11]:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.22}$$

$$R_y(\theta) = \begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.23}$$

$$R_z(\theta) = \begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.24}$$

1.6.3 Změna měřítka

Pro změnu velikosti objektu v obraze slouží změna měřítka. Při tomto procesu se daný objekt zvětšuje pokud je faktor změny měřítka větší než 1 a objekt se zmenšuje, pokud je faktor menší než 1 (například 0.5). Změny měřítka je dosaženo vynásobením souřadnic původního bodu s faktorem změny velikosti s_x a s_y, viz. rovnice 1.25. Rovnice 1.26 je upravená pro využití homogenních souřadnic [12].

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{1.25}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{1.26}$$

U 3D objektů je poté možné využít rovnici 1.27 [11].

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{1.27}$$

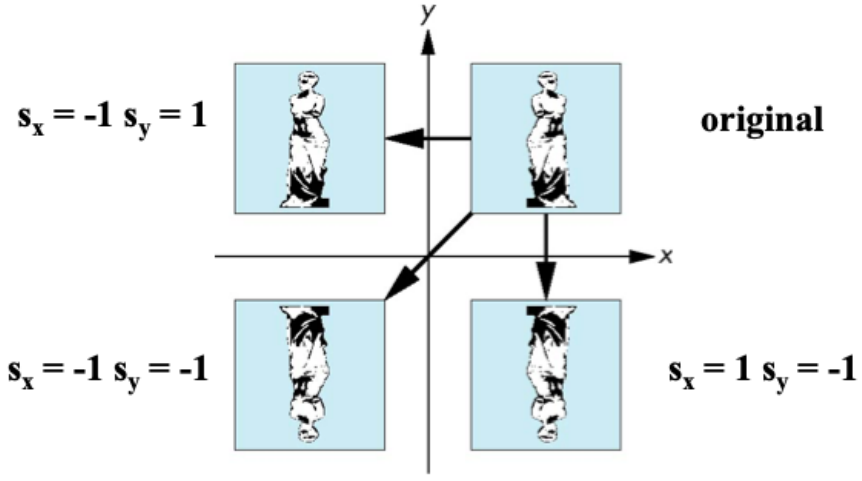
1.6.4 Odraz

Tato transformace slouží pro zrcadlové zobrazení daného objektu. Jedná se tedy o rotaci o 180° (Obr. 1.9). Při této transformaci nedochází ke změně velikosti, nebo tvaru objektu. U 2D objektu je možné provést tuto transformaci vůči ose x, y, ale také vůči oběma osám současně. Transformace odrazu je ve své podstatě změna měřítka se zápornými koeficienty. Rovnice 1.28 (odraz vůči ose x), 1.29 (odraz vůči ose y) a 1.30 (odraz vůči x a y) již zobrazují popis odrazu pro homogenní souřadnice [12].

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{1.28}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{1.29}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{1.30}$$



Obr. 1.9 Vliv hodnot parametrů na výsledný odraz obrázku [11]

U 3D objektů je možné provést transformaci vůči 3 plochám. Rovnice odrazu vůči ploše XY je popsána rovnicí 1.30, vůči ploše YZ rovnicí 1.31 a vůči rovině XZ rovnicí 1.32 [13].

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{1.31}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{1.32}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{1.33}$$

1.6.5 Zkosení

Transformace zkosení (Shear) obsahuje u 2D objektu 2 parametry zkosení, které určují, jak velké bude zkosení na dané ose. Parametr sh_x označuje velikost zkosení na ose x a sh_y označuje zkosení na ose y. Stejně jako u předchozích transformací je vhodnější využít homogenních souřadnic a výsledná rovnice bude mít následující podobu [12]:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{1.34}$$

Na 3D objekty je taktéž možné aplikovat transformaci zkosení. Oproti 2D objektu zde přibude jeden další parametr a to zkosení na ose z. Rovnice 1.34 zobrazuje spojené matice zkosení pro všechny tři osy. Horní index u jednotlivých parametrů značí, na které ose bude zkosení provedeno a spodní index značí, které parametry z dané trojice je potřeba aplikovat pro zkosení v dané ose [13].

$$\begin{bmatrix} x' \\ y' \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x^y & sh_x^z & 0 \\ sh_y^x & 1 & sh_y^z & 0 \\ sh_z^x & sh_z^y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{1.35}$$

1.6.6 Kombinace transformací

V případě, že je zapotřebí provést na jednom objektu několik transformací, je možné tyto transformace spojit dohromady, čímž vznikne jedna složená transformační matice. Pro toto spojení je ale důležité, aby jednotlivé dílčí transformace byly stejného typu. K tomu slouží právě převod do homogenních souřadnic, tak jak je popsán v rovnicích u jednotlivých transformací [12].

Při kombinování jednotlivých transformací je důležité dávat pozor na pořadí prováděných transformací. Jelikož jednou z vlastností matic je, že při násobení nejsou kumulativní a mohlo by se stát, že v případě špatného seřazení matic, může mít výsledný transformovaný objekt jiné vlastnosti, než jak byly předpokládány. Hlavním konceptem tohoto kombinování je zvýšení efektivnosti transformací, jelikož aplikování jedné zkombinované matice na vstupní body, je mnohem rychlejší, než aplikování jednotlivých dílčích transformací po sobě [12].

1.6.7 Homografie

Homografie je typ transformace, která slouží pro popis mapování jednoho bodu v daném obraze do stejného bodu v jiném obraze. Tato homografie nachází uplatnění například při mapování 2D rovinných bodů na body v obraze a tomuto mapování se říká rovinná homografie. Homografii mezi dvěma body je možné popsat pomocí rovnice 1.35 [14] [15].

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = H \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_0 0 & h_0 1 & h_0 2 \\ h_1 0 & h_1 1 & h_1 2 \\ h_2 0 & h_2 1 & h_2 2 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} \tag{1.36}$$

Pro zjištění této homografie je nutná znalost minimálně čtyř korespondujících bodů z obou obrazů, které jsou zpracovány aproximační metodou. Mezi nejvíce rozšířené aproximační metody patří Metoda nejmenších čtverců, RANSAC, LMEDS a RHO [15].

2 OpenCV

OpenCV je Open Source knihovna, která byla vyvinuta v roce 1999 Garym Bradskim, který v té době pracoval u společnosti Intel. Knihovna OpenCV byla vyvinuta za účelem zlepšení dostupnosti technologií pro počítačové vidění a umělou inteligenci pomocí solidní infrastruktury, kterou může kdokoli využít ve svých projektech. Knihovna je psána v jazyce C a C++ a je určena pro využití na operačních systémech Linux, Windows a MacOS X. Díky popularitě této knihovny se v dnešní době pracuje na integraci do programovacích jazyků jako je např. Java, Python a Matlab. Aktivně jsou také vyvíjeny upravené knihovny pro operační systémy Android a iOS, které slouží pro využití v mobilních aplikacích. Vývoj této knihovny je z větší části podporován firmami, jako je například Intel, Google nebo Itseez, která stojí za vývojem knihovny v prvopočátcích. V posledních letech se k vývoji přidala také firma Arraiy, která se stará o udržování webových stránek OpenCV.org [15] [16].

Knihovna OpenCV byla vytvořena tak, aby byla co nejefektivnější ve výpočtech a bylo ji možné použít v aplikacích, běžících v reálném čase. Jak již bylo zmíněno, knihovna je napsána v jazyce C++ a díky tomu je schopna těžit z vyššího výpočetního výkonu vícejádrových procesorů. Pokud knihovna běží na procesorech od firmy Intel, je možné využít knihovnu IPP (Integrated Performance Primitives), která je složená z nízkoúrovňových optimalizačních funkcí, které opět slouží pro další zrychlení výpočtů. Knihovna IPP je placená, což způsobuje, že většina uživatelů tuto knihovnu nevyužije. Společnost Intel proto vydala odvezenou knihovnu IPPICV, která je zdarma dostupná ke stažení [15].

Open Source licence pro knihovnu OpenCV byla vytvořena tak, aby mohl kdokoli tuto knihovnu využívat jak pro svoje soukromé účely, tak i pro komerční účely. V případě komerčního využití není nutné platit jakékoliv licenční poplatky vývojářům. Tento způsob licencování zajistil obrovskou popularitu této knihovny a v dnešní době ji využívají i velké nadnárodní společnosti jako například IBM, Microsoft, SONY, Siemens apod. Využití našla tato knihovna i ve výzkumných střediscích, jako například Standford, MIT, CMU, Cambridge nebo INRIA. Díky této popularitě nalezla knihovna OpenCV uplatnění ve velkém množství aplikací a produktů. Mezi tyto aplikaci patří například spojování satelitních fotografií v mapových aplikacích, zarovnávání obrazu při skenování, redukce šumu ve fotografiích určených pro medicínské využití, analýza objektů, detekční systémy pro zabezpečení budov, inspekce výrobků ve velkých továrnách, detekce poznávacích značek automobilů, nebo také nachází své uplatnění i ve vojenském vybavení [15].

Historie knihovny se datuje od roku 1999, kdy jeden ze zakladatelů Gary Bradski navštěvoval různé univerzity, kde si všímal toho, že každá univerzita má svou vlastní

infrastrukturu (knihovnu) pro zpracování obrazu, která putuje mezi studenty, kteří jí sami upravují pro své použití a pak dále předávají. Toto předávání knihovny umožnilo daným studentům vytvořit svůj vlastní projekt bez nutnosti začínat od nuly. Na základě tohoto jeho pozorování začal s vývojem OpenCV knihovny. Bradski byl zaměstnancem společnosti Intel, kde také vývoj této knihovny začal, ve spojení s ruským týmem vývojářů firmy Intel. V ruské divizi firmy se tohoto vývoje ujal Vadim Pisarevsky, který vyvíjel a optimalizoval kód pro tuto knihovnu. Cílem bylo vyvinout knihovnu, která by byla volně dostupná a umožnila by vývojářům snazší a optimalizovanější vývoj jejich vlastních aplikací. Zpřístupnění této knihovny také znamená, že uživatelé a vývojáři budou potřebovat výkonnější procesory, jelikož i přes dobrou optimalizaci, je počítačové vidění stále velice náročné na výkon počítače. Potřeba vyššího výkonu znamená zvýšení prodejů nových procesorů, což by mohl být jeden z důvodů, proč se společnost Intel rozhodla financovat vývoj této knihovny. V průběhu let byly vydávány nové verze OpenCV, ve kterých byly přidávány nové funkce a také byly opravovány nalezené chyby [15].

2.1 Struktura knihovny

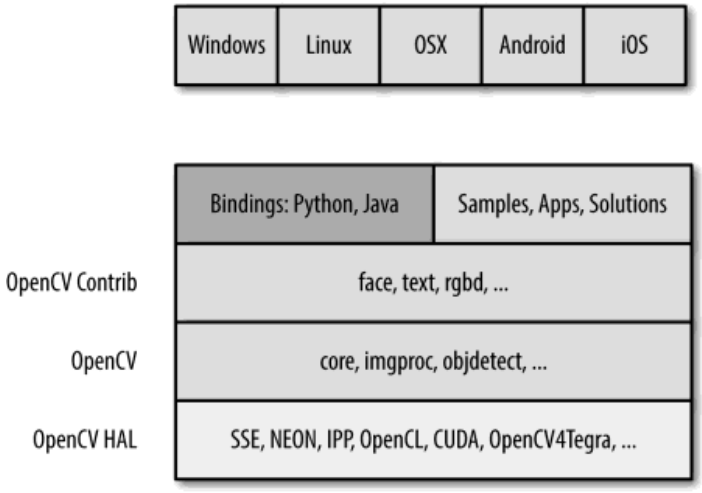
Struktura knihovny OpenCV je rozdělena do několika vrstev (Obr. 2.1). V nejvyšší vrstvě se nachází operační systém, pod kterým OpenCV běží. V druhé vrstvě se nachází jazykové vazby, např. mezi jazyky Python nebo Java a také se zde nachází ukázkový kód pro různé typy aplikací. Pod touto vrstvou se nachází vrstva opencv_contrib, která obsahuje převážně vysoko úrovňovou funkcionalitu. Čtvrtá úroveň obsahuje hlavní jádro knihovny OpenCV a poslední vrstva nazvaná HAL (hardware abstraction layer) obsahuje různé hardwarové optimalizace [15].

2.1.1 OpenCV Core

Vrstva OpenCV Core je rozdělena do několika modulů, které poté obsahují funkce pro daný typ využití. Všechny moduly a funkce v této vrstvě jsou spravovány OpenCV týmem, který zajišťuje, že tyto funkce budou funkční a stabilní. Seznam všech modulů se s postupem času rozšiřuje. Mezi tyto moduly patří například core, který obsahuje základní definice datových objektů spolu s jejich funkcemi, dále se zde nachází modul calib3d, který obsahuje implementace algoritmů pro kalibraci kamery. Podrobný seznam s popisem všech modulů je možné nalézt v dokumentaci, která je dostupná na webové stránce www.opencv.org [15].

2.1.2 OpenCV Contrib

Vrstva Contrib má stejnu strukturu jako vrstva Core, s tím rozdílem, že jednotlivé moduly, které se nacházejí v této vrstvě nejsou spravovány OpenCV týmem, ale na jejich vývoji se podílí komunita OpenCV. Moduly v této vrstvě mohou být nestabilní, ale také se zde mohou nacházet moduly, které nespadají pod OpenCV licenci, nebo mohou například obsahovat patentované algoritmy. Mezi tyto moduly patří například modul face, který byl vytvořen pro účely detekce a rozpoznání lidského obličeje, nebo také modul tracking, který obsahuje moderní algoritmy pro detekci objektů v obraze [15].



Obr. 2.1 Struktura knihovny OpenCV [17]

2.2 Datové typy

Knihovna OpenCV obsahuje velké množství datových typů, které byly vytvořeny za účelem zlepšení a ulehčení práce s důležitými koncepty počítačového vidění. Z organizačního hlediska je výhodnější rozdělit tyto datové typy do tří hlavních kategorií. V první řadě to jsou základní datové typy, které je možné vytvořit z existujících primitivních datových typů, které se nacházejí v jazyce C++ (například int nebo float). Mezi tyto primitivní typy patří vektory, jednoduché matice, ale také například body a velikosti [15]:

- **cv::Vec<> :** jedná se o kontejnerovou třídu, která je schopná ukládat primitivní datové typy. Tato třída je taktéž nazývána jako třída fixních vektorů. Rozdíl mezi tímto vektorem a vektorem dostupným v jazyce C++ je ten, že tento vektor má při kompilaci kódu vždy známou fixní velikost a může obsahovat pouze malé množství proměnných. Tento vektor je tedy vhodný pro části kódu, kde je dbáno

na rychlé a efektivní zpracování dat. Pro zjednodušení práce má tato třída předdefinovány aliasy, jako například cv::Vec2i nebo cv::Vec3d, kdy číslovka za Vec označuje počet prvků v daném vektoru a písmeno za tímto číslem označuje využití primitivní datový typ (int, float).

- **cv::Matx<>:** tato třída bývá také označována jako třída fixních matic. Stejně jako datový typ cv::Vec je cv::Matx určen pouze pro malé množství dat. V oblasti počítačového vidění je využíváno velké množství matic o velikosti 2x2, 3x3 a zřídka i 4x4, které jsou určeny například pro různé transformace. Tento datový typ byl opět vytvořen za účelem zrychlení a vylepšení efektivity kódu a díky fixní velikosti je možné eliminovat například problémy s dynamickým alokováním paměti. U tohoto datového typu je tedy opět nutné znát velikost matice před kompilací kódu.

- **cv::Point:** opět se jedná o kontejner s fixní velikostí, který slouží pro ukládání dvou nebo tří primitivních datových typů. Třída bodů má svoji vlastní šablonu a není tedy odvozena od třídy cv::Vec. Hlavní rozdíl mezi tímto datovým typem a typem cv::Vec je ten, že u tohoto datového typu je přístup k jednotlivým proměnným zajištěn pomocí pojmenovaných proměnných x, y případně z, namísto přístupu pomocí indexů, jak je tomu u třídy cv::Vec. Opět jsou zde předvytvořené aliasy, jako například cv::Point2f.

- **cv::Scalar:** tato třída je odvozena od cv::Vec a slouží pro ukládání jednotlivých hodnot pixelů, které poté slouží pro vykreslení dané barvy. Tato třída může obsahovat až 4 proměnné typu double (hodnoty s desetinnou čárkou).

