VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

NÁVRH PROGRAMU PRO OBSLUHU KAMER A PROVÁDĚNÍ STROJOVÉHO UČENÍ
DESIGN OF THE APPLICATION FOR THE CAMERA CONTROL AND MACHINE LEARNING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Jakub Lukaszczyk**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Šimon Bilík**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Jakub Lukaszczyk ***ID:*** 211157

Ročník: 3 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Návrh programu pro obsluhu kamer a provádění strojového učení

POKYNY PRO VYPRACOVÁNÍ:

Úkolem studenta je provést rešerši dostupných programů pro ovládání a nastavení parametrů kamer (jako IC Capture, PylonViewer, nebo JAI Control Tool) a vytvořit podobnou modulární aplikaci pro ovládání kamer fungujících pod standardem GenICam (GiGE Vision a USB3 Vision) s využitím jazyka Python. Tato aplikace také bude umožňovat automatické pořizování a ukládání snímků, dále pak nahrání hotových modelů NS vytvořených v knihovně Tensorflow (přes prostředí Keras) a jejich využití pro učení i pro klasifikaci.

1. Seznámení se s dostupnými programy pro ovládání kamer a jejich řešením, vypracování rešerše.
2. Návrh architektury programu a jeho uživatelského rozhraní.
3. Implementace části programu pro ovládání kamery.
4. Implementace části programu pro nahrání a ovládání předvytvořených NS.
5. Celkové zhodnocení dosažených výsledků a jejich shrnutí.

DOPORUČENÁ LITERATURA:

1. MATTHES, Eric. Python crash course: a hands-on, project-based introduction to programming. 2nd edition. San Francisco: No Starch Press, [2019]. ISBN 978-1-59327-928-8.

2. Dokumentace ke kamerovým standardům GenICam

Termín zadání: 8.2.2021 ***Termín odevzdání:*** 24.5.2021

Vedoucí práce: Ing. Šimon Bilík

doc. Ing. Václav Jirsík, CSc.
předseda rady studijního programu

ABSTRAKT

Tato bakalářská práce řeší návrh programu pro ovládání průmyslových kamer. První část se věnuje současným aplikacím, jejich designu a nedostatkům. V praktické části je poté navržena obdobná aplikace v jazyce Python. Oproti současně dostupným aplikacím poskytuje navrhovaná aplikace modulární a otevřený design a lze ji tudíž dále rozšiřovat a modifikovat. Aplikace je nadále doplněna o vazbu na knihovnu Tensorflow a umožňuje tak klasifikaci obrazu a učení modelů umělých neuronových sítí. Aplikace byla otestována a jeví se být funkční. V závěru práce jsou výsledky zhodnoceny a jsou nastíněny možnosti dalšího vývoje.

KLÍČOVÁ SLOVA

Python, kamera, uživatelské rozhraní, USB3 Vision, GigE Vision, GenICam, počítačové vidění, Tensorflow, Keras

ABSTRACT

This bachelor thesis deals with the design of a program for controlling industrial cameras. The first part deals with current applications, their design and shortcomings. In the practical part, a similar application is then developed using Python. Compared to currently available applications, the developed application provides a modular and open design and can therefore be further extended and modified. The application is further complemented with a link to the Tensorflow library to enable image classification and training of artificial neural network models. The application has been tested and appears to be functional. The thesis concludes by evaluating the results and outlining possibilities for further development.

KEYWORDS

Python, camera, user interface, USB3 Vision, GigE Vision, GenICam, computer vision, Tensorflow, Keras

Vysázeno pomocí balíčku `thesis` verze 4.03; `http://latex.feec.vutbr.cz`

LUKASZCZYK, Jakub. *Návrh programu pro obsluhu kamer a provádění strojového učení*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2021, 65 s. Bakalářská práce. Vedoucí práce: Ing. Šimon Bilík

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Jakub Lukaszczyk
VUT ID autora:	211157
Typ práce:	Bakalářská práce
Akademický rok:	2020/21
Téma závěrečné práce:	Návrh programu pro obsluhu kamer a provádění strojového učení

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno . . . . . 24.5.2021 . . . .

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Šimonovi Bilíkovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Úvod

Kamery a další záznamová zařízení jsou stále důležitějším prvkem průmyslového sektoru. I přes to lze stále nalézt mnoho výrobců kamer, kteří pracují se vzájemně nekompatibilními standardy. V případě, že koncový uživatel používá kamery od různých výrobců, může být komplikované najít aplikaci, která bude korektně komunikovat se všemi těmito zařízeními. Cílem této práce je proto návrh a vývoj modulární aplikace pro ovládání kamer.

V současné době se také stále více rozmáhá využití systémů počítačového vidění, a to jak v průmyslovém sektoru, tak i na úrovni fyzických osob. Pro takovou aplikaci aktuálně existuje velmi málo programů, které by přímo kombinovaly možnost nastavení kamer a zároveň propojení se systémem počítačového vidění. Systémy počítačového vidění umožňují nad nasnímanými daty provést softwarové operace, které v obraze detekují konkrétní objekty nebo vlastnosti bez zásahu člověka. Aplikaci těchto systémů můžeme nalézt v průmyslu, zabezpečovací technice a v poslední době také v zábavním průmyslu.

Aplikace, která je výstupem této práce, bere ohled na tyto požadavky. Program byl napsán tak, aby byl snadno rozšiřitelný o další standardy. Aplikace také implementuje rozhraní pro nahrání modelů neuronových sítí vytvořených v knihovně Tensorflow. Díky tomu je možné přímo v aplikaci zpracovávat obraz pomocí již vytvořených neuronových sítí, případně tyto sítě dále učit.

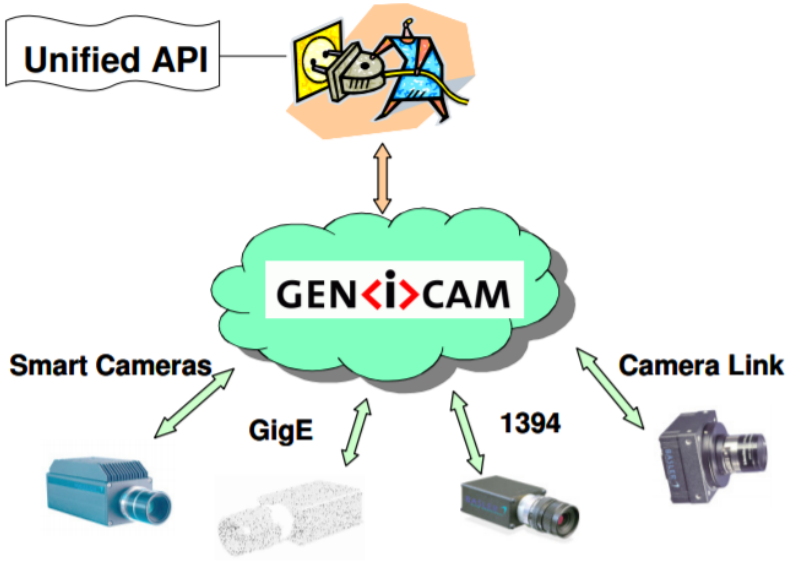
Při návrhu byl brán ohled na maximální uživatelský komfort. Uživatelské prostředí je navrženo podle platných konvencí a pro nového uživatele tak bude snadné se v aplikaci zorientovat a začít využívat všech obsažených funkcí.

Aplikace je napsána v programovacím jazyce Python ve verzi 3.8. Jedná se o jazyk, který existuje již více než 20 let, ale velké popularitě se začíná těšit teprve v posledních letech. Přidání nových funkcí by mělo být pro osobu seznámenou s jazykem Python velmi jednoduché díky modulární povaze aplikace a přiložené dokumentaci.

1 Standardy pro počítačové vidění

V minulosti byla velkou překážkou pro rozvoj aplikací počítačového vidění nejednotnost rozhraní a komunikačních protokolů jednotlivých výrobců. Různí výrobci sice používali například shodné fyzické rozhraní USB 2.0, avšak komunikace s kamerou na softwarové úrovni probíhala bez jakékoliv standardizace napříč průmyslem.

V roce 2006 byla vydána první verze standardu GenICam (Generic Interface for Cameras). Tento standard definuje jednotné softwarové rozhraní a umožňuje tak kamerám pod různých výrobců komunikovat unifikovaně. Myšlenka standardu je znázorněna na obr.1.1. Z tohoto standardu vychází mimo jiné standardy USB3 Vision a GigE Vision, jejichž implementace je obsahem této práce.



Obr. 1.1: Základní myšlenka standardu GenICam [1]

Standard definuje několik základních součástí: GenAPI, GenTL, SFNC, CLProtocol a GenCP. První tři z těchto modulů jasně definují filozofii celého standardu. GenAPI je programovací rozhraní, které představuje referenční implementaci standardu. Pomocí GenAPI lze postavit aplikaci, která není vázána na výrobce. Jinou možností je dle vzoru GenAPI implementovat GenICam komunikaci, která vyhovuje alternativním požadavkům koncové aplikace.

GenTL je modul, který definuje transportní vrstvu a komunikaci mezi kamerami a implementací GenAPI. GenTL bývá definován pomocí tzv. cti souborů, které umožňují výrobcům komunikovat v rámci standardu GenICam jednotně aniž by byli nuceni mezi sebou sdílet vnitřní fungování svých výrobků. Tyto soubory jsou často

zaměnitelné mezi výrobci je-li využit standardní hardware a komunikace mezi kamerou a GenTL je taktéž standardní. Pokud výrobce požaduje, aby jeho kamery šly použít pouze v oficiálních aplikacích, vytvoří proprietární komunikaci na úrovni transportní vrstvy. Na vyšší úrovni zpracování v modulu GenAPI bude zařízení komunikovat již standardně. Pokud není v takovém případě výrobcem poskytnut soubor transportní vrstvy, nelze s kamerou komunikovat jinak než oficiálními nástroji.

Poslední částí, která je nezbytná pro jakoukoliv aplikaci v rámci systému GenICam je SFNC neboli Standard Feature Naming Convention [2]. Tento dokument definuje standardní názvy parametrů, jejich určení, datové typy a zařazení jak do kategorií, tak i do konfiguračních úrovní. Díky tomuto dokumentu je zajištěno, že pokud kamera pracuje pod některým ze standardů odvozených z GenICam, potom budou její parametry dostupné skrze jednotná volání.

Tato architektura tedy umožňuje, jak již bylo nastíněno, výrobcům zachovat své hardwarové implementace a libovolné komunikační metody pro přenos dat mezi kamerou a hostitelským počítačem. Modul GenTL poté zajistí správnou interpretaci těchto dat a jejich prezentaci podle SFNC. V konečné aplikaci, která je implementací GenAPI lze následně s kamerami nakládat libovolně nehledě na výrobce, jelikož všechna zařízení se po průchodu GenTL ovládají jednotně.

1.1 GigE Vision

Tento standard počítačového vidění vychází ze standardu GenICam, díky tomu poskytuje unifikované komunikační rozhraní mezi různými kamerami a uživatelem. Hlavním cílem standardu GigE Vision je vytvořit robustní průmyslové řešení pro přenos obrazu v reálném čase [3],[4].

Obr. 1.2: GigE Vision [5]

Standard strojového vidění GigE Vision [5] používá standard Gigabit Ethernet pro realizaci svého fyzického rozhraní. Tím je zajištěno rozhraní s vysokou přenosovou rychlostí 1Gb/s, ve vyšší variantě až 10Gb/s. Dále je zaručeno nízké zpoždění, a

tudíž jednoduchá realizace aplikací, které vyžadují přenos dat v reálném čase. Velkou výhodou ethernetového rozhraní je také možnost použít vedení dlouhé až 100 metrů, které lze současně využít pro napájení koncového zařízení pomocí technologie Power over Ethernet (PoE).

1.2 USB3 Vision

Stejně jako GigE Vision vychází tento standard přímo ze standardu GenICam a používá tak běžné konfigurační metody. [6],[7] USB3 Vision je standard pro počítačové vidění, který používají zařízení s fyzickým rozhraním USB 3.0, případně s jeho novějšími revizemi USB 3.1 a USB 3.2 [8]. Silnými stránkami tohoto fyzického rozhraní je převážně vysoká přenosová rychlost a plug&play podpora.

Obr. 1.3: USB3 Vision [8]

Cílem standardu USB3 Vision je sjednotit způsob komunikace kamer s počítači skrze sériové rozhraní USB. Před uvedením tohoto standardu na trh měl každý výrobce vlastní komunikační protokol pro komunikaci skrze USB rozhraní. S příchodem USB3 Vision byla tato komunikace sjednocena a všechny kamery pod tímto standardem tak používají shodné názvosloví parametrů. Díky tomu je vývoj univerzálních aplikací mnohem jednodušší.

2 Zadané nástroje

Zadání práce specifikuje programovací jazyk, ve kterém má být práce napsána. Dále je definovaná knihovna Tensorflow a její API Keras, které mají být v aplikaci použity pro provádění strojového učení. API je pojem používaný pro označení aplikačního programovacího rozhraní a většinou jde o knihovnu, která má metody uzpůsobené pro začlenění do dalších programů [9]. Tato kapitola se krátce zabývá historií těchto nástrojů a popisuje jejich specifika a cílené aplikace.

2.1 Python

Python je interpretovaný, objektově orientovaný, vyšší programovací jazyk, který vznikl v roce 1991. V dnešní době se Python těší stále větší popularitě. Obecně je považován za jazyk vhodný pro nové programátory. Je tomu tak hlavně proto, že velkou část správy zdrojů obstarává jazyk na pozadí a programátor nemusí řešit věci jako: alokování a uvolnění paměti, specifikace datových typů, prototypování funkcí a další. Tato zjednodušení si s sebou však nesou jedno úskalí a tím je větší náročnost výsledného programu na hardware a jeho pomalý běh ve srovnání s kódem napsaným v jiných jazycích.

Interpretovaný jazyk potřebuje zvláštní program nazývaný interpreter, který za běhu překládá a vykonává kód. Naproti tomu kompilované jazyky, jako je například C++, vyžadují, aby byl kód před spuštěním přeložen do strojového kódu, kterému počítač rozumí na své nejnižší úrovni.

Objektově orientovaný jazyk staví na myšlence objektu. Objektem v programování nazýváme strukturu, která reprezentuje složitější koncept vzniklý kombinací základních datových typů, metod a také dalších objektů. Díky objektově orientovanému programování lze kód strukturovat, zajistit lokálnost změn a opakovaně využít již napsaný kód.

Pojmem vyšší programovací jazyk obecně označujeme jazyky, které jsou pro člověka čitelnější a pro spuštění vyžadují kompilaci nebo interpretaci, která kód převede do formy čitelné procesorem. Na druhé straně spektra jsou nižší programovací jazyky, které lze přímo číst konečným zařízením. Tento kód je pro člověka těžce čitelný a závislý na použitém procesoru. Nižší jazyky mají blíže k přirozeným instrukcím procesoru. Díky tomu mohou být programy a algoritmy lépe optimalizované a mají tak potenciál být mnohem rychlejší. Definice úrovně jazyka se často mírně liší a v některých případech je mezi nižší jazyky zařazován také jazyk C a C++, jelikož neposkytují automatickou správu paměti.

Zajímavou vlastností Pythonu je dynamická sémantika. Python je silně typovaný jazyk, to znamená, že každá proměnná musí mít přesně definovaný datový typ. Díky

dynamické sémantice je definice typu převedena do interpreteru a programátor tak nedefinuje datové typy vůbec a jejich definici, případně přetypování plně zajišťuje interpreter.