- **cv::Size:** je velice podobná třídě cv::Point, ale obsahuje pouze 2 proměnné. Tato třída slouží pro specifikování velikosti objektů, to může být například velikost vstupního obrazu. Přístup k proměnným v této třídě je možný pomocí width (šířka) a height (výška).

Druhou kategorií jsou tzv. pomocné objekty, které reprezentují abstraktní koncepty, jako například třída ukazatelů garbage-collecting, která slouží pro odstranění nevyužívaných ukazatelů, nebo také různé abstrakce, jako například ukončovací kritéria iteračních algoritmů. Třetí kategorií jsou tzv. velká pole. Objekty v této kategorii obsahují složená pole primitivních datových typů, nebo dříve specifikovaných základních datových typů. Hlavním příkladem této kategorie je objekt cv::Mat, který slouží k ukládání několika dimenzionálních polí, které obsahují libovolná data [15].

Datový typ cv::Mat patří do kategorie velkých polí a jedná se o základní stavební kámen celé této knihovny. Většina funkcí, která se nachází v této knihovně, pracuje

nějakým způsobem s tímto datovým typem. Tento datový typ slouží k ukládání jednotlivých polí o nekonečném počtu dimenzí. Jednotlivé prvky každého pole jsou do matice vkládány v pořadí do jednotlivých řádků matice, kde konec každého řádku je oddělen středníkem. Pokud je tedy do matice o šířce 100 vkládáno trojrozměrné pole, tak jeden řádek obsahuje 300 hodnot. Každý element matice nemusí nutně obsahovat pouze jednu hodnotu, ale může se zde nacházet více hodnot. Typ matice je možné specifikovat pomocí předvytvořených aliasů, mezi které patří například CV_32FC3, který specifikuje, že se jedná o třídimenzionální matici, kde každý prvek této matice může obsahovat 32 bitové číslo typu float [15].

Pro každý z výše uvedených datových typů existují funkce, které usnadňují práci s těmito datovými typy. Pro cv::Mat například existuje funkce cvtColor, které slouží k převodu dané matice do jiné podoby. Pomocí tohoto převodu je například možné změnit formát matice na jednorozměrné pole, což způsobí převod obrázku z barevného na černobílý [15].

2.3 Kalibrace

Proces kalibrace slouží k získání parametrů, pomocí kterých je možné odvodit proces převodu reálného obrazu do digitální formy. Jak již bylo popsáno v kapitole 1.4, tento převod je realizován s využitím matice kamery a matice vnějších parametrů. V následujících kapitolách budou popsány metody kalibrace vnitřních parametrů kamery spolu s prostorovou kalibrací kamery, které jsou dostupné v knihovně OpenCV [15].

2.3.1 Kalibrace kamery

Proces kalibrace kamery s využitím knihovny OpenCV spočívá v analýze několika fotografií, předem známého kalibračního objektu, pořízených daným fotoaparátem. Jako kalibrační objekt je možné využít několik typů předem známých obrazců. Jedním z těchto kalibračních objektů je černo bílá šachovnice, u které jsou předem známy jednotlivé velikosti. Druhý typ kalibračního objektu je složen s několika vyplněných kruhů, které mají stejnou velikost a také jsou známy i velikosti mezer mezi nimi. V závislosti na kalibračním objektu je poté nutné zvolit správnou funkci, která slouží k nalezení těchto objektů v obraze. Pro šachovnici je to funkce findChessboardCorners() a pro vyplněné kruhy se využívá funkce findCirclesGrid(). Výstupem první zmíněné funkce jsou pozice rohů jednotlivých čtverců v obraze a výstupem druhů funkce jsou pozice středů daných kruhů [15] [18].

Po zjištění těchto pozic je možné začít s kalibrací kamery. Pro tuto kalibraci slouží funkce calibrateCamera(), která jako vstup bere pozice bodů z předchozích funkcí a referenční body, jejichž pozice se nachází ve 2D rovině. Funkce calibrateCamera poté

provede odhad jednotlivých parametrů na základě množiny vstupních bodů. Výstupem této funkce je poté matice vnitřních parametrů kamery, rotační a translační vektory pro každou fotografii s kalibračním objektem a vektor parametrů, které slouží pro odstranění zakřivení objektivu. Rotační a translační vektory popisují polohy kalibračního objektu v obraze vůči kameře, což znamená, že jsou tyto vektory vázány na danou fotografii a není tedy možné je využít pro práci s jiným pohledem kamery. Matice vnitřních parametrů se váže na danou kameru a je tedy možné ji nadále využívat [15] [18].

Hlavním důležitým výstupem této kalibrace je tedy vektor popisující zkreslení objektivu. Problém se zkreslením objektivu je možné pozorovat u levných kamer, nebo také u kamer s velkým zorným polem. U dnešních profesionálních kamer se toto zkreslení nemusí vůbec projevovat, a pokud ano, tak pouze v takové míře, že nějakým zásadním způsobem neovlivní výsledný obrázek. Nevýhodou této kalibrační metody je fakt, že je zapotřebí pomocí dané kamery pořídit několik snímků, což může být problém u kamer, které se například nacházejí na nějakém stožáru, nebo v těžce dostupných místech. Avšak, pokud se i tak v obraze nachází nějaké zkreslení, které výrazně ovlivňuje výsledný obrázek, je vhodnější zvolit manuální způsob kalibrace zkreslení, například pomocí metody uvedené v kapitole 1.5 [15] [18].

2.3.2 Kalibrace prostoru

V počítačovém vidění je kalibrace prostoru definována jako rovinná homografie, při které dochází k mapování bodů z jedné roviny do druhé. V jedné rovině jsou definovány referenční body a na ně jsou mapovány body ze vstupního obrazu. Z těchto bodů je poté pomocí aproximačního algoritmu zjištěna podoba výsledné matice. Knihovna OpenCV obsahuje několik funkcí, které slouží pro zjištění této homografie [15].

První a zároveň nejvíce využívanou funkcí je findHomography(), která na základě vstupních obrazových a referenčních bodů vypočítá matici homografie. U této funkce je možné specifikovat typ aproximačního algoritmu pomocí parametru. Mezi tyto algoritmy patří [15] [18]:

- **Metoda nejmenších čtverců :** Jedná se o výchozí metodu pro výpočet matice homografie. Pro výpočet matice používá všechny vstupní body. Ze všech možných řešení poté vybírá takové, které minimalizuje chybu reprojekce. Tato chyba je v tomto případě suma umocněných euklidovských vzdáleností mezi součinem matice homografie s originálními body a s referenčními body. Vstupní body zde ale mohou obsahovat extrémní hodnoty, které se drastickým způsobem liší od zbytku hodnot a kvůli kterým poté vznikají zkreslené výsledky. Jelikož tato metoda pracuje se všemi body, mohou tyto extrémní body způsobit značné vychýlení od správného výsledku. Z tohoto důvodu vznikly jiné metody, které řeší

tento problém.

- **RANSAC:** Tato metoda náhodně vybere nějakou podmnožinu bodů ze vstupních bodů a pro tuto podmnožinu vypočítá matici homografie. Zbývající body jsou využity k vylepšení tohoto řešení, pokud jsou alespoň částečně konzistentní s prvotním výpočtem. Tímto způsobem dojde k eliminaci extrémních hodnot a ke zlepšení kvality výsledku. RANSAC algoritmus vypočítá tímto způsobem několik možných řešení a vybere takové, které je složeno z největšího počtu bodů.

- **LMEDS:** LMEDS algoritmus vychází z metody nejmenších čtverců, ale dochází zde k minimalizaci střední chyby. Výhodou je, že tento algoritmus nepotřebuje žádné další vstupní podmínky pro svoji funkci. Nevýhodou ale je, že pro správný výpočet potřebuje, aby počet neextrémních bodů byl většinový.

- **RHO:** Jedná se o algoritmus, který byl představen ve verzi OpenCV 3.0 a jedná se o vylepšenou verzi metody RANSAC, která se nazývá PROSAC a ta využívá při výpočtech váhy. Tato metoda je mnohem rychlejší v případě, že vstupní množina obsahuje větší množství extrémních bodů.

Druhou funkcí pro výpočet matice homografie je getPerspectiveTransform(), která pro výpočet matice využívá pouze 4 obrazové a 4 referenční body. Tato metoda je vhodnější v případě manuálního zadávání bodů, jelikož na rozdíl od metody findHomography() zde není možné specifikovat požadovanou metodu výpočtu [15].

3 Programovací jazyk C++

Jedná se o programovací jazyk, který je standardizovaný, univerzální a objektově orientovaný. Jazyk C++ je odvozený z jazyka C a jedná se od nadmnožinu jazyku C. Téměř všechny výrazy používané v jazyce C je možné využít i v jazyce C++, opak ale není možný. Jedním z rozdílů je, že tento jazyk je schopen pracovat s objektově orientovaným programováním. Jazyk C++ patří mezi nejpoužívanější programovací jazyky a je využíván ve velkých projektech a aplikacích, jako například Adobe Photoshop, Spotify, Youtube, Amazon a Mozilla Firefox [19] [23].

Jazyk C++ začal vznikat v roce 1979, kdy Bjarne Stroustrup pracoval na své rigorózní práci, u které využíval programovací jazyk Simula. Jazyk Simula vznikl za účelem simulací a je považován za první objektově orientovaný programovací jazyk. Stroustrup vypozoroval, že tento objektově orientovaný přístup by mohl být vhodný pro vývoj softwaru, ale jazyk Simula byl na toto využití nesmírně pomalý. Stroustrup tedy začal pracovat na vývoji jazyka, který zpočátku nazval C s třídami, což jak již název vypovídá, byl programovací jazyk založený na jazyce C. Jeho cílem bylo přidat objektově orientované programování do jazyka C, který byl v té době hojně využívaným a uznávaným programovacím jazykem. C s třídami zpočátku obsahoval vše, co bylo dostupné v jazyce C spolu s třídami, základní dědičností, výchozími argumenty funkcí a velice silným kontrolováním typů proměnných [22] [20].

V roce 1983 bylo jméno tohoto jazyka změněno z C s třídami na C++. V tomto roce bylo přidáno veliké množství dalších funkcí, jako například virtuální funkce, přetěžování funkcí, reference atp. V roce 1985 byla Stroustrupem vydána kniha C++ programovací jazyk, která sloužila jako velice důležitý manuál pro práci s tímto jazykem, jelikož v této době nebyl ještě standardizován. V roce 1998 byl vydán první standard jazyka C++ nazvaný C++98. V roce 2003 vznikl nový standard, jelikož původní C++98 obsahoval několik problémů, a proto musel být upraven, tak vznikl standard C++03. V roce 2005 vydala komise pro C++ standardy zprávu, ve které vývojáře informovala o detailech připravovaných změn v následujícím standardu, jehož vydání bylo plánováno někdy před rokem 2010. Ovšem k vydání nového standardu došlo až v roce 2011, tento standard dostal název C++11 a obsahuje nesčetné množství vylepšení a oprav známých chyb. Od standardu C++11 vydává C++ komise nové standardy každé tři roky. Existují tedy standardy C++14, C++17 a C++20, kdy v každém standardu byly přidány nové technologie. V roce 2020 se konala schůzka komise C++ a byl vytvořen plán pro vydání nového standardu C++23, jehož vydání je plánováno na rok 2023 [20] [21].

3.1 Objektově orientované programování

Objektově orientované programování (OOP) popisuje způsob, jakým je kód strukturován, aby byl co nejpřehlednější, znovu použitelné a zapouzdřené. Před příchodem tohoto modelu programování byly využívány programovací modely, které měly velké množství limitací a u velkých projektů způsobovaly rozsáhlé problémy. Mezi programovací jazyky, které nepracují s objektově orientovaným programováním patří například jazyk C nebo Pascal, kterým se také říká procedurální jazyky. Program napsaný procedurálním jazykem je ve své podstatě seznam instrukcí. Pro velmi malé programy je tento přístup zcela dostatečný, ale problémy nastávají při rozsáhlejších programech. Z tohoto důvodu se u těchto jazyků začaly využívat tzv. funkce (toto označení je využíváno u C a C++, ostatní jazyky využívají označení, jako např. podprogram, procedura atd.), kde každá funkce má jasně definovaný účel a má jasně definované rozhraní pro ostatní funkce v programu. Tomuto typu programování se také říká strukturované programování. Ovšem i strukturované programování má své limity, jedním z nich je například neomezený přístup ke všem proměnným v kódu, což u komplexních programů může způsobovat obrovské problémy, jelikož se může stát, že vývojář nevědomky pracuje s daty, které náleží k jiné části kódu [23].

Kvůli problémům, které jsou uvedeny v předchozím odstavci, vznikl model objektově orientovaného programování. Základní myšlenkou tohoto modelu je zkombinovat jednotlivá data a k nim náležící funkce, které s nimi pracují do jednoho bloku, zvaného objekt. Funkce, které náleží danému objektu jsou většinou jediným způsobem, jakým je možné získávat a upravovat data v tomto objektu. Data by v tomto objektu měla být uložena tak, aby k nim bylo možné přistupovat pouze pomocí daných funkcí objektu. Tomuto způsobu uložení dat se říká zapouzdření a je tím zajištěno, že v případě potřeby úpravy dat je přesně stanoveno, jaká data se upravují, na rozdíl od metod v předchozím odstavci, kdy bylo možné přistupovat ke všem datům v daném kódu. Na jednotlivé objekty je možné nahlížet jako na reálné věci kolem nás. Každá reálná věc má dvě charakteristiky, nějakou vlastnost a chování. Hlavním stavebním prvkem při programování s modelem OOP je třída, která obsahuje jednotlivé data a funkce pro práci s těmito daty. Jednotlivé objekty jsou poté instancí této třídy, kde daným datům a funkcím jsou přiřazeny konkrétní hodnoty. Dalším důležitým prvkem OOP je dědičnost, která umožňuje vytvořit podtřídy, které dědí vše z nadřazených tříd a zároveň mohou specifikovat svá vlastní data a funkce [22] [23].

3.2 Typy knihoven

Knihovna je balíček kódu, který je vytvářen za účelem znovuvyužití u několika různých programů. Pro implementaci knihovny do kódu jsou většinou zapotřebí dvě části. První

část obsahuje hlavičkové soubory, které definují funkcionalitu dané knihovny. Druhá část obsahuje předkompilovaný binární soubor do strojového jazyka, který popisuje implementace jednotlivých funkcí. Knihovny jsou většinou předkompilovány z několika důvodů. Prvním důvodem je, že tyto knihovny nebývají často upravovány, a proto není zapotřebí je kompilovat při každé kompilaci programu. Druhým důvodem je bezpečnost dat, jelikož jsou předkompilované knihovny zobrazovány jako strojový kód, není možné z nich jednoduše získat data, nebo tyto data upravovat. V jazyce C++ se knihovny rozdělují do dvou kategorií [24] :

- **Statické :** tento typ knihovny se kompiluje do daného programu, ve kterém je knihovna využívána. Při kompilaci programu, který obsahuje statickou knihovnu se kompletně celá knihovna nakopíruje do výsledného spustitelného programu. V operačním systému Windows jsou tyto knihovny označeny koncovkou .lib a u operačního systému Linux koncovkou .a. Využívání statické knihovny má výhody v tom, že při distribuci výsledného programu je nutné distribuovat pouze jeden spustitelný soubor a jelikož je tedy knihovna součástí tohoto spustitelného souboru, je zaručena kompatibilita mezi knihovnou a daným programem. Nevýhodou této knihy je fakt, že tato knihovna zvyšuje velikost daného programu, což v případě existence více programů se stejnou knihovnou způsobuje zabírání zbytečného místa na disku. Další nevýhodou je, že v případě aktualizace knihovny je zapotřebí znovu zkompilovat celý program, který danou knihovnu využívá.

- **Dynamické :** tato knihovna není součástí programu, ale načítá se do programu až při spuštění. Při kompilaci programu není knihovna kopírována do výsledného spustitelného souboru, ale zůstává jako separátní jednotka. V operačním systému Windows mají tyto knihovny koncovku .dll a u operačního systému Linux koncovku .so. Pokud se v počítači nachází více programů, které tuto knihovnu využívají, každý tento program přistupuje do této jediné knihovny, což na rozdíl od statické metody šetří místo na disku. Jelikož knihovna není součástí každého programu, je možné provádět v této knihovně změny bez nutnosti znovukompilovaní celých programů.