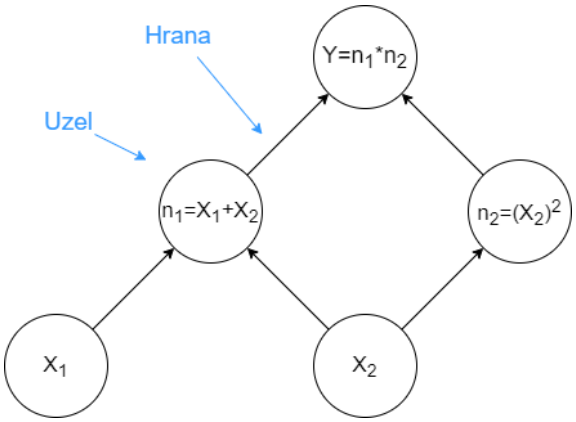
Velký důraz je u jazyka Python kladen na čitelnost kódu. Robert C. Martin ve své knize uvedl, že kód je obecně mnohem častěji čten, než je psán [10]. Také proto existuje soubor pravidel, kterými by se měli programátoři řídit. V případě Pythonu je to obzvláště důležité, protože se počítá s širokým využitím cizího kódu (tzv. modulů). Je-li program psán podle standardních pravidel, dokáže se v něm kdokoliv velmi rychle zorientovat. Soubor těchto pravidel se nazývá PEP (Python Enhancement Proposal) v době psaní práce je ve verzi PEP8 a je dostupný na oficiálním webu Pythonu [11].

Mezi jednu z často komunikovaných nevýhod jazyka patří tzv. GIL neboli Global Interpreter Lock (Globální zámek interpreteru), což je mechanismus, který v tomto jazyce zajišťuje správu vláken. Ve zkratce jde o mechanismus, který znemožňuje vláknům běžet současně na různých jádrech procesoru. Řešení není považováno za ideální, avšak v současné době stále nebyla nalezena alternativa, která by zajistila plnou podporu starších programů a zároveň nesnížila rychlost provádění programů běžících pouze v jednom vlákně.

2.2 Knihovna Tensorflow a Keras

Tato knihovna vytvořená společnosti Google byla ve své první verzi vydána roku 2015. Hlavní oblastí využití této knihovny je implementace metod strojového učení. Je v ní možné vytvářet plně propojené a konvoluční neuronové sítě, dále také nachází využití např. v oblastech zpracování dat, obrazu a porozumění jazyka. Pro svou funkci používá tzv. výpočtové grafy, které se skládají z hran a uzlů viz obrázek 2.1. Skrze hrany data proudí a v uzlech jsou nad nimi prováděny matematické operace. Výpočtové grafy v Tensorflow jsou vždy acyklické (nemohou začínat a končit stejným uzlem), obecně však grafy mohou být také cyklické. Veškeré výpočty v rámci této knihovny pracují s tenzory (uspořádaný n rozměrný soubor čísel, v podstatě zobecnění vektoru).

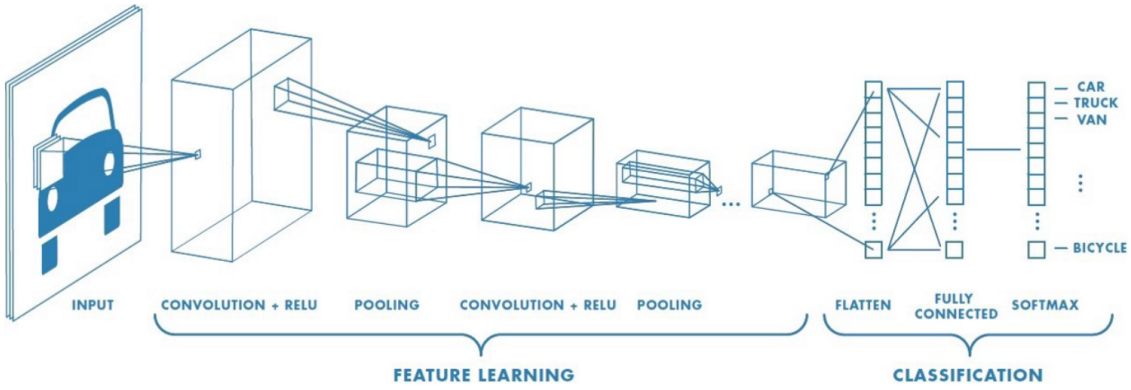
Knihovna Tensorflow je ve své podstatě nízkoúrovňová a její využití pro koncové aplikace je tak relativně náročné a implementace zdlouhavá. Pro tento účel vznikla ve stejném roce knihovna Keras, která zprostředkovává často používané funkce Tensorflow v jednodušší formě. Od roku 2017 je modul Keras oficiální součástí Tensorflow a umožňuje tak plně využít všechny její funkce. Pro většinu aplikací je využití Keras upřednostňováno ať už z pohledu jednoduché implementace, tak i z důvodu přehlednosti výsledného kódu.



Obr. 2.1: Acyklický výpočtový graf

2.2.1 Konvoluční sítě

Tyto sítě se obvykle využívají pro zpracování dat, která jsou přirozeně uspořádaná do mřížky. Nejčastějším představitelem tohoto typu dat je obraz. Oproti plně propojeným sítím, je konvoluční síť schopná příznaky ze vstupních dat extrahovat samostatně. V technické praxi se pojmem konvoluční síť často označuje již kompletní implementace, která obsahuje více vrstev. V takovém případě se konvoluční síť obvykle skládá z vrstev: konvoluční, pooling a plně propojená.

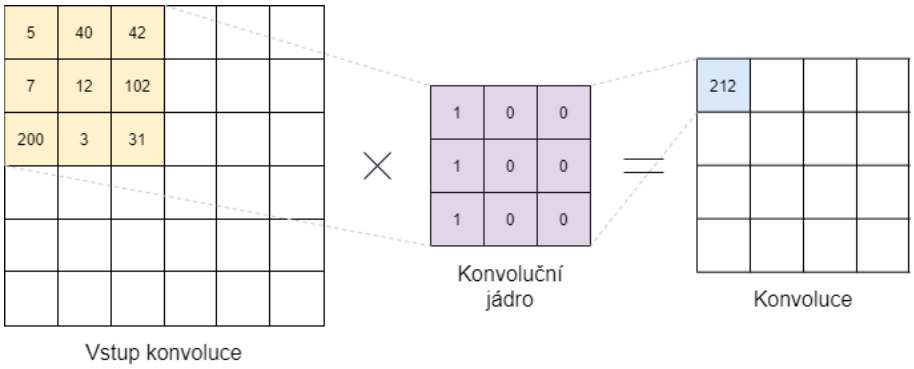


Obr. 2.2: Standardní implementace sítě s konvoluční vrstvou [12]

Konvoluční vrstva

Konvoluční vrstva má za úkol ve vstupním obraze nalézt rysy, které definují určité objekty nebo vlastnosti klasifikované kategorie. V průběhu konvoluce aplikujeme na vstup jeden nebo více filtrů. Filtr neboli konvoluční jádro je prvek, který jednoznačně odlišuje tyto sítě od ostatních umělých neuronových sítí.

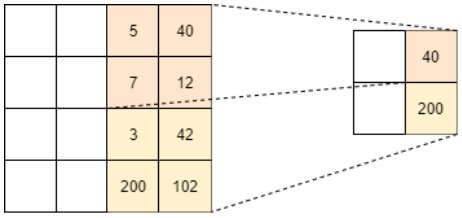
Konvoluční jádro (standardně čtvercová matice) je postupně přikládáno na úseky obrazu s definovaným krokem (stride). V každém kroku jsou vynásobeny prvky jádra s hodnotami obrazu v dané oblasti překrytí, výsledek je sumován a přiřazen výstupu na pozici středu jádra (obrázek 2.3). Čím více odpovídá oblast obrazu obsahu jádra, tím silnější bude aktivace v této oblasti. Cílem učení je tedy identifikovat takové hodnoty matice filtru, které spolehlivě detekují žádaný objekt v obraze, a naopak jsou inertní vůči ostatním objektům.



Obr. 2.3: Princip konvoluce

Vrstva pooling

Hlavním cílem vrstvy pooling je zjednodušení složitého vstupního tenzoru. Tímto docílíme snížení celkového počtu parametrů a rozsáhlosti sítě. Mezi používané metody patří průměrování, detekce svislých čar, maximum a globální pooling [13]. Nejčastěji používaná metoda je max pooling, její princip lze vidět na obrázku 2.4. Metoda spočívá v definování oblasti (např. 2x2), vstup je poté rozdělen na tyto oblasti a v každé oblasti je identifikována nejvyšší hodnota, která je předána výstupu. Tato metoda je jednoduchým způsobem, jak efektivně dosáhnout vyšší rychlosti učení i klasifikace sítě. Dále také do určité míry snižuje náchylnost sítě k přeučení, jelikož principiálně zvyšuje obecnost.



Obr. 2.4: Max pooling

Plně propojená vrstva

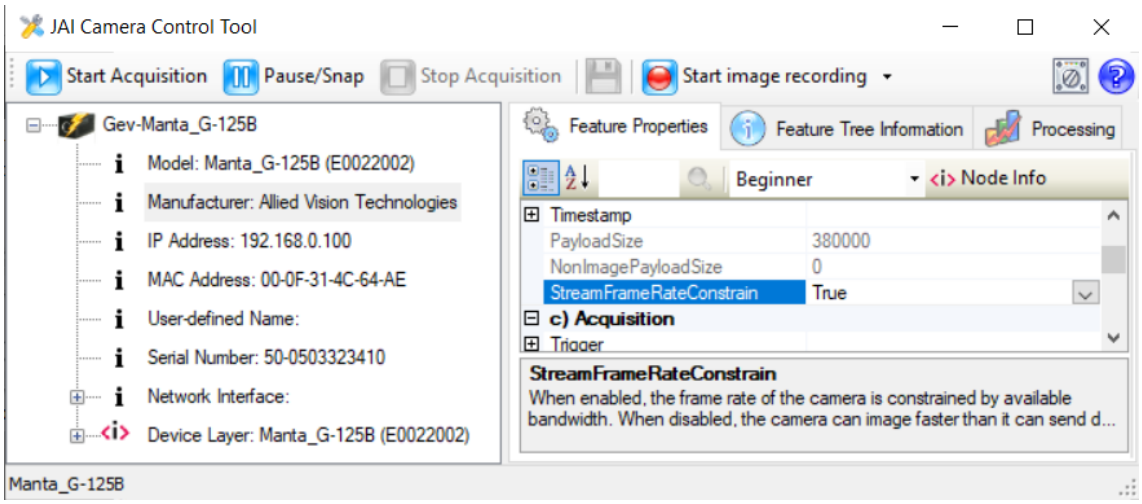
Tato vrstva bývá poslední částí konvoluční neuronové sítě. Jde o klasickou plně propojenou umělou neuronovou síť. Na jejím vstupu jsou již identifikované příznaky detekovaných objektů a úkolem této vrstvy je na základě síly jejich aktivace provést klasifikaci do jedné z výstupních kategorií. Výstup bývá často doplněn vrstvou softmax, která hodnoty rozloží statisticky a výstup normalizuje na rozsah 0-1 neboli 0-100 %. Tuto část sítě lze vidět na obrázku 2.2 v části classification.

3 Rozbor programů pro ovládání kamer

Existuje mnoho různých programů pro ovládání kamer. Tyto programy se liší funkcemi, uživatelským rozhraním a často také cíleným použitím. V rámci této práce byly zkoumány nejčastěji používané programy pro komunikaci s průmyslovými kamerami. Vzhledem ke skutečnosti, že se nejedná o *open-source* programy, je proveden pouze rozbor vlastností, které jsou přímo přístupné koncovému uživateli, vnitřní fungování jednotlivých programů zkoumáno nebylo.

3.1 JAI Control Tool

JAI Control tool je software společnosti *JAI* [14] určený pro odzkoušení fungování kamery. Tento software v současné době není nadále vyvíjen a již pro něj nebudou vydávány další aktualizace [14]. Program podporuje kromě připojení *JAI* kamer také připojení kamer jiných výrobců.

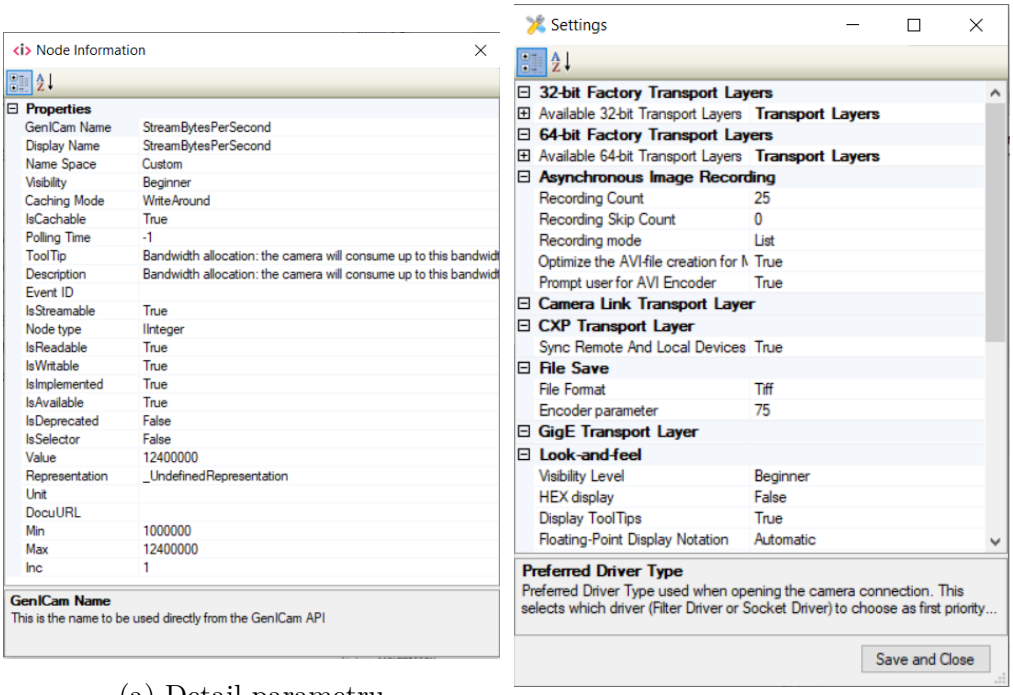


Obr. 3.1: Uživatelské prostředí programu JAI Control Tool [14]

Po spuštění programu můžeme v levé části vidět seznam detekovaných kamer a soupis doplňujících informací (Obr. 3.1). Mezi zobrazované informace patří model, název, výrobce nebo třeba IP a MAC adresa v případě GigE Vison kamer. V pravé části v záložce Feature Properties má uživatel možnost vybrat si úroveň konfigurace od základní po úplnou. V seznamu parametrů jsou poté parametry přehledně rozděleny do kategorií. Jednotlivé parametry lze měnit kliknutím do pole s hodnotou. Pomocí šipky v pravé části lze vyvolat nabídku možností nebo posuvník v případě numerických parametrů.

V horní části uživatelského rozhraní se nachází základní prvky pro ovládání kamery (spustit nepřetržitý náhled, pozastavit/sejmout snímek, zastavit náhled), dále je možnost uložit aktuální snímek nebo spustit, případně nastavit nahrávání (Obr. 3.2b). Náhled obrazu kamery a nasnímaného záznamu se vždy ukazuje v novém okně.

Program JAI Control tool umožňuje čtení a nastavování všech parametrů kamery v souladu s definicí GenICam. Tato parametrizace je možná ve třech kategoriích rozdělených dle předpokládané odbornosti uživatele. Program také poskytuje technické informace o jednotlivých parametrech kamery. Bližší informace o parametru si lze zobrazit kliknutím na *Node Information* (Obr. 3.2a). Aplikace také umožňuje nastavit otočení obrazu kamery nebo zobrazení histogramu.