4 Qt

Qt je multiplatformní framework, který slouží pro tvorbu počítačových, vestavěných a mobilních aplikací. Ve většině případů ale slouží jako sada nástrojů pro tvorbu grafických rozhraní. Mezi podporované platformy patří například Windows, MacOS X, Linux, Android nebo iOS [25].

S vývojem tohoto frameworku začali v roce 1990 Švédští programátoři Eirik Chambeeng a Naavard Nord. V roce 1995 byla představena verze Qt 0.9, která byla volně dostupná ke stažení a fungovala prozatím pouze na operačním systému Linux. Verze 1.0 byla představena v roce 1996 a v tomto roce se také prvním velkým zákazníkem stala Evropská vesmírná agentura, následně v roce 1999 byla vydána upravená verze 2.0. V této době Qt podporovalo pouze operační systémy Linux a Windows, což se změnilo v roce 2001 s verzí Qt 3.0, která přidala podporu pro operační systém MacOS X a zároveň byl představen nový nástroj pro tvorbu grafického rozhraní. V roce 2005 prošel celý framework kompletním předěláním, a tím vznikla verze 4.0. S verzí Qt 5.0, která byla představena v roce 2012 přišla další kompletní přeměna tohoto frameworku a zároveň byla firma odkoupena společností Digia. Nejnovější verze Qt 6.0 byla představena 8. prosince 2020 [26].

Qt je složeno z velkého množství sad určených pro vývoj software (SDK), které obsahují širokou škálu nástrojů a knihoven sloužících k ulehčení práce softwarovým vývojářům. Pro ulehčení práce při vývoji programů zde existuje vývojové prostředí Qt Creator, které nabízí spoustu užitečných nástrojů pro práci. Mezi tyto nástroje patří například GUI designer, který slouží k modelování grafického prostředí dané aplikace. Pro vývoj aplikací s grafickým prostředím pro mobilní aplikace zde existuje vývojové prostředí Qt Quick Designer, které obsahuje různé nadstandardní prvky, využitelné při vývoji mobilních aplikací [25].

5 Open source

Pojem open source označuje něco, co je veřejně dostupné, a každý uživatel si tuto věc může upravovat a sdílet podle vlastní libosti. Tento pojem vznikl v kontextu softwarového vývoje a popisoval určitý způsob vývoje počítačových programů. V dnešní době se ale pod tímto pojmem nachází a skrývá více věcí. Open source projekty pracují na principu otevřeného sdílení dat, spolupráce při vývoji, zrychleného prototypování a průhlednosti [27].

Open source v kontextu vývoje aplikací značí software, jehož zdrojový kód je veřejně dostupný a kdokoli s ním může nakládat podle svého uvážení. Open source software je sice volně dostupný, ale při vytváření a nakládání s ním, je nutné se striktně držet instrukcí, které jsou specifikovány organizací Open source initiative (OSI), která tyto důležité požadavky specifikuje v dokumentu Open source definition (OSD) [27].

Jedním z pravidel nakládání s open source programy je, že v případě prodeje softwaru, který pracuje s nějakým jiným open source projektem je nutné zdarma uvolnit zdrojový kód prodávaného projektu. Vývojáři, kteří chtějí zpeněžit svůj program, ale nechtějí uvolnit kompletní zdrojový kód volí raději takovou cestu, kdy svůj program nabízejí zdarma ke stažení, což jim umožňuje uchovat své zdrojové kódy skryté. V tomto případě získávají finance jiným způsobem, jako například zpoplatněnou uživatelskou podporou. Tímto směrem se vydala například společnost RedHat, která poskytuje svůj operační systém Fedora zdarma ke stažení a finance získává právě díky zpoplatněné uživatelské podpoře [27].

Na druhou stranu existují také společnosti, které nemají svůj produkt zpoplatněn a navíc také nabízejí zdrojový kód volně ke stažení. Tyto společnosti poté fungují díky dárcovským nebo sponzorským darům od různých společností nebo jedinců. Do této kategorie se řadí například Linux, WordPress, GIMP, OpenOffice, Blender, Firefox atp. [28]

6 GitHub

GitHub je webová stránka a online úložiště, které pomáhá vývojářům s ukládáním a spravováním jejich kódu. K pochopení významu GitHubu je nutné pochopit význam verzovacího systému a systému Git. Verzovací systém pomáhá vývojářům sledovat a spravovat změny v jejich programových kódech. S rostoucí velikostí projektu vstupuje na scénu právě verzovací systém, který pomáhá udržovat celý projekt pod kontrolou. Hlavním prvkem verzovacích systémů je větvení a spojování větví. Při větvení si vývojář duplikuje část zdrojového kódu, se kterou potřebuje pracovat. V této části poté může bezpečně provádět potřebné změny bez narušení funkčnosti hlavního kódu. Po provedení potřebných změn je možné tuto upravenou část spojit s hlavní částí kódu. Všechny takto provedené změny je možné sledovat a v případě potíží tyto změny z hlavního kódu odstranit. Tento celek také někdy nese název repozitář [29].

Systém Git je specifický open source verzovací systém, který byl vytvořen v roce 2005 Linusem Torvaldsem, který mimo jiné stojí za vznikem operačního systému Linux. Jedná se o distribuovaný verzovací systém, což znamená, že kompletní kód a seznam v něm provedených změn je uložen na počítačích všech vývojářů, kteří na daném projektu spolupracují [29].

GitHub je postaven na verzovacím systému Git a jeho cílem je zjednodušit a zpřístupnit využívání tohoto verzovacího systém pro širokou veřejnost. GitHub obsahuje přehledné grafické rozhraní, které je přizpůsobeno jak pro začátečníky, tak i pro pokročilé vývojáře. Tento systém ovšem není limitován pouze na ukládání kódu, ale jelikož slouží jako bezpečené online úložiště, našel oblibu i například u spisovatelů, kteří využívají tento verzovací systém pro přehlednější zobrazení provedených změn. Mimo větvení a spojování větví je jedno z hlavních dominant tohoto systému Pull požadavek, pomocí kterého mohou jednotliví vývojáři, kteří pracují na daném projektu navrhovat jimi upravené části kódu. U každého takového požadavku je možné sledovat provedené změny v kódu, které jsou graficky zobrazeny například pomocí červené a zelené barvy, kdy červená značí část odstraněného kódu a zelená část značí uživatelem přidaný kód. V případě schválení této změny kódu, dojde ke spojení obou částí - tomuto procesu se říká commit, který by měl obsahovat popis provedených změn [29] [30].

II. PRAKTICKÁ ČÁST

7 Definice požadavků

Výsledná knihovna by měla sloužit k prostorové kalibraci vstupního obrazu a k následnému kreslení do zkalibrovaného obrazu. Na základě této knihovny je také zapotřebí vytvořit testovací aplikaci, která bude sloužit k otestování a představení jednotlivých možností této knihovny. Knihovna i testovací aplikace by měly mít volně dostupný zdrojový kód, který bude moci kdokoli a jakýmkoli způsobem využívat. Knihovna by měla splňovat následující požadavky:

- **Přesnost:** knihovna by měla umožnit relativně přesnou prostorovou kalibraci, která zajistí přesné vykreslení požadovaných objektů do obrazu a díky které by bylo možné přesně měřit vzdálenosti mezi body v obraze.
- **Rychlost:** zpracování jednotlivých požadavků by mělo být relativně rychlé, aby nedocházelo ke zbytečnému zdržování celého procesu kalibrace a kreslení.
- **Odstranění zakřivení objektivu :** u novějších kamer je zakřivení objektivu ve většině případů odstraňováno v rámci daného zařízení. Může ale nastat situace, kdy toto zařízení nedisponuje touto technologií, nebo není dostatečná. Knihovna by tedy měla obsahovat funkci, která umožní případné zkreslení objektivu minimalizovat.
- **Univerzálnost :** hlavním zaměřením knihovny by měla být prostorová kalibrace pro využití na sportovních stadionech. Knihovna by ale měla být strukturována tak, aby ji bylo možné využít i mimo toto odvětví.
- **Práce se zkalibrovaným obrazem:** knihovna by taktéž měla obsahovat technologie, které umožní do zkalibrovaného obrazu vkládat různé objekty. Mezi požadované objekty patří horizontální a vertikální čáry, čtverce, obdélníky, kružnice a v poslední řadě také různé obrázky či loga.
- **Výstup:** po provedení procesu kalibrace by mělo být možné získat vytvořenou homografní matici, kterou by bylo možné znovu využít u daného pohledu kamery. V případě, že budou do zkalibrovaného obrazu vkládány objekty, je zapotřebí tyto objekty vykreslit do požadovaného obrazu a na průhledné pozadí.
- **Dokumentace:** knihovna by měla být řádně okomentovaná, aby bylo možné vytvořit dokumentaci pro tuto knihovnu, například pomocí aplikace Doxygen.
- **Implementace:** otestovaná a okomentovaná knihovna by poté měla být nahrána na nějaké open source úložiště a zde by měla obsahovat stručný popis, spolu s postupem instalace a jednoduchou ukázkou využití.

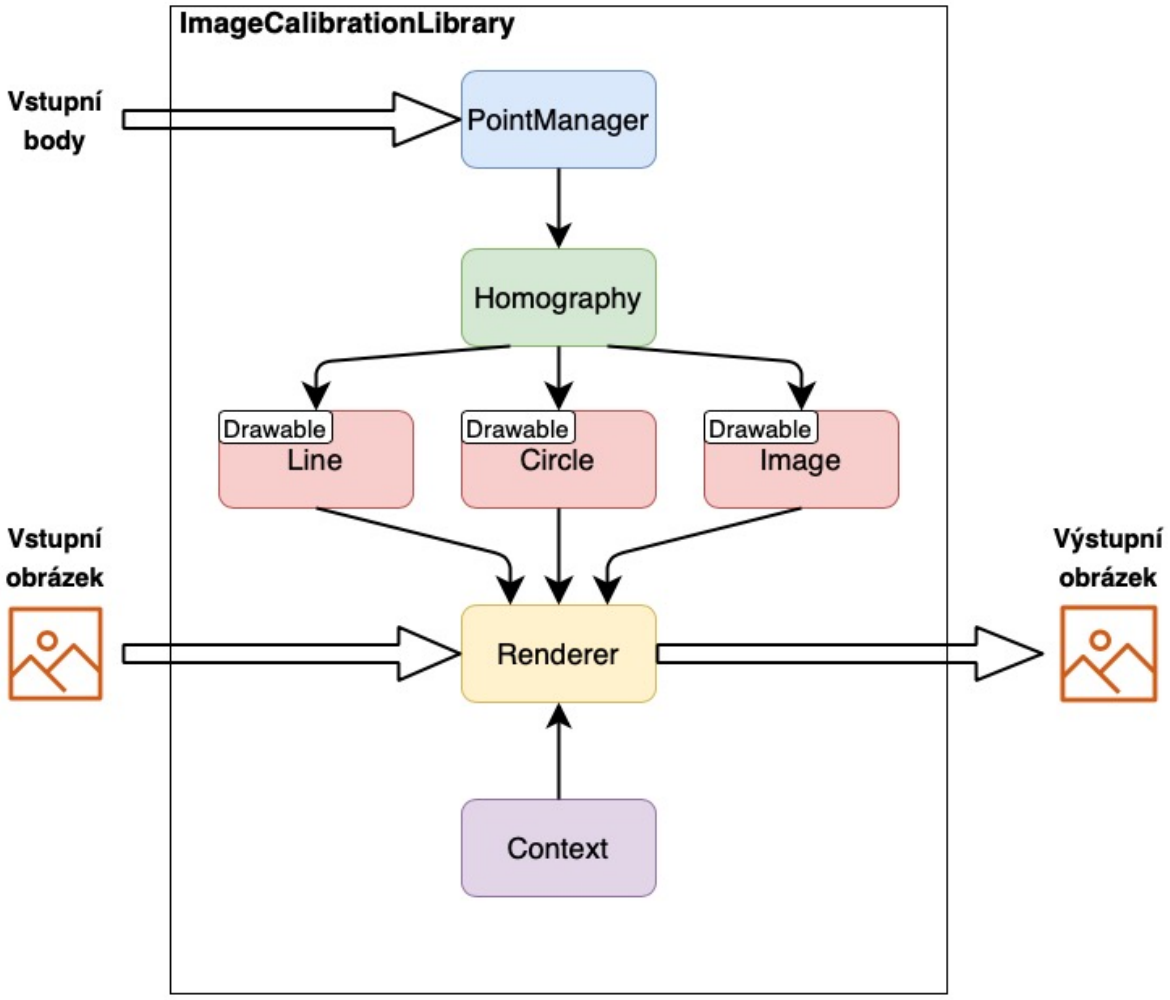
Na základě vytvořené knihovny je poté požadováno vytvořit testovací aplikaci, která by taktéž sloužila pro představení různých funkčností dané knihovny. Tato testovací aplikace by měla být vytvořena pomocí vývojového prostředí Qt a měla by splňovat následující požadavky:

- **Rychlost:** testovací aplikace by neměla zbytečně zatěžovat počítač zdlouhavými výpočty a nemělo by docházet k nežádoucím zásekům při práci.
- **Členění aplikace:** jednotlivá okna aplikace by měla být rozdělena přehledně a intuitivně. V hlavním menu by mělo být možné vytvořit nový projekt, nebo otevřít již existující projekt. Jedna část by měla obsahovat možnost korekce zkreslení objektivu i s možným uložením upraveného obrazu do nového souboru. Další část by měla sloužit pro prostorovou kalibraci obrazu, s využitím jednotlivých funkcí v knihovně. Poslední část by měla umožnit uživateli kreslení objektů, které jsou specifikovány u požadavků na knihovnu. Měla by se zde také nacházet historie vykreslených objektů s možností editaci či odstranění jednotlivých objektů.
- **Univerzálnost :** všechny dílčí části procesu kalibrace, tedy odstranění zkreslení a výpočet homografie, by mělo být možné uložit do souboru, který by umožnil znovu využít již vypočítané parametry pro kreslení do jiných obrazů ze stejné kamery. Pokud se v aplikaci budou nacházet další důležité parametry, je nutné tyto parametry taktéž zapsat do souboru, aby bylo možné je bez nutnosti dalších výpočtů znovu využít.
- **Výstup:** pokud je využita možnost kreslení do obrazu, je důležité implementovat možnost uložení výsledného obrazu do souboru. U možnosti uložení by mělo být možné vybrat, zdali je vyžadováno uložení se zobrazeným pozadím, nebo s transparentním pozadím.
- **Implementace:** stejně jako výše zmíněná knihovna by i tato aplikace měla být nahrána na nějaké open source úložiště, kde by taktéž měla obsahovat stručný popis, spolu s popisem instalace a krátkým manuálem.

Výslednou knihovnu a aplikaci je nutné řádně otestovat, aby byly odhaleny případné chyby při vývoji a tyto chyby by měly být odstraněny, aby byla zaručena funkčnost obou součástí.

8 Sestavení architektury

Na základě výše uvedených požadavků byla sestavena architektura knihovny. Aby bylo možné zkalibrovat daný pohled kamery, je nutné získat významné body ve vstupním obraze, které poté budou spojeny s mapovacími body, které odpovídají reálným rozměrům daných sportovních hřišť. Na základě těchto párů bude vypočítána výsledná homografní matice. Homografní matici bude možné využít pro kreslení do perspektivně zkresleného obrazu. Jednotlivé objekty určené pro kreslení budou uchovány v části nazvané Renderer. Všechny objekty budou vykreslovány do nově vytvořeného průhledného pozadí, jehož velikost bude odpovídat velikost vstupního obrazu. Po dokončení výběrů požadovaných objektů bude možné tyto objekty získat jak na průhledném pozadí, tak i jako vložené do vstupním obrazu. Výsledná knihovna bude rozdělena do několika částí (Obr. 8.1):



Obr. 8.1 Architektura knihovny

- **Část určená pro zpracování bodů (označená modře):** bude obsahovat

metody pro celkovou práci s body, které budou využity pro výpočet homografie. Jelikož se jedná primárně o knihovnu pro sportovní události, budou zde vytvořeny metody pro získání mapovacích bodů, které budou založeny na reálných velikostech sportovních hřišť.