(a) Detail parametru

(b) Nastavení záznamu

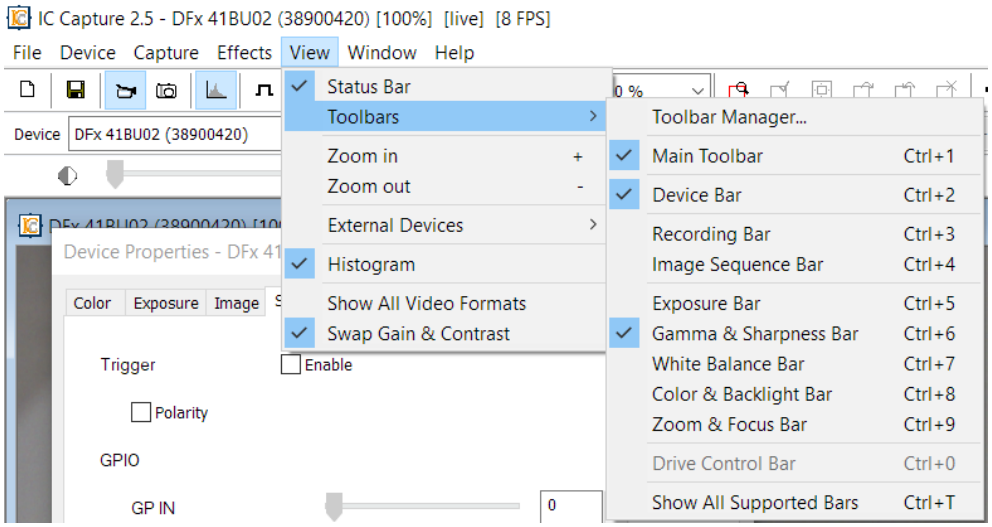
Obr. 3.2: JAI Control Tool - Detail parametru a nastavení záznamu [14]

JAI Control Tool je na první pohled software, který není určen pro koncové uživatele. Je však vhodný pro otestování funkčnosti a komunikace s kamerou. Nevýhodou softwaru je náhled kamery, který se promítá do nového okna, a tudíž se při jakékoliv manipulaci s hlavním oknem přesune do pozadí. Software také není dále vyvíjen a nelze tedy očekávat kompatibilitu s budoucími standardy nebo jejich revizemi [14].

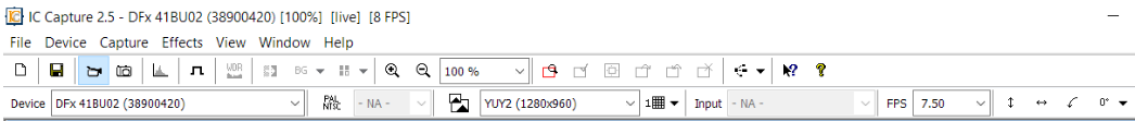
3.2 IC Capture

Tento software, vyvíjený společností *The Imaging Source* [15] je určen pro jednoduché připojení kamer tohoto výrobce. Cílem software je poskytnout uživateli vše potřebné pro práci s kamerou a jednoduché zpracování snímků.

Protože software cílí na jednoduchost, umožňuje pouze základní modifikaci parametrů. Velká část pokročilých parametrů není uživateli zpřístupněna. Dostupné parametry jsou však pro většinu základních aplikací dostačující a jelikož jich je přítomno méně, je software jako celek pro méně pokročilé uživatele lépe použitelný.



Obr. 3.3: IC Capture - Nástrojové lišty [15]



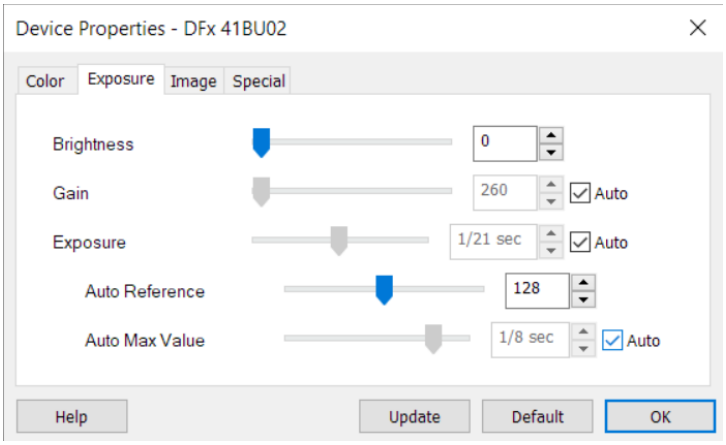
Obr. 3.4: Paleta nástrojů programu IC Capture [15]

V horní liště hlavního okna můžeme vidět nástrojový řádek (Obr. 3.4). Tento řádek umožňuje otevření nové kamery, uložení snímku, spuštění kontinuálního přenosu obrazu, sejmout jeden snímek, zobrazit histogram nebo například zapnout externí spoušť. Další možnosti modifikují přiblížení obrazu a oblast zájmu. Nastavení parametrů kamery se provádí pomocí záložky Device->Device properties (Obr. 3.5). Všechny tyto záložky lze také zobrazit jako nástrojovou lištu přímo v hlavním okně (Obr. 3.3). Rozhraní pro nastavení záznamu (sequence settings) je obdobné jako v případě nastavení parametrů. Pro záznam lze modifikovat cílový adresář, název

souboru nebo také příponu ukládaných snímků a klávesovou zkratku pro zobrazení aktuálně ukládaného obrázku.

V řádku kamery je možné vybrat aktivní kameru, nastavit NTSC nebo PAL formát (je-li funkce podporována), formát obrazových dat, externí spoušť a jiné než výchozí snímkování. Poslední možností je otočení náhledu obrazu v rámci programu.

Obraz připojené kamery se nachází v hlavní části okna a je v samostatném okně. Program umožňuje mít souběžně připojených více kamer. V dolní části programu je stavový řádek poskytující především informaci o množství přijatých a zahozených snímků.



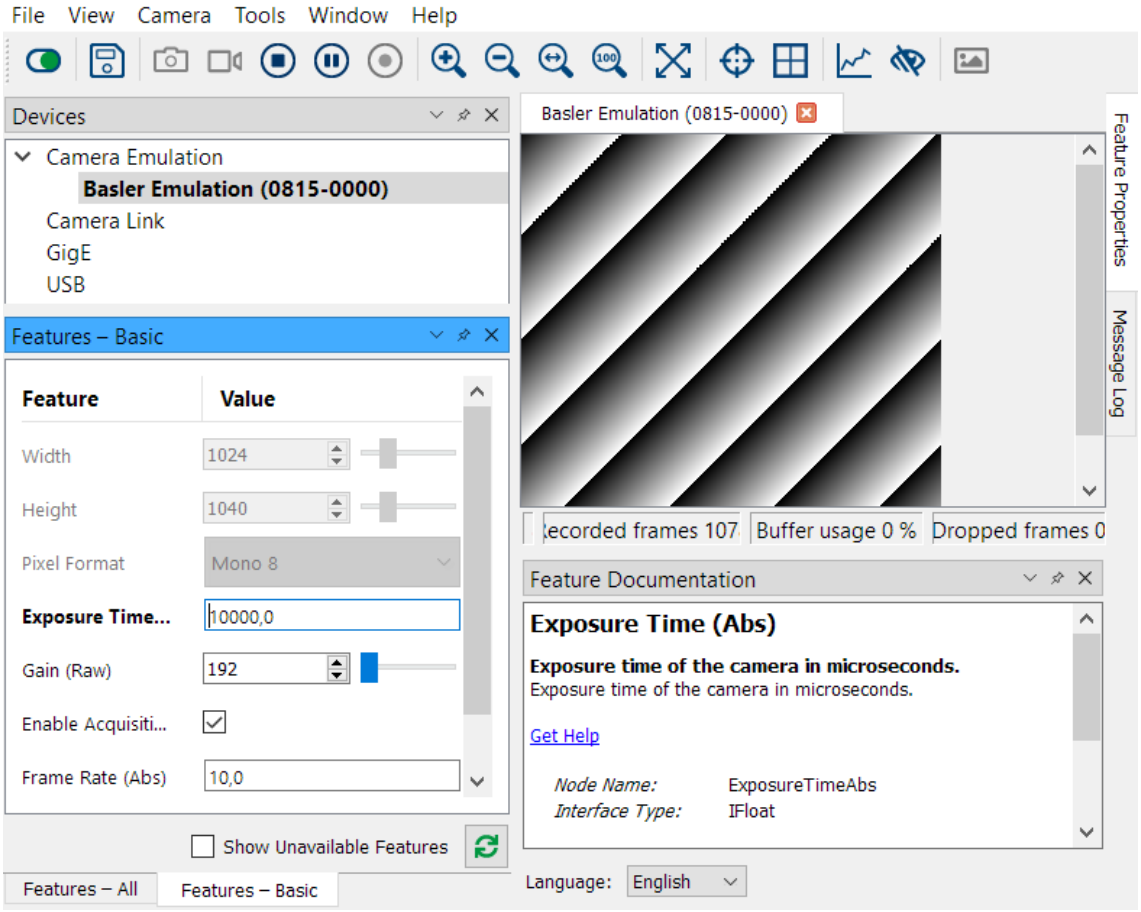
Obr. 3.5: IC Capture - Nastavení parametrů [15]