- **Část pro výpočet homografie (označená zeleně):** zde se budou nacházet metody, které na základě vstupních párů bodů vypočítají homografní matici. Vstupem do této části bude instance třídy pro zpracování bodů.
- **Část určená pro kreslení objektů (označená červeně):** tato část bude brát jako vstupní argument instanci homografie a na základě této homografie poté budou vykreslovány jednotlivé objekty do obrazu. Jelikož téměř všechny objekty zmíněné v požadavcích mají plno společných atributů, bude tato část řešena pomocí dědičnosti. Zde se bude nacházet hlavní třída, která ponese společné prvky a metody jednotlivých objektů, které budou z této části dědit. Jednotlivé objekty budou mít svou vlastní třídu, která bude mimo jiné obsahovat individuální atributy a metody specifické pro daný objekt.
- **Část určená pro vykreslení do daného vstupního obrazu (označena žlutě):** bude mít za úkol vykreslovat požadované objekty do finálního obrázku. Jednotlivé instance objektů budou uchovány v proměnné, což umožní mazání a upravování jednotlivých objektů. Zde bude také možné vybrat, zdali je požadováno vykreslení do vloženého obrázku, nebo na transparentní pozadí, které pak bude možné do obrazu vložit pomocí externího programu.
- **Část určená pro uchování vykreslených objektů (označená fialově):** tato část se bude automaticky vytvářet společně s vykreslovací částí (označena žlutě) a bude sloužit pro vykreslování daných objektů. Zde bude vytvořen obrázek s průhledným pozadím a do něj budou kresleny všechny objekty.
- **Část pro uchování ostatních funkcí :** tato část není vyobrazena v nákresu architektury, jelikož se bude jednat pouze o sbírku funkcí, které budou užitečné při práci s knihovnou, ale hierarchicky nepatří do některé z ostatních částí.

Testovací aplikace bude vytvořena pomocí frameworku Qt s využitím programovacího jazyka C++ a knihovny OpenCV. Aplikace bude rozdělena do čtyř částí:

- **Hlavní menu:** bude sloužit pro navigaci mezi ostatními částmi. Zde také bude možné vytvořit nový projekt, který bude sloužit k ukládání důležitých dat, která se budou moci znovu využít pro daný pohled kamery. V této části bude také možné vybrat již existující projekt, jehož data se nahrají do aplikace a stav těchto dat bude zobrazen pomocí příslušných ukazatelů.

- **Kalibrace zkreslení objektivu:** v této části bude možné pomocí změny hodnoty koeficientu upravovat vstupní obrázek tak, aby bylo odstraněno zkreslení objektivu.

- **Homografie:** bude sloužit pro vytvoření homografní matice. Pomocí příslušných tlačítek se zde vybere vstupní obrázek a požadovaný typ mapovacích bodů. Párování obrazových a mapovacích bodů bude možné provést klikáním na příslušné mapovací body a jejich označováním ve vstupním obraze. Dále zde bude možnost upravit pozici obrazových bodů, která po aktivaci tlačítka automaticky poupraví pozici obrazových bodů tak, aby lépe odpovídaly mapovacím bodům. Výslednou homografní matici bude možné uložit pomocí stisku tlačítka, čímž dojde i k uložení této matice do souboru projektu.

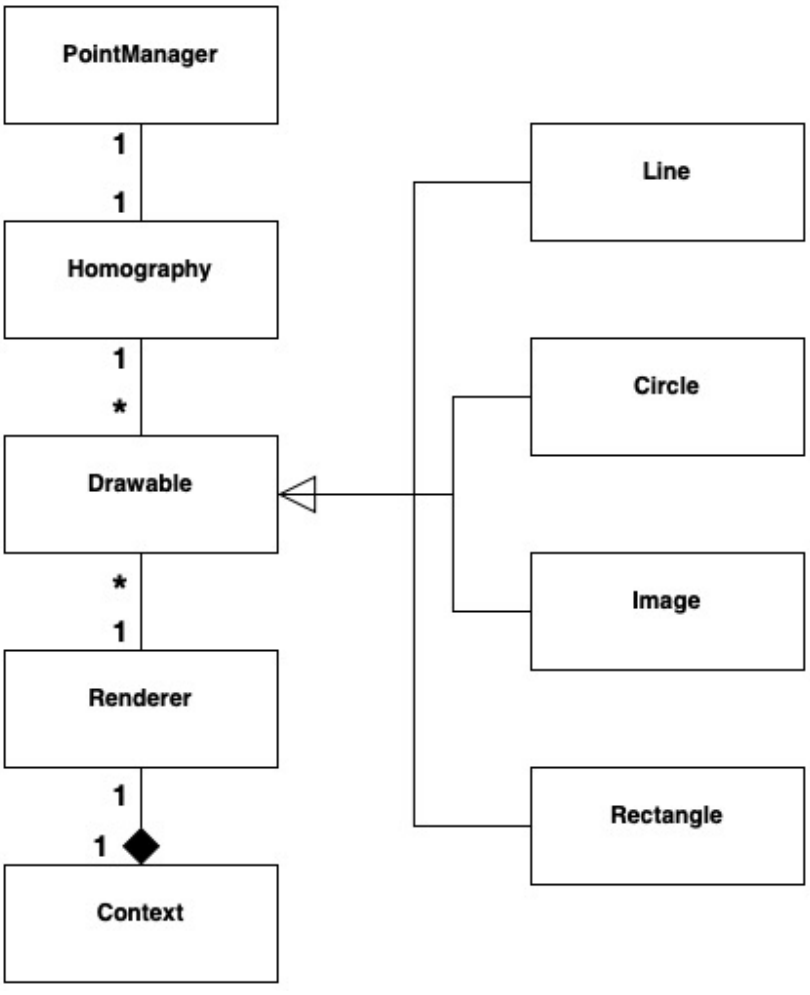
- **Kreslení :** v této části bude možné kreslit požadované objekty do zkalibrovaného obrazu. Pomocí několika tlačítek bude možné vybrat typ objektu, který bude vykreslen. Po vykreslení objektu dojde k aktualizaci tabulky, která bude zobrazovat historii přidaných objektů a umožní tyto objekty zpětně upravovat či mazat. Po dokončení vkládání požadovaných objektů bode možné pomocí tlačítek uložit výsledný obrázek. Ukládání obrázku bude možné provést dvěma způsoby. První způsob vloží požadované objekty do vstupního obrázku a ten poté vrátí. Druhý způsob vrátí objekty na průhledném pozadí.

8.1 Technologický základ knihovny

Jelikož je u knihovny kladen důraz na rychlost zpracování, tak byl pro vývoj knihovny zvolen programovací jazyk C++. Dále bude při vývoji využita volně dostupná knihovna OpenCV, která patří mezi hojně využívané knihovny. Knihovna OpenCV obsahuje velké množství funkcí a tříd, které jsou vhodné pro efektivní práci s obrazem, a díky své rozšířenosti, jsou tyto funkce velice dobře optimalizované. Pro získání výsledného knihovního archivu bude využita technologie cmake, která patří mezi nejrozšířenější technologie pro kompilaci C++ kódu. Knihovna taktéž bude obsahovat soubor .pri, který slouží ke zjednodušení integrace knihovny do prostředí Qt, a tedy i do testovací aplikace.

9 Implementace knihovny

Implementace knihovny byla provedena na základě požadavků a sestavené architektury. Pro vývoj byl využit programovací jazyk C++ a knihovna OpenCV. Jednotlivé třídy byly navrhnuty podle sestavené architektury tak, aby výsledná knihovna byla přehledná a funkční. Obrázek 9.1 obsahuje zjednodušený diagram, který zobrazuje rozdělení knihovny do jednotlivých tříd a také zobrazuje relace mezi těmito třídami. Dle sestavené architektury byly vytvořeny odpovídající třídy. Třída PointManger bude tedy sloužit k práci s mapovacími body, které budou využity při výpočtu homografní matice. Tuto homografní matici bude možné využít pro kreslení objektů do vstupního obrazu. V následujících podkapitolách budou tyto třídy podrobněji popsány.



Obr. 9.1 Zjednodušený diagram tříd knihovny

9.1 Popis vytvořených tříd

9.1.1 PointManager

Tato třída slouží pro práci s obrazovými a mapovacími body, které jsou později důležité pro výpočet homografní matice. Diagram této třídy je možné nalézt v příloze P1. Pro uložení párů mapovacích a obrazových bodů se zde nachází struktura UserPoint, do které jsou tyto body vkládány tak, aby bylo zachováno mapování těchto bodů, které jsou velmi důležité pro budoucí výpočty. Instanci této třídy je možné vytvořit využitím konstruktoru, nebo pomocí metod createFor. Metody pro vytváření mapovacích bodů jsou určeny pro sportovní hřiště a obsahují předem vložené body, které odpovídají reálným velikostem daných hřišť. Mezi podporované sporty patří badminton, basketbal, fotbal, hokej, tenis a volejbal. V případě, že nejsou tyto body vhodné pro prostorovou kalibraci, je možné vytvořit prázdnou instanci třídy pomocí metody createCustom, do které je nutné vložit uživatelem specifikované mapovací body, které by měly odpovídat reálným rozměrům daného objektu.

Po dokončení mapování je možné všechny body v UserPoints využít pro výpočet homografní matice pomocí třídy Homography. Před tímto krokem je ovšem možné využít metody improvePoints, která na základě vstupního obrázku vylepší pozici obrazových bodů. Správnost homografní matice je možné ověřit pomocí překreslení vstupního obrazu do ptačí perspektivy. Předem vytvořené mapovací body jsou udávány v metrech, což při vykreslení pohledu z ptačí perspektivy způsobí, že výsledný obrázek bude velmi malý a téměř nečitelný. Z tohoto důvodu je možné při vytváření mapovacích bodů specifikovat velikost měřítka, které zajistí, že výsledné zobrazení pohledu z ptačí perspektivy bude mít větší velikost a bude lépe čitelné. Dalším parametrem je zde offset, neboli odsazení, které umožňuje posunout mapovací body na ose x a y. Toto odsazení způsobí posun mapovací roviny vůči vstupnímu obrazu a při zobrazování pohledu z ptačí perspektivy dojde k zobrazení pixelů i za hranicí mapovacích bodů.

9.1.2 Homography

Matice homografie je jedním z nejdůležitějších stavebních prvků této knihovny. Instance této třídy je využívána u kreslení objektů a je tedy důležité dbát na její správné vytvoření. Pro výpočet a uchování této matice byla vytvořena třída, jejíž instanci je možné vytvořit dvěma způsoby. Diagram třídy homography je zobrazen v příloze P II.

Prvním způsobem je vytvoření instance pomocí konstruktoru, který na vstupu vyžaduje vložení existující instance třídy PointManger, která obsahuje spárované obrazové a mapovací body. Při použití tohoto konstruktoru dochází automaticky k výpočtu a k uložení homografní matice. Uloženou matici je možné získat zavoláním metody getHomographyMatrix, případně getInverseHomographyMatrix, která vrátí inverzní homo-

grafní matici.

V případě, že pro daný pohled kamery existuje dříve vypočítaná homografní matice a tato kamera se nachází ve stejném stavu, tedy na stejné pozici a se stejnou velikosti přiblížení, je možné tuto existující matici znovu využít. V takovém případě je možné vytvořit instanci této třídy pomocí defaultního konstruktoru a není nutné znovu procházet celým procesem mapování bodů. Pro správnou funkčnost je nutné tuto existující matici vložit do instance Homography třídy pomocí metody setHomographyMatrix.

9.1.3 Drawable

Objekty určené k vykreslování do obrazu mají společné atributy, a proto byla pro lepší přehlednost výsledné knihovny vytvořena třída Drawable, která zastřešuje všechny dostupné objekty pro vykreslení a obsahuje jejich společné atributy a metody. Mezi tyto společné atributy patří průhlednost, barva a tloušťka čar. Diagram této třídy je vyobrazen v příloze P II. Jednotlivé objekty byly rozděleny do čtyř tříd, které jsou popsány níže.

- **Line :** Třída Line slouží pro vykreslování horizontálních a vertikálních čar do obrazu. Vykreslení daného typu čáry je závislé pouze na jednom vstupním bodu. Na základě tohoto bodu jsou poté vypočítány nové body, které zajistí vykreslení daného typu čáry do obrazu. Nové body jsou odvozeny z pozice původního bodu a z velikosti vstupního obrázku. Pro správné vykreslení je nutné při vytváření instance třídy specifikovat velikost mapovacího okna, kterou je možné zjistit metodou getWindowSize u třídy PointManager. Nově vytvořené body jsou poté zkontrolovány, zdali se nacházejí uvnitř daného obrazu. V případě, že tyto body mají nesmyslné hodnoty, dochází k upravení těchto bodů pomocí metody calculateLineIntersection, která zjišťuje, ve kterých místech požadovaná čára protíná okraje obrazu. Souřadnice protnutí jsou poté nastaveny jako body po finální vykreslení čáry.

  U horizontální i vertikální čáry je možné specifikovat hodnotu odsazení, neboli offSet, který způsobí, že krajní body výsledné čáry budou odsazeny tak, aby korespondovaly s daným typem sportovního hřiště a aby výsledná čára nezasahovala mimo toto hřiště. Hodnotu odsazení je vhodné specifikovat v případě, že tato hodnota byla nastavena při kalibraci pohledu pomocí tříd PointManager a Homography.

- **Circle :** Pro kreslení kružnic do obrazu je možné využít třídu Circle. Podobně jako u třídy Line je zde nutné specifikovat pouze jeden vstupní bod, kterým je označen střed vykreslované kružnice. Dále je zapotřebí specifikovat poloměr kružnice, který je zde udáván v pixelech.

- **Image :** Knihovna umožňuje i vkládání obrázků do vstupního obrazu. Zde je nutné specifikovat 2 body v obraze, mezi které bude požadovaný obrázek vložen. Pozice prvního obrazového bodu koresponduje s levým horním rohem vstupního obrázku, pozice druhého bodu koresponduje s pravým spodním rohem obrázku. Druhý bod dále také určuje velikost oblasti pro vložení obrázku. V závislosti na této velikosti je vstupní obrázek zmenšen případně zvětšen tak, aby vyplnil označenou oblast. Každý vstupní obrázek je možné rotovat vzhledem k pozici vstupních bodů. Rotaci je možné vybrat z výčtu rotací, který obsahuje možnosti 0°, 90°, 180° nebo 270°.

  Třída Image je schopna vkládat jak obrázky bez transparentního pozadí, tedy například ve formátu JPEG, tak i s obrázky, které obsahují transparentní pozadí. Pokud vstupní obrázek obsahuje toto transparentní pozadí, je vykreslena pouze netransparentní část.

- **Rectangle :** Posledním typem je třída Rectangle, která slouží k vykreslování obdélníků a čtverců. Stejně jako u třídy Image je i zde nutné specifikovat 2 body v obraze, které slouží k získání finálního objektu. U kreslení obdélníku tyto obrazové body specifikují protilehlé rohy obdélníku. Pokud je požadováno vykreslení čtverce, specifikuje druhý obrazový bod velikost oblasti, do které má být čtverec vykreslen. Velikost této oblasti slouží k výpočtu velikosti hrany čtverce a to tak, aby výsledný obrazec byl opravdu čtverec a zároveň nepřesahoval mimo stanovou oblast.

Diagramy výše zmíněných tříd je možné nalézt v příloze P III. Instanci výše zmíněných tříd je možné vytvořit využitím konstruktoru, nebo pomocí metody create. Pokud je instance vytvářena konstruktorem, je nutné dále doplnit potřebné atributy třídy pomocí metod set. U všech tříd je nutné doplnit body, které slouží pro vykreslení daného objektu. Při využití metody create jsou všechny důležité atributy vkládány jako vstupní proměnné a odpadá tedy nutnost nastavování atributů pomocí metod set. Metoda create vrací ukazatel na nově vytvořenou instanci dané třídy se všemi potřebnými atributy. Pokud pro vykreslování není využita instance třídy Renderer, je možné daný objekt přímo vykreslit pomocí metody draw do již vytvořené instance třídy Context. Pokud je ovšem pro vykreslování objektů využívána třída Renderer, je možné nově vytvořený objekt přidat pomocí metody addDrawable. Pro vykreslování je ovšem doporučeno vždy využívat třídu Renderer, která umožňuje spojení více objektů do jednoho výsledného obrázku.

9.1.4 Renderer

Třída Renderer slouží pro práci s jednotlivými instancemi vykreslovaných objektů. Uvnitř této třídy je definován vektor, který slouží pro uchovávání objektů. Vkládání nových instancí do zmíněného vektoru je možné pomocí metody addDrawable. Aby bylo možné nakládat s jednotlivými objekty, byla zde vytvořena metoda getDrawables, která vrací vektor všech dostupných objektů v dané instanci. Na základě tohoto výstupního vektoru je možné jednotlivé objekty upravovat a měnit jejich vnitřní atributy bez nutnosti generování nové instance objektu. V případě potřeby odstranění je možné vyvolat metodu removeDrawable, která z vnitřního vektoru odstraní požadovaný objekt. Diagram třídy Renderer je vyobrazen v příloze P IV.