Program IC Capture je uživatelsky přívětivý a jednoduchý na ovládání. Jako jeden z mála programů umožňuje současné připojení více kamer zároveň. Možnosti konfigurace nejsou natolik rozsáhlé jako v případě jiných programů, ale pro průměrného uživatele budou dostačující. Součástí programu je také velice dobře zpracovaná nápověda. Nevýhodou je podpora pouze kamer výrobce *The Imaging Source*.

3.3 Pylon Viewer

Pylon Viewer [16] je program, který na první pohled působí velmi moderně a čistě. Opět jde o program, který poskytuje dostatek funkcí pro použití v koncové aplikaci. Pylon Viewer, oproti ostatním programům, umožňuje rozsáhlé přeskupení uživatelského rozhraní. Uživatel má možnost přidat nebo odebrat záložky, změnit jejich umístění nebo velikost. Rozložení uživatelského prostředí lze samozřejmě uložit.

V horní liště se nachází základní ovládací prvky (Obr. 3.6). V okně *Devices* lze vybrat jednu z dostupných kamer. Dostupné kamery jsou primárně tříděny podle standardu, avšak tuto možnost lze vypnout. Je-li aktivována možnost *Auto-scan* jsou



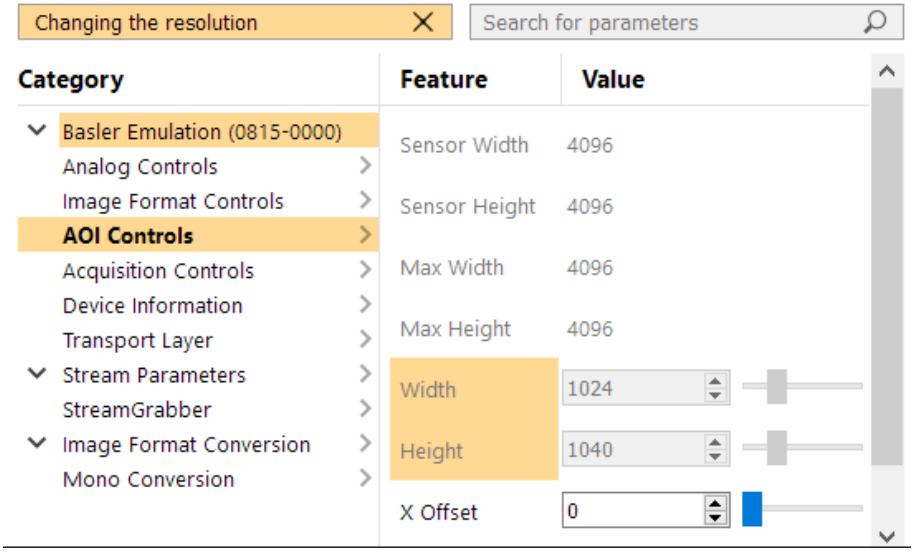
Obr. 3.6: Uživatelské prostředí programu Pylon Viewer [16]

nově připojené kamery okamžitě detekovány a připraveny k použití. V tomto okně lze také vybrat testovací simulovanou kameru pod záložkou *Camera Emulation*.

Velkou část rozhraní zabírá náhled obrazu z kamery (Obr. 3.6). Náhled lze přiblížit nebo oddálit pomocí ikon lupy v hlavní nástrojové liště. Kromě samotného náhledu nalezneme ve spodní části okna informaci o objemu přenášených dat, počtu zaznamenaných snímků a chyb.

Nastavování parametrů je možné pomocí okna *Features*. Podle aktuálně zvolené kamery jsou dostupné záložky s jednotlivými parametry. Výhodou tohoto programu je detailně zpracovaná nápověda, která se navíc zobrazí po kliknutí na kterýkoliv z konfiguračních parametrů. Velmi zajímavá je funkce pro rychlou konfiguraci. Uživatel má možnost si z nabídky vybrat často prováděné konfigurační úkony a aplikace zvýrazní parametry, které je pro daný úkon potřeba měnit (Obr. 3.7). Jedná se o zajímavý způsob, jak novému uživateli pomoci zorientovat se v množství komplexních parametrů. V případě, že chceme provést pouze jednoduchou konfiguraci, využijeme záložku *Features - Basic*, která nám umožní jednoduše měnit nejdůležitější parame-

try právě připojené kamery, tuto skutečnost lze vidět na výše uvedeném náhledu uživatelského rozhraní na obrázku 3.7.



Obr. 3.7: Záložka pro zjednodušenou konfiguraci parametrů [16]

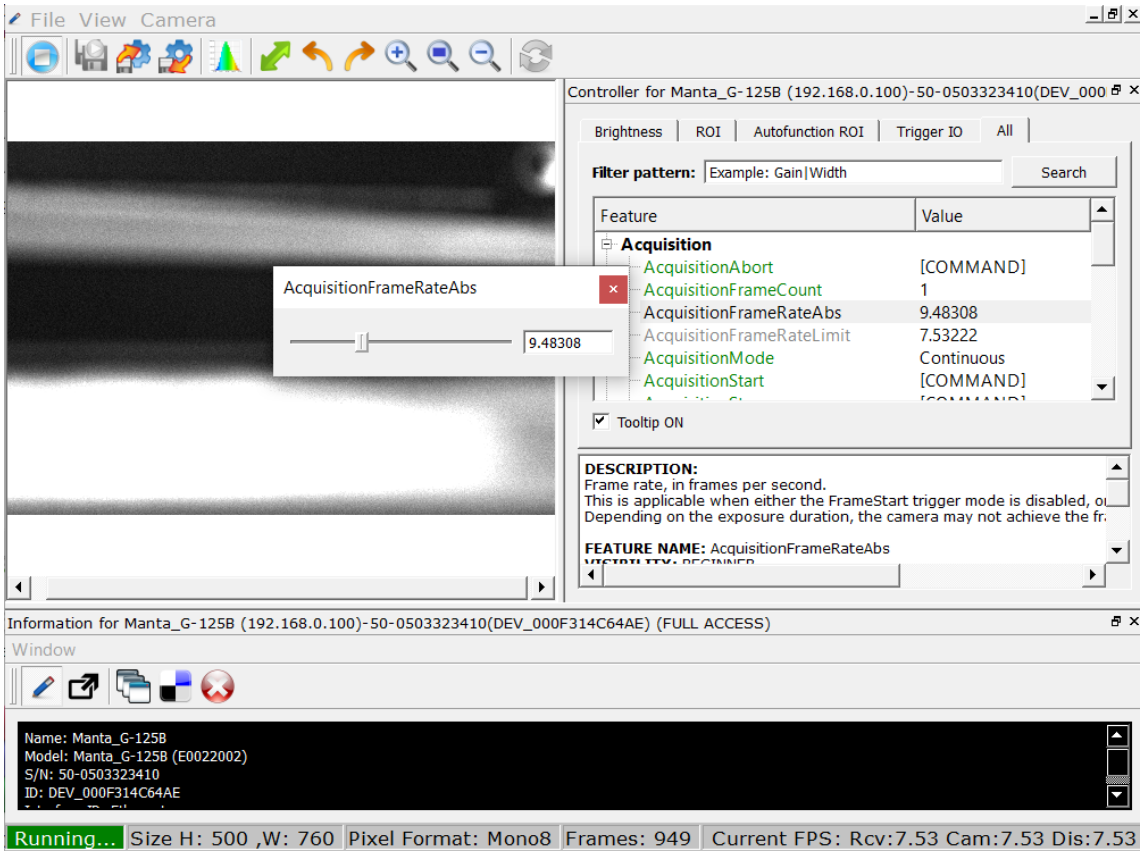
Program Pylon Viewer je ze zkoumaných programů pravděpodobně nejkomplexnější a obsahuje nejvíce funkcí. Zvýraznění parametrů, které je nutné změnit při některém z předdefinovaných úkonů zajišťuje použitelnost aplikace novými uživateli. Zkušení uživatelé přitom nejsou touto funkcí nijak omezováni. Toto návrhové rozhodnutí dává aplikaci silnou pozici na trhu.