Po vytvoření instance třídy Renderer, s využitím defaultního konstruktoru, je nutné do této instance vložit obrázek, do kterého budou vykresleny finální objekty. Při vložení tohoto obrázku je automaticky vytvořena nová instance třídy Context, která slouží jako průhledné pozadí pro vykreslení objektů a má stejnou velikost jako vstupní obrázek. Zavoláním metody render dojde k vykreslení všech existujících objektů do instance třídy Context. Pro získání vykreslených čar slouží metoda getOutputImage, která má jako vstupní parametr proměnnou includeBackground. Pokud je tato proměnná nastavena na hodnotu true, dojde ke spojení vloženého obrázku s instancí třídy Context obsahující objekty a výsledný obrázek je poté navrácen. Pokud je proměnná includeBackgorund nastavena na false, dochází k navrácení instance třídy Context s vykreslenými objekty.

9.1.5 Context

Tato třída slouží jako pozadí pro vykreslování objektů. Při vytvoření instance této třídy dochází k vytvoření nového obrázku, který má nastavenou absolutní průhlednost všech pixelů a slouží jako pozadí pro kreslení objektů. Instance této třídy je automaticky vytvořena při nastavení pozadí u třídy Renderer. V případě manuálního vytvoření je nutné specifikovat velikost daného pozadí, do kterého je možné ručně kreslit jednotlivé objekty pomocí metody draw. Pro získání výsledného pozadí s vykreslenými objekty slouží metoda getImage. Diagram třídy Context je vyobrazen v příloze P IV.

9.1.6 Sbírka funkcí Utils

Zde se nachází podpůrné funkce, které jsou užitečné při práci s knihovnou, ale hierarchicky nezapadají do výše zmíněných tříd. Mezi tyto funkce patří blendImages, která slouží pro spojení dvou vstupních obrázků, které obsahují transparentní pozadí a convertBGRtoBGRA, která slouží pro přidání alfa kanálu do již existujícího obrázku. Dále se zde také nachází funkce computeBirdsEyeView, která ze vstupního obrázku a vstupní instance třídy homografie vytvoří pohled z ptačí perspektivy. Pro korekci za-

křivení objektu se zde nachází funkce undistort, která na základě vstupních parametrů upraví vstupní obrázek.

9.2 Rozložení knihovny a implementace na Github

Každá třída v této knihovně se skládá ze dvou souborů

- jedním z nich je hlavičkový soubor, který obsahuje definice využívaných atributů a metod dané třídy.
- druhý soubor s koncovkou .cpp obsahuje implementace jednotlivých metod.

Pro lepší přehlednost byly tyto soubory rozděleny do dvou složek include a src, kde složka include obsahuje všechny hlavičkové soubory a složka src obsahuje všechny implementační soubory. Složky include a src dále obsahují podložku drawables, ve které se nacházejí jednotlivé objekty pro vykreslení, které dědí z třídy Drawble.

Aby bylo možné z daných souborů vytvořit knihovní archiv, byl vytvořen soubor CMakeLists.txt, který obsahuje důležité definice pro vytvoření knihovního archivu a je využíván programem CMake. Dále se zde nachází soubor s koncovkou .pri, který slouží pro vložení této knihovny do aplikace vytvořené pomocí frameworku Qt. Při využití .pri souboru není nutné vytvářet knihovní archiv pomocí aplikace CMake, jelikož se o vložení této knihovny stará samotný Qt framework.

Součástí všech hlavičkových souborů knihovny jsou komentáře, které slouží pro popis funkčnosti jednotlivých funkcí a metod v nich obsažených. Tyto komentáře byly napsány tak, aby umožnily vytvoření automatické dokumentace pro celý projekt. Tuto dokumentaci je možné vygenerovat, například pomocí programu Doxygen.

Všechny výše uvedené složky a soubory byly vloženy do nového repozitáře na webové stránce Github. Do zmíněného repozitáře dále byl dále přidán soubor README.md, který obsahuje základní informace o této knihovně spolu s postupem instalace. Jelikož se jedná o open source projekt, musí zde být specifikována licence, která informuje potencionální zájemce o podmínkách soukromého a komerčního využití této knihovny. U tohoto projektu je využita licence MIT, která opravňuje kohokoliv využívat tuto aplikaci jak pro soukromé, tak i pro komerční účely bez jakýchkoliv restrikcí. Výsledný repozitář je dostupný na webové adrese www.github.com/richardfous/ImageCalibrationLibrary

9.3 Postup instalace

Jak již bylo zmíněno v předchozí části, instalace knihovny je možné provést dvěma způsoby. Prvním způsobem je instalace pomocí aplikace CMake s využitím přiloženého souboru CMakeLists.txt. Druhým způsobem je vložení knihovny do existujícího Qt projektu s využití souboru s koncovkou .pri.

9.3.1 CMake

Před samotnou instalací je nutné nainstalovat knihovnu OpenCV, která je využita v této knihovně. Postup instalace OpenCV pro dané platformy je popsán na webové stránce www.opencv.org. Dále je také zapotřebí nainstalovat aplikaci CMake. Postup instalace CMake je popsán na webové stránce www.cmake.org.

Po vyřešení těchto závislostí je možné stáhnout tuto knihovnu z repozitáře GitHubu, který je dostupný na webové stránce ww.github.com/richardfous/ImageCalibrationLibrary. Po stažení je možné přejít k samotné instalaci knihovny. Při využití existujícího souboru CMakeLists.txt je nutné v kořenové složce knihovny vytvořit novou složku 3rdparty, do které je dále nutné vložit vytvořený archiv knihovny OpenCV spolu s hlavičkovými soubory knihovny OpenCV. Pokud je knihovna OpenCV instalována jako systémová knihovna, je nutné specifikovat cestu k této knihovně v sekci include_directories v souboru CMakeLists.txt.

Po provedení výše popsaných kroků je možné přistoupit k instalaci knihovny. V závislosti na využívané platformě se bude postup instalace lišit. Následující postup je možné využít s operačním systémem Linux. V první řadě je nutné vytvořit složku build, do které bude vložen výsledný knihovní archiv. Dále je nutné vytvořit inicializační soubory pomocí příkazu cmake .. a následně spustit instalaci pomocí příkazu make. Následující část kódu obsahuje všechny čtyři kroky spojené do jednoho příkazu.

```
$ mkdir build && cd build && cmake .. && make
```

Po dokončení se ve složce build vytvoří archiv s koncovkou .a, který lze poté vložit spolu s hlavičkovými soubory ve složce include do požadované aplikace.

9.3.2 Qt framework

Druhým způsobem instalace je využití souboru s koncovkou .pri, který je možné vložit do již existující aplikace vytvářené pomocí frameworku Qt. V tomto případě je nutné staženou knihovnu extrahovat do kořenové složky požadované Qt aplikace a následně do existujícího .pro souboru vložit odkaz na knihovní .pri soubor. Tato existující aplikace musí v .pro souboru obsahovat odkaz na nainstalovanou knihovnu OpenCV. Následující kód znázorňuje vložení .pri souboru do souboru .pro v Qt aplikaci.

```
include(ImageCalibrationLibrary/ImageCalibrationLibrary.pri)
```

Při využití tohoto způsobu není nutné pracovat s aplikací CMake, jelikož o instalaci knihovny do aplikace se postará samotný Qt framework.

9.4 Ukázka práce s knihovnou

Pro vykreslování objektů je důležité vytvořit instanci třídy Homography, která bude obsahovat homografní matici pro daný pohled kamery. Tu matici je možné vypočítat na základě obrazových a mapovacích bodů, které je nutné vložit do instance třídy PointManager. Následující část kódu zobrazuje vytvoření instance třídy PointManager pro fotbalové hřiště o šířce 105 metrů a výšce 68 metrů a přidání nového páru mapovacích bodů.

```
// Vytvoření instance třídy pointManager pro fotbal.
std::unique_ptr<PointManager> m_pointManager;
m_pointManager = PointManager::createForFootball(105, 68); ;
// Přidání nového páru mapovacích bodů.
// m_imagePoint značí bod v obraze.
// m_mappingPoint značí korespondující mapovací bod.
m_pointManager->addUserPoint(m_imagePoint, m_mappingPoint);
```

Po získání alespoň čtyř párů mapovacích bodů je možné vytvořit novou instanci třídy Homography s využitím konstruktoru, který má jako vstupní argument instanci třídy PointManager. V případě, že již někdy byla homografní matice pro daný pohled kamery vypočítána, je možné vytvořit defaultní konstruktor u třídy Homography a existující matici do této instance vložit metodou setHomographyMatrix.

```
// Vytvoření proměnné pro uchování ukazatele na novou instanci třídy
↪  Homography.
std::shared_ptr<Homography> m_homography;
// Vytvoření nové instance třídy na základě vstupní instance třídy
↪  PointManager.
m_homography = std::make_shared<Homography>(*m_pointManager);
// Vytvoření nové instance třídy pro již existující matici a vložení
↪  této matice do instance.
m_homography = std::make_shared<Homography>();
m_homography -> setHomographyMatrix(cv::Mat matrix);
// Získání vložené homografní matice a inverzní homografní matice.
m_homography->getHomographyMatrix();
m_homography->getInverseHomographyMatrix();
```

Instance třídy Homography, která obsahuje validní homografní matici může být využita pro vykreslování požadovaných objektů. Před samotným vykreslováním je nutné vytvořit instanci třídy Renderer, do které budou jednotlivé objekty vkládány.

```
// Vytvoření instance třídy Renderer
Renderer m_renderer;
// Vložení obrázku na pozadí do instance. Vstupní obrázek je zde
↪  reprezentován maticí cv::Mat.
m_renderer.setBackgroundImage(image);
//Vytvoření nového objektu pro vykreslení. Pro ukázku bylo vybráno
    zobrazení kružnice, vkládání ostatních typů objektů je velmi
    podobné. Bod m_imagePoint zde značí bod, kterým má procházet
↕   vertikální čára. Radius značí poloměr výsledné kružnice.
std::unique_ptr<Line> circle = Circle::create(m_homography, {
↪  m_imagePoints[0].x, m_imagePoints[0].y }, radius);
```

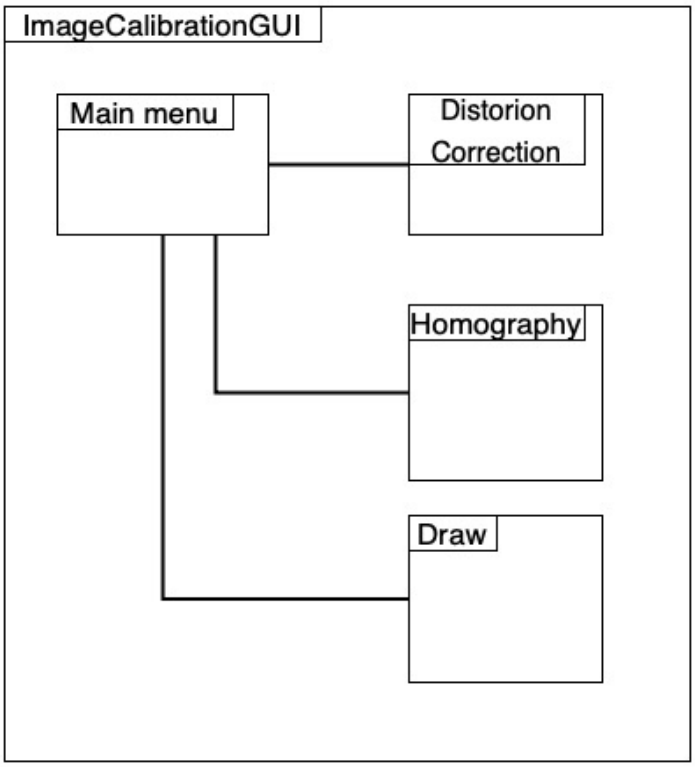
```
// Nastavení jednotlivých atributů daného kruhu. Alpha značí
    průhlednost vykresleného objektu, Color značí zvolenou barvu a
↪   Thickness značí šířku čáry výsledného objektu.
Circle->setAlpha(m_alpha);
Circle->setColor(m_color);
Circle->setThickness(m_thickness);
// Vkládání nového objektu do instance třídy Renderer.
m_renderer.addDrawable(std::move(circle));
```

Po vložení požadovaných objektů je možné u třídy Renderer zavolat metodu render, která všechny objekty vykreslí do vnitřně vytvořené instance třídy Context. Výsledný obrázek s vykreslenými objekty je možné získat zavoláním metody getOutputImage, která má jako vstupní atribut proměnnou includeBackground, a která určuje, zdali budou objekty vykresleny na vložený obrázek, nebo budou navráceny na průhledném pozadí.

```
// Vykreslení objektů.
m_renderer.render();
// Navrácení výsledného obrázku s vloženými objekty.
m_renderer.getOutputImage();
// Navrácení vykreslených objektů na průhledném pozadí.
m_renderer.getOutputImage(false);
```

10 Testovací aplikace

Testovací aplikace je postavena na základě vytvořené kalibrační knihovny. Tato aplikace slouží jak pro testování funkčnosti knihovny, tak i pro prezentaci všech schopností této knihovny. Pro tvorbu aplikace byl využit framework Qt s programovacím jazykem C++. Výsledná aplikace je rozdělena do čtyř hlavních částí (Obr. 10.1), které jsou uzpůsobeny tak, aby práce s touto aplikací byla jednoduchá a přehledná. V následujících podkapitolách se nachází podrobnější popis zmiňovaných čtyř hlavních částí.



Obr. 10.1 Diagram částí aplikace

10.1 Hlavní menu

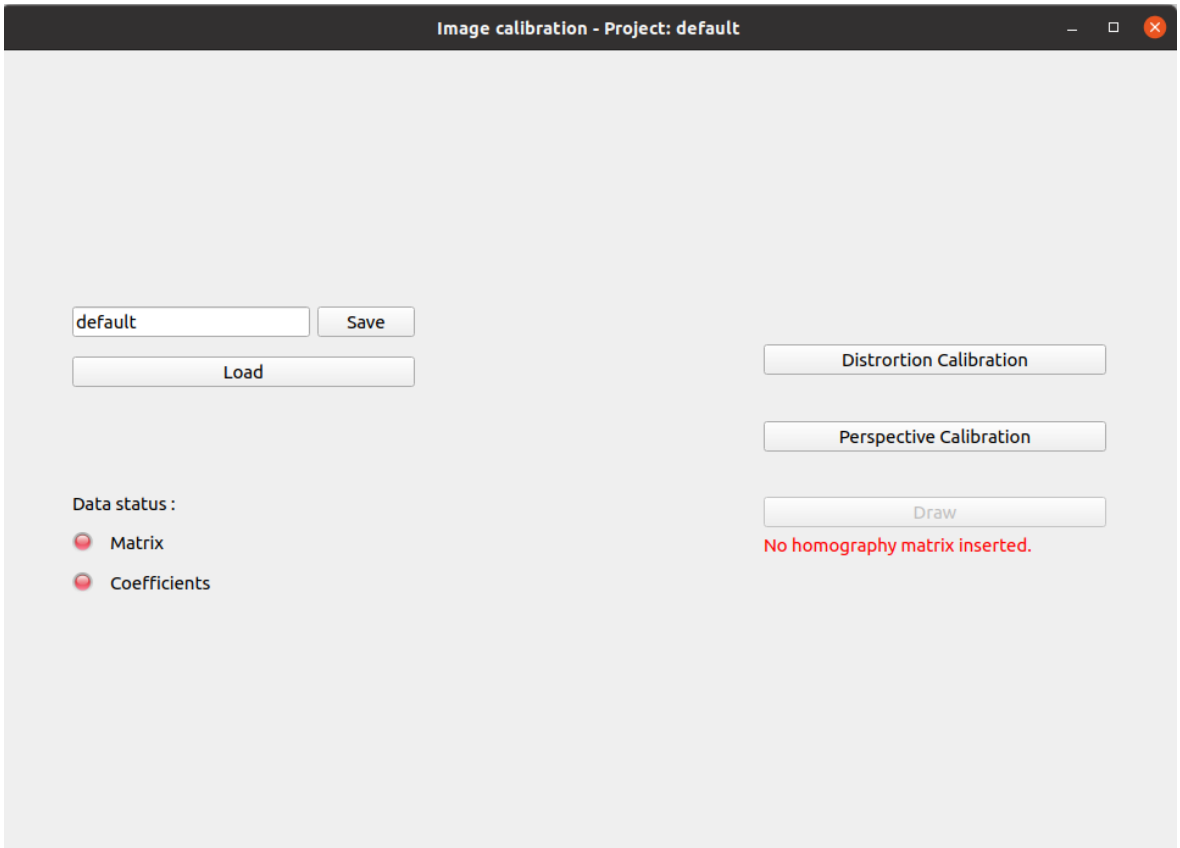
Část hlavní menu slouží pro správu projektů a pro přeskakování mezi ostatními částmi. Zde je možné vytvořit nový projekt, který bude sloužit pro ukládání získaných parametrů.