Aplikace je vytvořena pro práci s kamerami výrobce *Basler*. Podporovanými standardy jsou: Camera Link, USB3 Vision a GigE Vision. Kamery jiných výrobců v této aplikaci nejsou podporovány. Součástí aplikace jsou také vývojové nástroje Pylon SDK. Tato sada nástrojů je určena pro implementaci komunikace s kamerami přímo do systému, který bude zákazník používat. Dodávané nástroje jsou uzpůsobeny pro použití s programovacími jazyky C, C++ a .NET.

Relativně nová, zato ve stále větší míře využívaná, je knihovna PyPylon pro programovací jazyk Python. Jedná se o napojení na vývojové nástroje jazyka C a je hojně využívaná jak veřejností, tak i zaměstnanci firmy *Basler*. Opět se však jedná o knihovnu, která podporuje pouze zařízení firmy *Basler*.

3.4 Vimba Viewer

Vimba Viewer je software vyvinutý společností *Allied Vision Technologies* [17]. Software slouží pro připojení a testování kamer tohoto výrobce. Dle oficiálního webu [17] jsou podporovány kamery pod standardem USB3 Vision a GigE Vision, na platformě Windows jsou dále podporovány standardy Camera Link a FireWire.



Obr. 3.8: Vimba Viewer - Uživatelské rozhraní [17]

Po spuštění programu je uživateli v jednoduchém okně umožněno zvolit některou z detekovaných kamer. Kamery jsou přehledně rozděleny do kategorií. Po zvolení kamery je otevřeno nové okno se všemi prvky pro ovládání. Náhled uživatelského rozhraní je vidět na obrázku 3.8. Obraz přijímaný kamerou se zobrazuje v hlavní části okna. V záhlaví se nacházejí základní ovládací prvky umožňující spuštění živého náhledu, ukládání snímků, načtení a uložení konfigurace kamery. Další možností je zobrazení histogramu. Levá část okna potom obsahuje veškeré možnosti konfigurace kamery. Záložka *Brightness* umožňuje uživateli jednoduše nastavit zisk kamery, závěrku a podobné základní parametry, uživatel může také nechat na kameře, aby automaticky zvolila nejlepší možnosti. Pokud uživatel požaduje hlubší konfiguraci, využije záložku *All*, která mu umožní modifikaci veškerých GenICam parametrů

kamery. Parametry jsou rozděleny do základních kategorií tak, jak jsou definovány standardem GenICam, konkrétně jeho modulem SFNC [2]. Ke každému parametru je zobrazena krátká nápověda. Pro změnu hodnoty parametru na něj stačí dvojitě kliknout, podle typu se zobrazí posuvník s možností zadat číselnou hodnotu nebo rozevírací nabídka s vypsanými možnostmi nastavení.

Záložky *ROI* a *Autofunction ROI* umožňují nastavení oblasti zájmu, Trigger IO potom umožňuje nastavit externí spoušť připojenou ke kameře pomocí vhodného rozhraní. Spodní část programu zobrazuje aktuální stav kamery, rozlišení, formát obrazových dat a počet přijatých snímků společně s informací o množství snímků za sekundu.

Allied Vision Technologies společně s aplikací distribuuje soubory pro transportní vrstvu GenICam. Dále je součástí API vhodné pro začlenění komunikace s kamerami tohoto výrobce do jiných programů. API je poskytováno pro jazyky C, C++, .NET a Python.

Software Vimba Viewer poskytuje uživateli všechny nástroje potřebné pro ovládání *Allied Vision* kamer. Software v hojné míře informuje uživatele o aktuálním stavu. Uživatel tak má po celou dobu používání softwaru přehled o prováděných úkonech. Uživatelské rozhraní je jasně rozděleno a lze se v něm rychle vyznat díky jednoznačným ikonám.

Nevýhodou je podpora kamer pouze tohoto výrobce. Další potenciální nevýhodou je seznam kamer, který zůstává na pozadí ve vlastním okně. Nutnost mezi okny přepínat může být pro uživatele, který má připojeno více kamer obtěžující.

3.5 Zhodnocení programů

Z rozboru je patrné, že funkcionalita, kterou program poskytuje je silně vázána na jeho cílené použití. Pokud je program vytvořen za účelem profesionálního použití, je obvykle mnohem složitější a poskytuje více možností konfigurace. Na druhou stranu, programy cílené na širší veřejnost mají obvykle čistší design, ale poskytují méně možností parametrizace.

Pro všechny programy je společná možnost jednoduše spustit záznam, v případě průmyslových řešení se ve většině případů nepočítá s uložením ve formě videa, ale jsou ukládány jednotlivé snímky. Uložení samostatných snímků zjednodušuje následné zpracování ať už se jedná o zpracování manuální nebo automatizované.

Vytvořená aplikace bude stejnou funkcionalitu poskytovat, avšak oproti programům konkrétních výrobců bude nutné aplikaci poskytnout některé pomocné soubory pro korektní připojení konkrétních kamer. Bližší informace o tomto mechanismu jsou součástí kapitoly 5.1.1.

Důležitá vlastnost, která programy navzájem odlišuje jsou podporované standardy. Všechny zkoumané programy podporují základní průmyslové standardy jako je USB3 Vision a GigE Vision. Některé dále podporují například CameraLink nebo FireWire. Mimo podporované standardy je také rozdíl v podporovaných výrobcích. Zatímco JAI Control Tool podporuje jakékoliv výrobce, jejichž kamery jsou detekovatelné transportní vrstvou *JAI*, tak Pylon Viewer, IC Capture a Vimba Viewer podporují pouze své vlastní kamery. Tato vlastnost razantně snižuje použitelnost programů v reálných aplikacích.

Navržena aplikace bude podporovat kamery libovolného výrobce za předpokladu, že kamera pracuje pod standardem odvozeným z GenICam a aplikaci je poskytnut vhodný soubor transportní vrstvy. Dále bude povaha aplikace modulární a není tedy problém implementovat komunikaci pomocí dalších standardů. Toto bude možné za předpokladu, že existují vhodné knihovny pro jazyk Python. Způsob této implementace je nastíněn v kapitole 5.1 a příslušných podkapitolách.

Některé programy, jako například Pylon Viewer, udržují veškerou konfiguraci v jednom okně. Jiné programy otevírají pro různé logické celky programu okna vlastní. V navrhované aplikaci bude využito přístupu programu Pylon Viewer a až na souborové dialogy bude veškerá práce s aplikací soustředěna do jednoho okna.