Pro změnu názvu projektu je nutné vepsat požadovaný název do editačního pole v levé části obrazovky (Obr. 10.2). Poté je možné zmáčknout tlačítko Save, které otevře nové vyskakovací okno, kde je nutné vybrat místo uložení daného projektu. Po potvrzení volby dojde k vytvoření souboru s daným jménem projektu a toto jméno také bude vypsáno do popisu hlavního okna, aby bylo možné z kterékoli části zjistit aktuálně otevřený projekt. V případě, že není vyplněn název nového projektu a je stisknuto

tlačítko Save, dojde k vytvoření nového souboru s názvem default. Stiskem tlačítka Load je možné vybrat již existující projekt, který bude do této aplikace nahrán. Bez vytvořeného či vloženého projektu není možné pokračovat do dalších částí aplikace.

Pod tlačítky pro vytvoření a nahrávání projektu se nachází zobrazení statusu matice a koeficientů zkreslení objektivu. V případě, že dojde v projektu k uložení případně nahrání matice homografie, změní se barva informační diody na zelenou a to stejné platí pro koeficienty zkreslení objektivu. Jakmile se barva diody matice změní na zelenou, zpřístupní se možnost Draw, která slouží pro kreslení do obrazu.

Pomocí tlačítek, které se nacházejí v pravé části obrazovky, je možné přistupovat k dalším částem aplikace. Tyto části jsou popsány v následujících kapitolách.



Obr. 10.2 Ukázka části Main menu

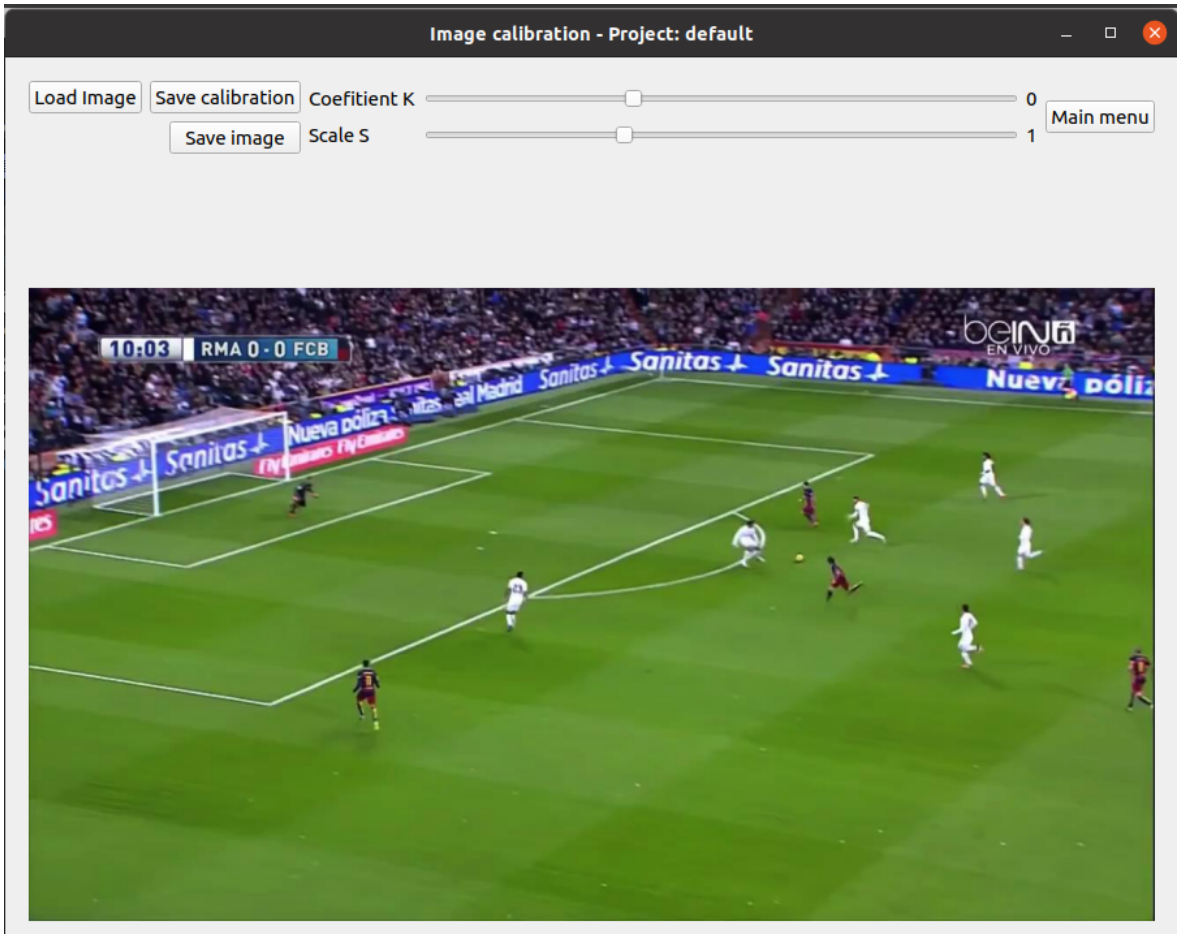
10.2 Odstranění zakřivení objektivu

V této části je možné odstranit případné zakřivení objektivu. Většina kamer, které jsou v dnešní době využívány, již obsahuje nějaký algoritmus pro odstranění tohoto zakřivení a není tedy nezbytně nutné toto zakřivení dále odstraňovat. Může ale nastat situace, kdy daná kamera tuto vlastnost nemá a nebo není plně funkční, a právě proto z tohoto důvodu byla tato část aplikace vytvořena. Pro korekci zkreslení je nejdříve

nutné pomocí tlačítka Load image, které se nachází v levém horním rohu (Obr. 10.3), nahrát vstupní obrázek, který bude upravován. Poté je možné pomocí posuvníku u koeficientu k upravovat obraz tak, aby bylo minimalizováno zakřivení objektivu. Posunem daného posuvníku směrem doprava je možné upravovat sudové zakřivení, posunem doleva dochází k úpravě poduškovitého zkreslení.

V závislosti na velikosti zkreslení se může v okolí obrázku vyskytovat černé pozadí. Toto pozadí je nutné eliminovat, aby nenarušovalo kvalitu kalibrace v následující části aplikace. Pro eliminaci tohoto pozadí je možné měnit hodnotu posuvníku Scale tak, aby toto černé pozadí bylo kompletně eliminováno.

Po dokončení úpravy zkreslení je možné výsledné koeficienty uložit do souboru s projektem pomocí tlačítka Save calibration v levé horní části aplikace. Dále je také možné uložit výsledný upravený obrázek pomocí tlačítka Save image. Pro přechod upět do hlavního menu se zde nachází tlačítko Main menu.



Obr. 10.3 Ukázka části Distortion calibration

10.3 Perspektivní kalibrace

Pro výpočet homografní matice slouží část perspektivní kalibrace. Zde dochází k získání obrazových a mapovacích bodů, které jsou poté využity pro výpočet výsledné matice. Prvním krokem v této sekci je výběr obrázku, který bude použit pro tuto kalibraci. Obrázek je možné vybrat pomocí tlačítka Load image v levém horním rohu (Obr. 10.4). Pokud pro daný pohled kamery existují koeficienty pro odstranění zakřivení a vstupní obrázek obsahuje toto zakřivení, je možné jej odstranit stiskem klávesy Undistort. Po načtení obrázku je nutné zvolit typ požadovaných mapovacích bodů pomocí výběrového menu, které se nachází vedle tlačítka Load Image. V případě výběru mapovacích bodů pro fotbal je nutné specifikovat velikost fotbalového hřiště ve vloženém obrázku. Dále je zde možné specifikovat velikost měřítka mapovacích bodů vepsáním požadovaného zvětšení do políčka, které se nachází vedle textu Point Scale Factor. Jak již bylo zmíněno, při popisu implementace knihovny, se toto měřítko zde nachází z důvodu vykreslování pohledu z ptačí perspektivy. Jelikož předem vytvořené mapovací body jsou udávány v metrech, ponechání meřítka na hodnotě 1 způsobí, že výsledný pohled z ptačí perspektivy bude velmi malý a téměř nečitelný. U měřítka je nastavena výchozí hodnota 10, ale tuto hodnotu je možné měnit v závislosti na rozlišení monitoru, na kterém je aplikace zobrazována. Pro potvrzení vybraného sportu a vepsaných parametrů je nutné stisknout tlačítko Set, které vytvoří požadované body a tyto body také vykreslí do nového okna, které je objeví v pravé části výběrového menu.

V menu výběru sportu se také nachází možnost Custom points, která umožňuje vytvářet vlastní mapovací body. Nový bod je možné přidat vepsáním souřadnic do políček X a Y a kliknutím na tlačítko set. Po kliknutí na tlačítko set dojde k přidání tohoto bodu do mapovacích bodů a tento bod je vykreslen do nového okna, které se objeví v pravém horním rohu obrazovky. Počáteční bod těchto souřadnic se nachází v levém horním rohu, s čímž je potřeba počítat při vkládání těchto bodů. Přidané body je možné vymazat stiskem tlačítka Clear points.

Po vytvoření těchto bodů je možné začít s párováním obrazových a mapovacích bodů. Nejdříve je nutné levým klikem myši vybrat požadovaný mapovací bod, který po výběru změní svoji barvu z červené na oranžovou. V případě špatného výběru je možné tuto akci zrušit stisknutím pravého tlačítka myši a vybrat jiný mapovací bod. Po výběru požadovaného mapovacího bodu je nutné vybrat korespondující obrazový bod. Pozice vybraného bodu je zobrazena ve vstupním obrázku pomocí modré tečky. Korespondující mapovací bod taktéž změní svoji barvu z oranžové na modrou. Tímto způsobem je nutné vybrat minimálně čtyři páry bodů.

Po dokončení párování bodů je možné vylepšit pozici obrazových bodů pomocí tlačítka Improve position of points. Tento proces slouží k přesnějšímu nalezení hran čar

v obraze. V případě nalezení lepší pozice je hodnota původního bodu přepsána. Tento krok není důležitý a je možné jej přeskočit. Pro finální výpočet homografní matice je možné stisknout tlačítko Calculate. Správnost homografní matice lze ověřit zakliknutím volby u Show Birds Eye View, která překreslí vstupní obrázek do pohledu z ptačí perspektivy. Výslednou matici lze uložit stiskem tlačítka Save, což taktéž způsobí zapsání této matice do projektového souboru. Takto vypočítanou matici je možné opětovně použít v případě, že se nezměnila pozice ani vnitřní parametry dané kamery a není tedy nutné před každým kreslením do obrazu vypočítávat pro stejný pohled novou matici.

Obr. 10.4 Ukázka části Perspective calibration

10.4 Kreslení do obrazu

Poslední část aplikace je určena pro kreslení objektů do zkalibrovaného obrazu. Tato část aplikace je přístupná pouze v případě, že pro daný projekt byla vypočítána matice

homografie. Pokud se v projektu nenachází validní matice, je tato část nepřístupná z hlavní nabídky a uživatel je o tomto faktu taktéž informován pomocí textového pole.

Po otevření této části je opět nutné vybrat vstupní obrázek, do kterého budou objekty vykreslovány a to pomocí tlačítka Load image, které se nyní nachází v pravém horním rohu okna (Obr. 10.5). Po načtení obrázku je opět možné odstranit zakřivení objektivu pomocí tlačítka Correct distortion.

Pomocí výběrového menu v pravé části aplikace je možné specifikovat požadovaný typ objektu pro vykreslení spolu s danými parametry tohoto objektu. Téměř všechny typy objektů, kromě vkládání obrázku, mají společné nastavení barvy objektu a šířku čar. Barvu je možné měnit stiskem tlačítka Choose color, které otevře nové okno uzpůsobené pro výběr barvy. Zde se také nachází možnost výběru průhlednosti dané barvy, která se následně promítne do výsledného objektu. Šířku čáry lze specifikovat změnou hodnoty u políčka Line Width. Šířka čáry je udávána v pixelech a musí se jednat o celočíselnou kladnou hodnotu.

Pokud je vybráno vykreslení kružnice (volba Circle), zobrazí se pod tímto výběrem nové pole, které slouží ke specifikování poloměru kruhu, které je opět udáváno jako počet pixelů a musí se tedy jednat o kladnou a celočíselnou hodnotu.

Pro vkládání obrázků do obrazu slouží možnost Image, u které je nutné vybrat požadovaný obrázek tlačítkem Select image. Dále zde můžeme specifikovat hodnotu průhlednosti obrázku, která je reprezentována desetinným číslem od 0 do 1. Dalším parametrem je úhel rotace vstupního obrázku. Funkčnost této rotace je popsána v kapitole 9.1.3, v odrážce Image.

Pozici požadovaného objektu lze specifikovat kliknutím levého tlačítka myši ve vstupním obraze. Horizontální a vertikální čáry společně s kružnicemi požadují právě jeden vstupní bod. Čtverec, obdélník a obrázek potřebují právě 2 obrazové body pro vykreslení. Po kliknutí do obrazu je daný bod vyobrazen pomocí modré tečky. V případě, že je zapotřebí tuto pozici změnit, lze tak učinit stisknutím pravého tlačítka myši, podobně jako tomu bylo u prostorové kalibrace. Po vybrání dostatečného množství bodů pro daný objekt dojde k zpřístupnění tlačítka Render Object, které požadovaný objekt vykreslí do obrazu.

V pravé části obrazu se taktéž nachází tabulka, která slouží pro výpis jednotlivých vykreslených objektů. První sloupec tabulky popisuje typ daného objektu, druhý sloupec zobrazuje barvu daného objektu a třetí sloupec zobrazuje šířku objektu. U všech objektů kromě obrázku je možné upravovat barvu a šířku čar. Pro úpravu barvy je nutné dvakrát kliknout na pole s danou barvou, což způsobí zobrazení nového okna, kde je možné specifikovat novou barvu. Pro editaci tloušťky čáry je opět nutné dvakrát kliknout na pole s aktuální šířkou, což opět způsobí zobrazení nového okna, nyní ale

s možností změny šířky čáry. Odstranění objektu z obrazu lze provést dvojkliknutím na položku DEL u daného objektu. Toto odstranění je nutné potvrdit v nově zobrazeném vyskakovacím okně. K odstranění všech vložených objektů zde slouží tlačítko Clear, které se nachází v pravém spodním rohu vedle tlačítka Main menu. Při stisknutí tlačítka Clear se opět zobrazí nové vyskakovací okno, které slouží k potvrzení tohoto vymazání.

Výsledné objekty lze následně uložit do obrázku pomocí tlačítek Save image nebo Save alpha. Po stisknutí těchto tlačítek se objeví vyskakovací okno, v němž lze specifikovat cestu pro uložení výsledného obrázku. Tlačítko Save image slouží pro uložení obrázku tak, jak je zobrazen v okně, tedy s vloženým vstupním obrázkem. Výsledný obrázek je poté uložen ve formátu JPEG. Tlačítko Save alpha slouží k uložení objektů na transparentním pozadí. Obrázek s transparentním pozadím je uložen ve formátu PNG. Pro návrat do hlavní části se v pravém spodním rohu nachází tlačítko Main menu. Při přechodu do hlavního menu dochází k vymazání všech existujících objektů v obraze.

Obr. 10.5 Ukázka části Draw

10.5 Rozložení aplikace a implementace na Github

Každá výše zmíněná část je složena ze tří souborů, které se nacházejí v kořenové složce aplikace. Každá tato část je reprezentována jako třída, které náleží hlavičkový soubor

s koncovkou .hpp a soubor obsahující implementace s koncovkou .cpp. Nově se zde také pro každou třídu nachází soubor s příponou .ui, který slouží pro tvorbu grafického rozhraní každé části. V tomto souboru jsou například specifikována jednotlivá tlačítka a jejich pozice.

V kořenové složce aplikace se dále nachází třída mainWindow, která slouží pro vytvoření hlavního okna aplikace, do kterého je poté vložena požadovaná část. Aby bylo možné klikáním označovat body v obraze, byly vytvořeny nové třídy imageLabel a mappingLabel. Třída imageLabel slouží pro zobrazování vloženého vstupního obrazu a pro správu kliknutí v tomto obraze. Třída mappingLabel slouží k zobrazování mapovacích bodů na interně vložené pozadí a také se stará o správu kliknutí v tomto obraze. Toto interní pozadí se nachází ve složce Resources, která je poté spravována souborem resource.qrc. Kompletní obsah složky Resources je při instalaci aplikace nakopírován do výsledného spustitelného souboru. Část aplikace určená pro výpočet homografní matice využivá instance obou zmíněných tříd a část určená pro kreslení využívá pouze instanci třídy imageLabel. Pro správu všech důležitých dat v aplikaci byla vytvořena třída calibrationdata, která uchovává všechny získané koeficienty a také vypočtenou matici a zároveň se stará o ukládání a načítání těchto dat.

Do kořenové složky aplikace byly taktéž vloženy všechny soubory vytvořené kalibračního knihovny, které jsou s touto aplikací spárovány v .pro souboru. Zmíněný .pro soubor zároveň slouží pro instalaci aplikace, jejíž postup je popsán v kapitole 10.6

Stejně jako v případě implementace knihovny i u této aplikace byla provedena implementace do nového repozitáře na webové stránce GitHub. Následně zde taktéž byl vytvořen soubor README.md, který obsahuje základní informace ohledně aplikace spolu s postupem instalace. Dále zde taktéž byl vytvořen soubor obsahující licenční podmínky využívání této aplikace, kde opět byla využita licence pro Open source software MIT, která opravňuje kohokoli a jakkoli nakládat s touto aplikací. Zmíněný repozitář je dostupný na webové adrese www.github.com/richardfous/ImageCalibrationGUI.

10.6 Postup instalace

Před instalací aplikace je nutné do daného zařízení nainstalovat knihovnu OpenCV a framework Qt. Postup instalace této knihovny je možné nalézt v oficiální dokumentaci na webové stránce www.opencv.org a popis postupu instalace frameworku Qt je dostupný v oficiální dokumentaci Qt na webové stránce www.qt.io. Pro správnou funkčnost aplikace je nutné využít framework Qt ve verzi 5.12.10 nebo novější.

Po úspěšném provedení instalace knihovny OpenCV a frameworku Qt je možné přejít k instalaci aplikace, kterou je možné stáhnout z GitHub repozitáře na webové adrese www.github.com/richardfous/ImageCalibrationGUI. Testovací aplikaci je možné nain-

stalovat na operačních systémech Linux a Windows. V případě, že je aplikace instalována na operačním systému Windows, je nutné upravit cestu ke knihovně OpenCV, která se nachází v souboru s příponou .pro na 48 a 49 řádku. Po upravení je možné postupovat v instalaci podle návodu v oficiální dokumentaci Qt, který je dostupný na webové adrese www.wiki.qt.io/Build_Standalone_Qt_Application_for_Windows. Pro instalaci na operačním systému Linux je možné využít následující kód, který je nutné spouštět v terminálu operačního systému Linux a zároveň musí být spouštěn v kořenové složce stažené aplikace.

```
qmake && make
```

První příkaz slouží pro přípravu všech důležitých součástí pro instalaci knihovny a druhý příkaz spustí tuto instalaci. Pokud při instalaci nedojde k žádným problémům, vznikne v kořenové složce nový spustitelný soubor s názvem ImageCalibrationGUI. Nově vytvořený spustitelný soubor je možné otevřít pomocí následujícího kódu.

```
./ImageCalibrationGUI
```

11 Ověření funkčnosti knihovny

Pro testování funkčnosti vytvořené knihovny byla využita výše zmíněná testovací aplikace. Testování bylo rozděleno na dvě části. V první části bylo provedeno testování tvorby mapovacích bodů a výpočet homografní matice. Druhá část testování byla zaměřena na kreslení objektů do vstupního obrazu.

Při testování mapovacích bodů a výpočtu homografní matice byly využity fotografie jednotlivých sportů. V případě testování fotbalových bodů byly taktéž nalezeny rozměry fotbalového hřiště, které korespondovaly s testovací fotografií. Pro každý typ sportu byly vybrány příslušné mapovací body, které byly dále využity pro spárování s body ze vstupního obrázku. Kontrola správnosti homografní matice byla následně provedena pomocí vykreslení pohledu z ptačí perspektivy. Tento pohled zobrazí herní plochu překreslenou na černé pozadí a zde je možné kontrolovat pozici tohoto hřiště. Při správném provedení kalibrace budou obrysové čáry hřiště částečně, nebo úplně skryté. Na obrázku 11.1 se nachází příklad takto vykresleného pohledu, který byl vytvořen ze stejného vstupního obrázku jako v případě ukázky výpočtu homografie na obrázku 10.4.

Obr. 11.1 Ukázka pohledu z ptačí perspektivy

Stejným způsobem byly otestovány všechny ostatní dostupné sporty. Při tomto testování nebyly nalezeny žádné chyby, které by měly vliv na funkčnost této části.

Druhá část testování byla zaměřena na kontrolu funkčnosti kreslení objektů. V této části byly využity homografní matice vytvořené v první části testování. Při tomto testování byly nalezeny dvě nové chyby, které způsobovaly špatné vykreslování některých objektů. První chyba se projevila při vykreslování horizontální čáry u některých typů sportů. Tento problém byl způsoben chybou při výpočtu nových bodů horizontální čáry. Druhá chyba způsobovala špatné vykreslování čtverců a obdélníků v případě, že daný objekt přesahoval mimo hranice obrázku. Obě tyto chyby byly odstraněny a upravené soubory byly nahrány do patřičného GitHub repozitáře. Při dalším testování nebyly nalezeny žádné nové chyby.

ZÁVĚR

Cílem této diplomové práce bylo navrhnout a implementovat knihovnu pro automatickou kalibraci prostoru v obrazu z kamery a funkčnost této knihovny otestovat pomocí vytvořené testovací aplikace.

V teoretické části se nachází popis jednotlivých stavebních prvků, které slouží k pochopení dané problematiky. První kapitola se zabývá popisem jednotlivých částí oblasti počítačového vidění, které jsou důležité pro vývoj knihovny. Tuto část je možné rozdělit do dvou kategorií, první se zabývá popisem funkčnosti kamer a popisem převodu získaného obrazu do digitální podoby. Druhá kategorie se zabývá popisem transformací obrazu, jejichž znalost je důležitá pro budoucí vývoj knihovny. Následující kapitola seznamuje čtenáře s open source knihovnou OpenCV a s možnostmi kalibrace v této knihovně. Třetí kapitola je věnována základnímu seznámení s programovacím jazykem C++, který byl využit pro vývoj knihovny. Ve čtvrté části je popsán framework Qt, díky kterému bylo možné vytvořit testovací aplikaci. Předposlední kapitola se zabývá popisem termínu open source. Poslední kapitola obsahuje základní informace o webové stránce GitHub, která je využita pro zpřístupnění kódu výsledné knihovny a testovací aplikace.

V první kapitole praktické části se nachází definice požadavků výsledné knihovny, na jejichž základě byla sestavena architektura knihovny popsaná v kapitole 8. Na základě definovaných požadavků a sestavené architektury byla provedena implementace výsledné knihovny. Výsledná knihovna slouží pro kalibraci perspektivy kamery, zejména pro sportovní utkání, mezi které patří fotbal, hokej, basketbal, volejbal, tenis a badminton. Pro každý sport byly předem vytvořeny body, které odpovídají reálným velikostem hřišť specifikovaných v mezinárodních pravidlech každého sportu. Pomocí jednoduché úpravy kódu knihovny je možné přidat další typy kalibračních bodů. Pokud je vyžadována kalibrace jiného sportu, nebo prostředí bez editace kódu, je možné přidat body manuálně pomocí vytvořených funkcí. Kvalitu výsledné kalibrace může v extrémních případech ovlivnit zkreslení objektivu daného záznamového zařízení. V případě výskytu tohoto zkreslení je možné využít vytvořené metody, která na základě vstupních koeficientů upraví obraz tak, aby bylo toto zkreslení co nejvíce eliminováno. Do zkalibrovaného obrazu je možné následně vkládat jiné obrázky a kreslit různé objekty. Mezi tyto objekty patří horizontální a vertikální čáry, kružnice, čtverce a obdélníky. Všechny zmíněné objekty jsou do obrazu vkládány tak, aby odpovídaly skutečné perspektivě. Kompletní popis knihovny se nachází v kapitole 9 spolu s implementací na GitHub, popisem instalace a ukázkovým kódem.

Na základě sestavené knihovny byla vytvořena s pomocí frameworku Qt testovací aplikace. Tato aplikace je rozdělena do čtyř částí, které jsou chronologicky uspořádány

tak, aby co nejpřehledněji a nejjednodušeji umožnily práci s jednotlivými částmi procesu kalibrace a kreslení. Po provedení kalibrace jsou výsledné koeficienty uloženy do projektového souboru, který lze opětovně využít a na jeho základě je možné kreslit objekty do obrazu. Po dokončení operace kreslení je možné výsledné objekty uložit buďto vykreslené na vložený obrázek, nebo na průhledném pozadí. Kompletní popis a manuál k této aplikaci se spolu s popisem implementace na GitHub a postupem instalace nachází v předposlední kapitole praktické části. Aplikace byla následně využita pro kompletní otestování knihovny, při kterém byly nalezeny 2 chyby v kreslících funkcích a ty byly ihned odstraněny.

SEZNAM POUŽITÉ LITERATURY

[1] SZELISKI, Richard. *Computer Vision: Algorithms and Applications.* London: Springer London, 2011. Texts in Computer Science. ISBN ISBN978-1-84882-935-0.

[2] Understanding Digital Camera Sensors. *Cambridge in Colour - Photography Tutorials & Learning Community* [online]. Copyright © 2005 [cit. 04.04.2021]. Dostupné z: https://www.cambridgeincolour.com/tutorials/camera-sensors.htm

[3] Understanding Color Models: A Review. *ResearchGate | Find and share research* [online]. Copyright ©2011 [cit. 07.04.2021]. Dostupné z: https://www.researchgate.net/publication/266462481_Understanding_Color_Models_A_Review

[4] CorelDRAW X6 Help. *Corel Community* [online]. Dostupné z: http://product.corel.com/help/CorelDRAW/540240626/Main/EN/Doc/wwhelp/wwhimpl/common/html/wwhelp.htm?context=CorelDRAW_Help&file=CorelDRAW-Understanding-color-models.html

[5] Glossary. GIMP Documentation [online]. Berkeley: GIMP, 2019 [cit. 2021-04-07]. Dostupné z: https://docs.gimp.org/2.8/en/glossary.html

[6] HSV – Wikipedie. [online]. Dostupné z: https://cs.wikipedia.org/wiki/HSV

[7] FORSYCH, David A. a Jean PONCE. *Computer Vision: A Modern Approach.* 2nd. ed. Hoboken, New Jersey: Prentical Hall, 2011. ISBN 978-0136085928.

[8] CS231A Course Notes 1: Camera Models *Kenji Hata, Silvio Savarese* [online]. [cit. 07.04.2021]. Dostupné z: https://web.stanford.edu/class/cs231a/course_notes/01-camera-models.pdf

[9] Radial Distortion Correction. *The Bonn Archaeological Software Package - AirPhoto* [online]. Dostupné z: http://www.baspsoftware.org/radcor_files/hs100.htm

[10] A Simple Method of Radial Distortion Correction with Centre of Distortion Estimation. *ResearchGate | Find and share research* [online]. Copyright © Springer Science [cit. 07.04.2021]. Dostupné z: https://www.researchgate.net/publication/220146291_A_Simple_Method_of_Radial_Distortion_Correction_with_Centre_of_Distortion_Estimation

[11] BREEN, David, William REGLI a Maxim PEYSAKHOV. *Computer Graphics: Transformations* [online]. Drexel, 2020 [cit. 2021-04-08]. Dostupné z: https://www.cs.drexel.edu/ david/Classes/CS536/Lectures/L-18_Transformations.pdf

[12] 2D Transformation *Tutorialspoint* [online]. Copyright © Copyright 2021. All Rights Reserved. [cit. 08.04.2021]. Dostupné z: https://www.tutorialspoint.com/computer_graphics/2d_transformation.htm

[13] 3D Transformation *Tutorialspoint* [online]. Copyright © Copyright 2021. All Rights Reserved. [cit. 08.04.2021]. Dostupné z: https://www.tutorialspoint.com/computer_graphics/3d_transformation.htm

[14] CV2.findhomography: Things You Should Know - Python Pool. *Home - Python Pool* [online]. Copyright © 2021 [cit. 08.04.2021]. Dostupné z: https://www.pythonpool.com/cv2-findhomography/

[15] KAEHLER, Adrian a Gary BRADSKI. *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library.* Kalifornie: O'Reilly Media, 2017. ISBN 978-1491937990.

[16] Raspberry Pi Computer Vision Programming - Second Edition | Packt. *Packt | Programming Books, eBooks & Videos for Developers* [online]. Dostupné z: https://www.packtpub.com/product/raspberry-pi-computer-vision-programming-second-edition/9781800207219

[17] OpenCV Block Diagram with supported Operating Systems [13] | Download Scientific Diagram. *ResearchGate | Find and share research*[online]. Copyright © 2008 [cit. 09.04.2021]. Dostupné z: https://www.researchgate.net/figure/OpenCV-Block-Diagram-with-supported-Operating-Systems-13_fig9_341251073

[18] BAGGIO, Shervin EMAMI, David Millán ESCRIVÁ a Khvedchenia IEVGEN. *Mastering OpenCV with Practical Computer Vision Projects.* Birmingham: Packt Publishing, 2012. ISBN 9781849517829.

[19] LISCHNER, Ray. *Exploring C++20: The Programmer's Introduction to C++.* New York: Apress, 2020. ISBN 9781484259603.

[20] History of C++ - C++ Information. cplusplus.com - The C++ Resources Network [online]. Copyright © cplusplus.com, 2000 [cit. 11.04.2021]. Dostupné z: https://www.cplusplus.com/info/history/

[21] C++23. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-04-11]. Dostupné z: https://en.wikipedia.org/wiki/C%2B%2B23

[22] DMITROVIĆ, Slobodan. *Modern C++ for Absolute Beginners: A Friendly Introduction to C++ Programming Language and C++11 to C++20 Standards.* New York: Apress, 2020. ISBN 9781484260463.

[23] LAFORE, Robert. *Object-Oriented Programming in C++. 4th ed.* Carmel, Indiana: CourseSams Publishing, 2001. ISBN 9780672323089.

[24] A.1 — Static and dynamic libraries | Learn C++. *Learn C++* [online]. Copyright © 2021 [cit. 11.04.2021]. Dostupné z: https://www.learncpp.com/cpp-tutorial/a1-static-and-dynamic-libraries/

[25] ENG, Lee Zhi. *Hands-On GUI Programming with C++ and Qt5 by Lee Zhi Eng.* Birmingham: Packt Publishing, 2018. ISBN 9781788397827.

[26] Qt History - Qt Wiki. *Qt Wiki* [online] [cit. 11.04.2021]. Dostupné z: https://wiki.qt.io/Qt_History

[27] What is open source software? | *Opensource.com. Opensource.com* | *Opensource.com* [online]. Copyright ©2021 [cit. 12.04.2021]. Dostupné z: https://opensource.com/resources/what-open-source

[28] Co je open source | Mioweb slovníček webových pojmů. [online]. Dostupné z: https://www.mioweb.cz/slovnicek/open-source/

[29] What Is GitHub? A Beginner's Introduction to GitHub. Kinsta - Managed WordPress Hosting for All, Large or Small [online]. Copyright © 2021 Kinsta Inc. All rights reserved. [cit. 12.04.2021]. Dostupné z: https://kinsta.com/knowledgebase/what-is-github/

[30] What is GitHub. *W3Schools Online Web Tutorials* [online]. Dostupné z: https://www.w3schools.com/whatis/whatis_github.asp

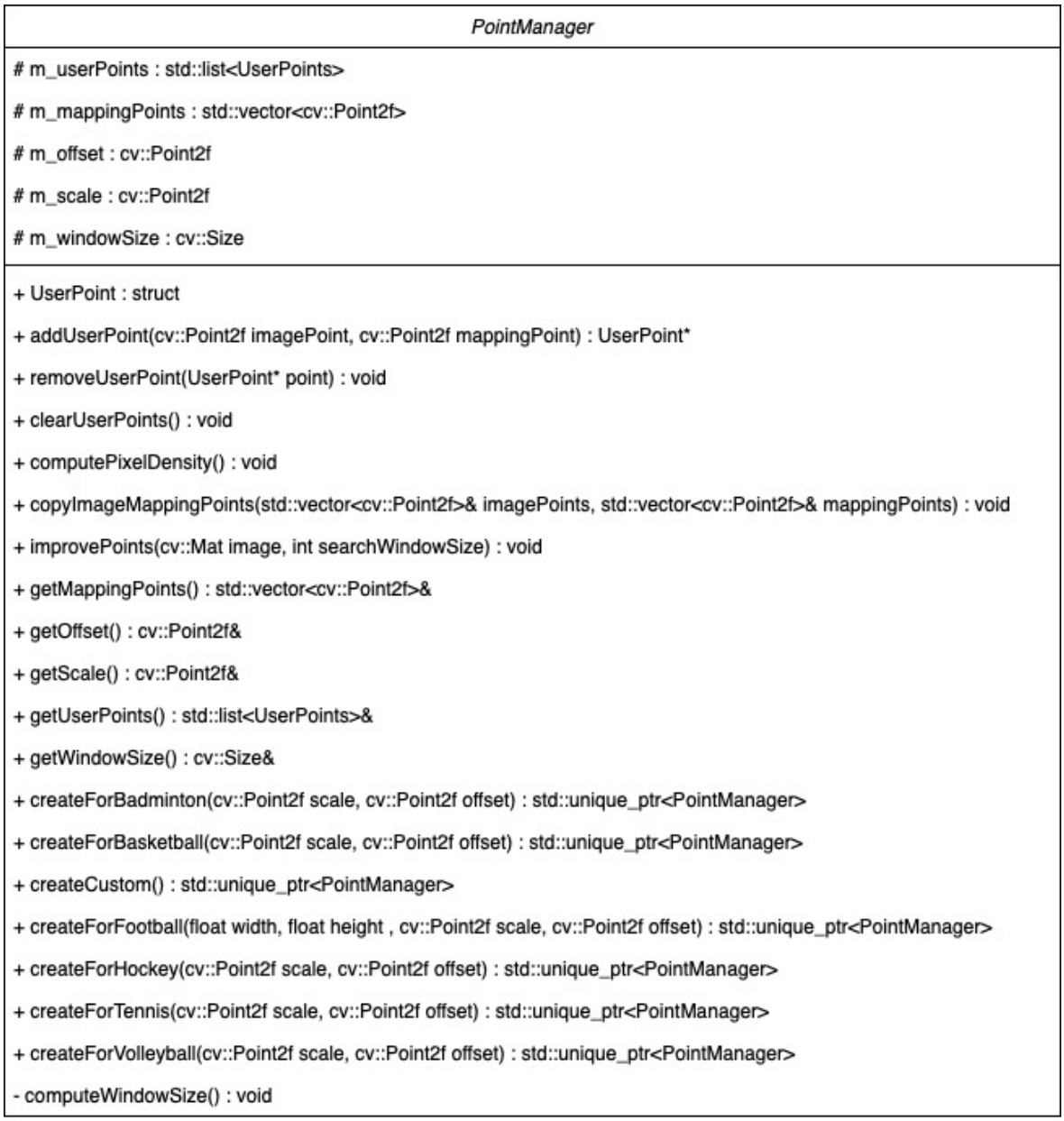
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

3D	Trojdimenzionální
2D	Dvojdimenzionální
OCR	Optické rozpoznávání znaků
ANPR	Automatic number-plate recognition
RGB	Barevný model červená-zelená-modrá
RGB	Barevný model červená-zelená-modrá s průhlednou vrstvou
HSB	Barevný model odstín-sytost-jas
CIELAB	Barevný prostor
RANSAC	Random sample consensus
LMEDS	Least Median of Squares
RHO	Rho-approximation
PROSAC	Progressive Sample Consensus
IPP	Integrated Performance Primitives
MIT	Massachusetts Institute of Technology
CMU	Carnegie Mellon University
INRIA	National Institute for Research in Digital Science and Technology
HAL	Hardware abstraction layer
OOP	Objektově orientované programování
SDK	Software development kit
OSI	Open Source Initiative
OSD	Open Source Definiton
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics

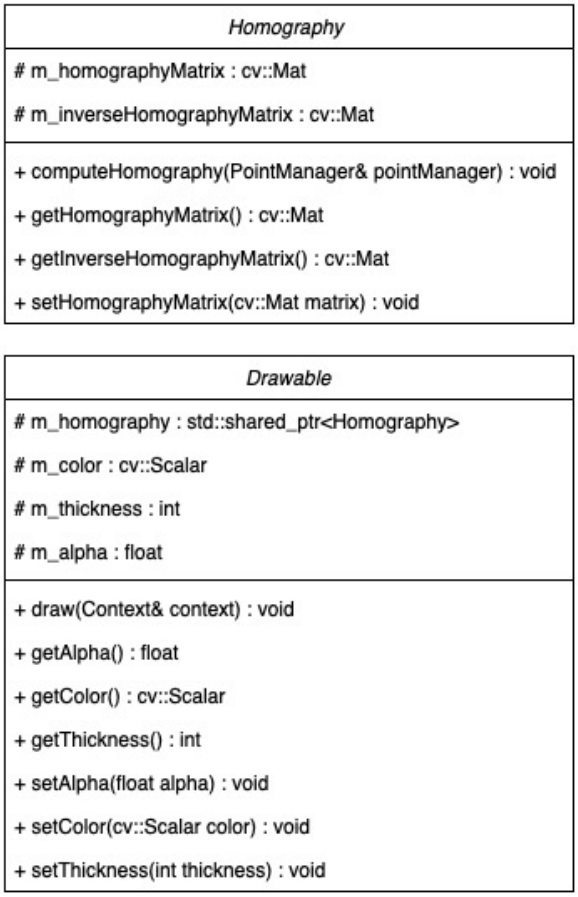
SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

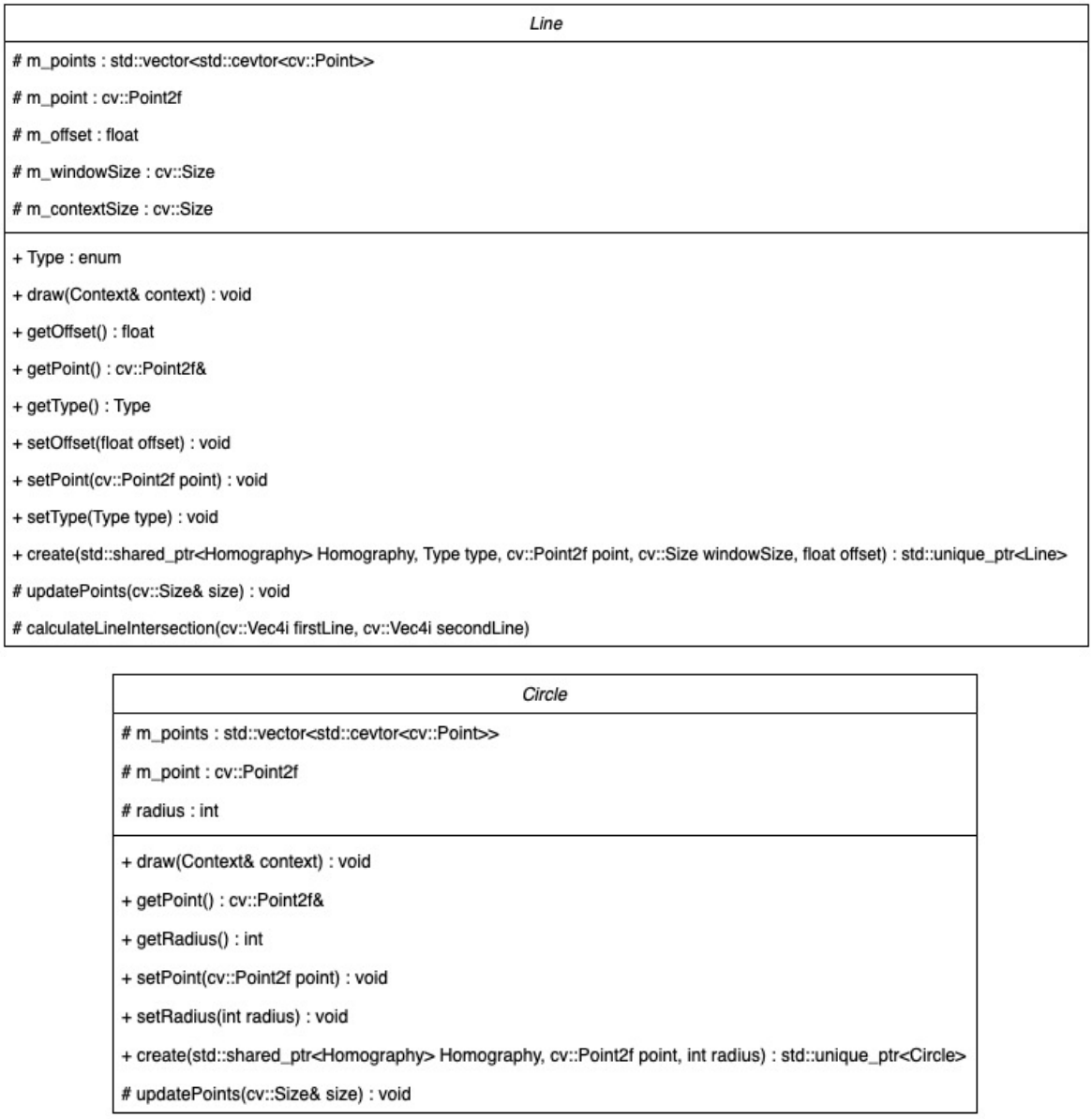
PŘÍLOHA P I. DIAGRAM TŘÍDY POINTMANAGER



PŘÍLOHA P II. DIAGRAM TŘÍDY HOMOGRAPHY A DRAWABLE



PŘÍLOHA P III. DIAGRAMY TŘÍD LINE, CIRCLE, IMAGE, RECTANGLE



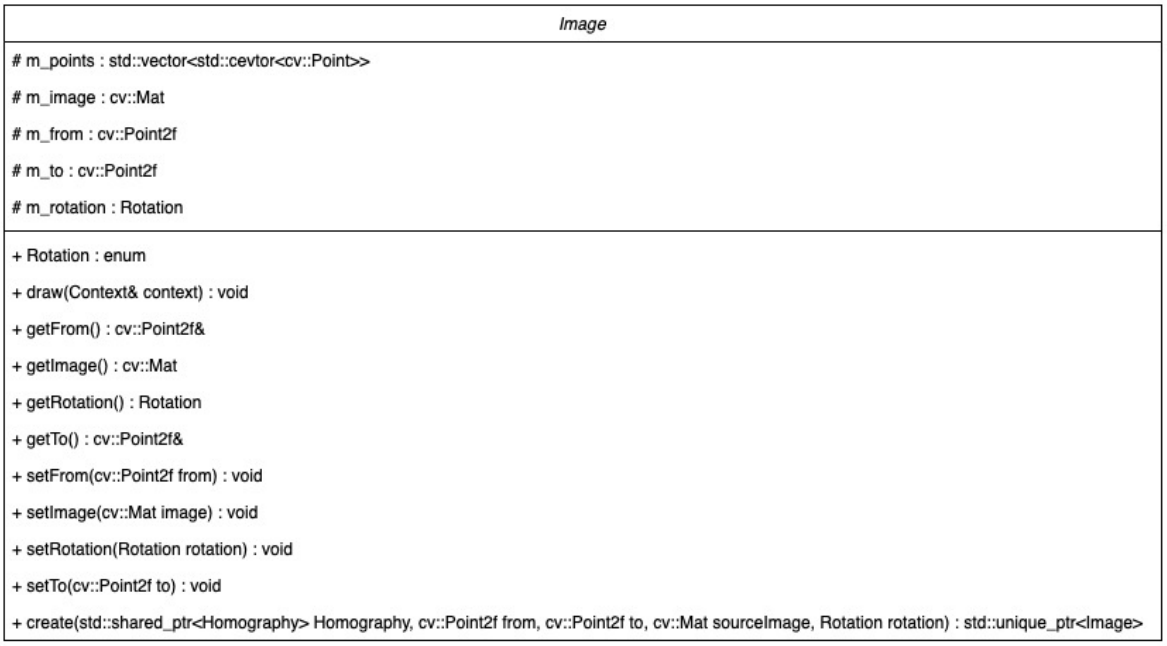
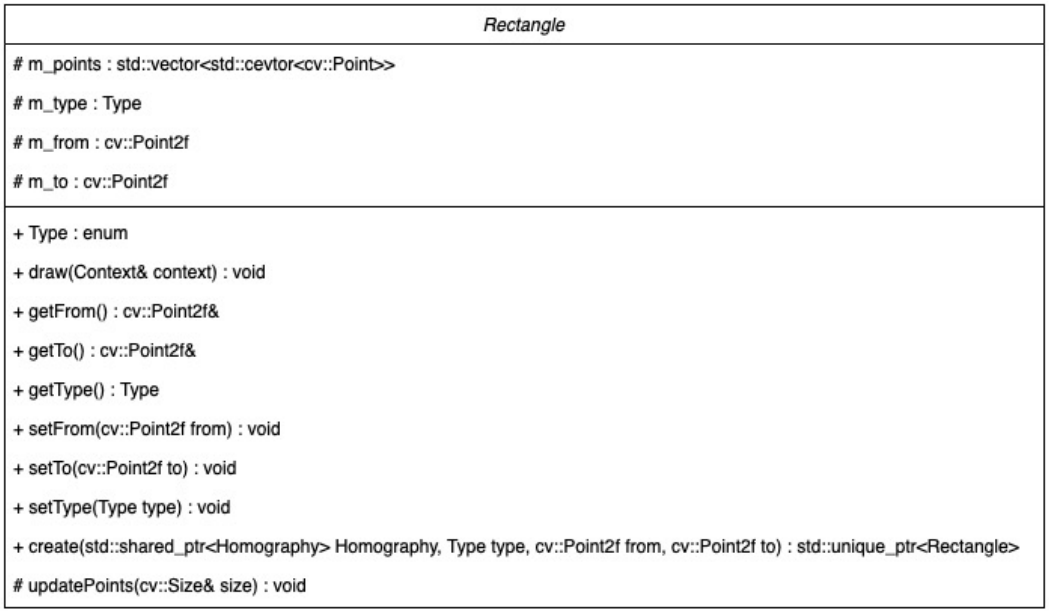


Image
m_points : std::vector<std::cevtor<cv::Point>>
m_image : cv::Mat
m_from : cv::Point2f
m_to : cv::Point2f
m_rotation : Rotation
+ Rotation : enum
+ draw(Context& context) : void
+ getFrom() : cv::Point2f&
+ getImage() : cv::Mat
+ getRotation() : Rotation
+ getTo() : cv::Point2f&
+ setFrom(cv::Point2f from) : void
+ setImage(cv::Mat image) : void
+ setRotation(Rotation rotation) : void
+ setTo(cv::Point2f to) : void
+ create(std::shared_ptr<Homography> Homography, cv::Point2f from, cv::Point2f to, cv::Mat sourceImage, Rotation rotation) : std::unique_ptr<Image>


Rectangle
m_points : std::vector<std::cevtor<cv::Point>>
m_type : Type
m_from : cv::Point2f
m_to : cv::Point2f
+ Type : enum
+ draw(Context& context) : void
+ getFrom() : cv::Point2f&
+ getTo() : cv::Point2f&
+ getType() : Type
+ setFrom(cv::Point2f from) : void
+ setTo(cv::Point2f to) : void
+ setType(Type type) : void
+ create(std::shared_ptr<Homography> Homography, Type type, cv::Point2f from, cv::Point2f to) : std::unique_ptr<Rectangle>
# updatePoints(cv::Size& size) : void

PŘÍLOHA P IV. DIAGRAMY TŘÍD RENDERER A CONTEXT