Konfigurace parametrů v programu IC Capture je uživatelsky velmi přívětivá, avšak pro pokročilého uživatele může být nedostatečná, při návrhu bude tudíž zvolen přístup ostatních programů. Uživatel bude mít tedy možnost modifikovat veškeré parametry, které daná kamera nabízí. Tato konfigurace bude prováděná ve třech standardních konfiguračních úrovních.

Nevýhodou zkoumaných programů je absence vazby na systémy počítačového vidění. Klasifikace nebo učení je možná pouze za pomoci externí aplikace. V navrhované aplikaci bude možné přímo provádět klasifikaci živého náhledu a také provádět učení. Učení však vzhledem k nutnosti předchozího roztřídění do kategorií bude možné pouze z dat uložených na disku, a to i v případě, že byla data vytvořena jinou aplikací.

Na základě rozboru lze dojít k závěru, že dostupné programy jsou plně použitelné v aplikacích, kde se nachází pouze zařízení jednoho výrobce. Naproti tomu v případech, kdy jsou zařízení různorodá je nabídka existujících aplikací mnohem užší a tvorbu nové aplikace tak lze dobře opodstatnit. Dále, pokud cílová aplikace požaduje testování umělých neuronových sítí na aktuálních datech, není v současné době moc způsobů, jak tato data zprostředkovat přímo pomocí průmyslových standardů pro počítačové vidění. Vytvořená aplikace se tedy bude inspirovat již existujícími programy v rámci návrhu rozhraní a konfiguračních možností. Navíc však bude vytvořena tak, aby teoreticky mohla pracovat s kamerami libovolného výrobce a při budoucím rozšíření také pod libovolným standardem. Přímo v aplikaci bude možné

provádět učení a klasifikaci pomocí umělých neuronových sítí, což je funkce, kterou rozebírané programy nenabízejí.

4 Návrh koncepce programu

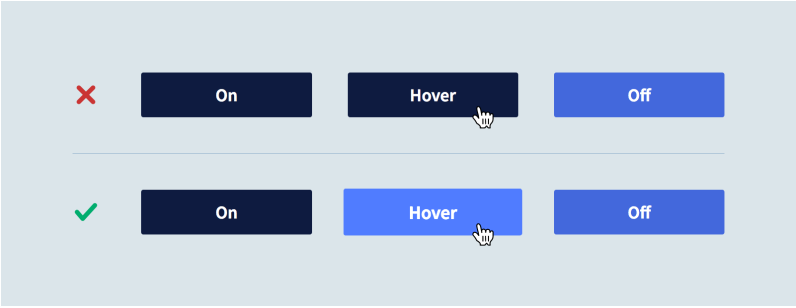
Tato kapitola nastiňuje způsob a kritéria, podle kterých byly zvoleny nástroje a knihovny použité při vývoji. V pozdější části kapitoly jsou poté popsány funkční celky programu a jejich vazba na třídy stejně jako vazba mezi třídami. Bližší informace k vnitřnímu fungování jednotlivých tříd jsou obsahem následujících kapitol.

4.1 Zásady návrhu uživatelského rozhraní

Vytváření uživatelského rozhraní je disciplína, která tvoří rozhraní mezi světem IT technologií a lidským vnímáním. Pro správný návrh je nutné správně ovládat nejen technologické nástroje pro tvorbu rozhraní, ale také mít určitou představu o způsobu myšlení koncového uživatele aplikace, webové stránky nebo jiného produktu s uživatelským rozhraním.

Podle knihy *Don't Make Me Think* od autora Steva Kruga [18] je při návrhu uživatelského rozhraní nejdůležitější nenutit uživatele přemýšlet. Tento princip je dále rozveden v myšlenku, že uživateli se s aplikací mnohem lépe pracuje, když nemá potřebu cokoliv hledat a všechny prvky se nachází na místech, kde je očekává.

Z tohoto principu vyplývá další pravidlo a tím je využití zavedených konvencí v co nejvyšší míře. Lidé v dnešní době používají desítky různých aplikací s různým cílem, avšak určité prvky očekávají jednotné napříč aplikacemi. Příkladem může být ikona pro ukládání, nebo standardní vzhled posuvné lišty a její pozice na pravé straně. Pokud tyto prvky změníme, bude uživatel nucen znovu se učit něco, co už vlastně zná a jeho uživatelský komfort rapidně poklesne. Na následujícím obrázku 4.1 lze vidět, jak je například vhodné tvořit tlačítka.



Obr. 4.1: Vizuální stavy tlačítka [19]

V případě struktury různých nabídek a záložek nezáleží z pohledu uživatelského komfortu na jejich množství, mnohem důležitější je, aby byly jednoznačné a uživatel nebyl nucen metodou pokus-omyl hledat správnou možnost.

Při návrhu aplikace je velice vhodné zpracovat nápovědu, díky které se dokáže uživatel rychle zorientovat v novém prostředí. Při psaní nápovědy je však důležité mít na paměti předcházející pravidla, jelikož pokud je něco nutné v rámci uživatelského rozhraní vysvětlovat, většinou je to způsobeno špatným návrhem, a ne složitostí funkce samotné.

4.2 Výběr knihovny pro GUI

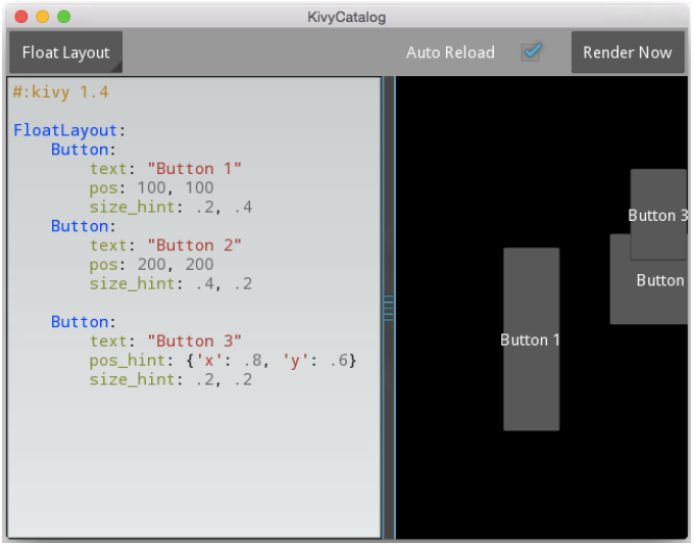
Před samotným návrhem uživatelského rozhraní je nutné zvolit vhodnou knihovnu, která umožní toto uživatelské rozhraní vytvořit. Pro programovací jazyk Python je k dispozici celá řada těchto knihoven, mezi nejčastěji používané patří: Tkinter, Kivy a Qt (C++ knihovna, která je pro Python zprostředkována knihovnou Pyside nebo PyQt). Jelikož navrhovaná aplikace vyžaduje využití vícevláknového zpracování, musí vybraná knihovna tuto funkci co nejlépe podporovat. Důvody pro využití vícevláknového zpracování jsou blíže popsány v kapitole 5.1.1.

4.2.1 Kivy

Kivy je knihovna vytvořená přímo v Pythonu, proto přirozeně podporuje konstrukty, které jsou pro tento jazyk specifické. Velkou výhodou tohoto je podpora široké škály zařízení. Program využívající tuto knihovnu lze jednoduše upravit pro spuštění jak na klasických stolních počítačích (pod OS Windows, Linux i MacOS), tak i na mobilních zařízeních se systémy Android a iOS [20].

Standardně probíhá návrh funkcí jednotlivých elementů uživatelského rozhraní v .py souborech (jazyk Python). Definice vzhledu a umístění prvků probíhá v jazyce Kv, což je jazyk vytvořený vývojáři této knihovny speciálně pro potřeby definice uživatelského rozhraní.

Společně s knihovnou je distribuován nástroj *Kivy Catalog* (obrázek 4.2), který umožňuje do určité míry vytvořit náhled uživatelského rozhraní bez potřeby skutečně aplikaci napsat. I s tímto nástrojem je však návrh uživatelského rozhraní pouze textový a pro tvorbu složitější aplikace s mnoha záložkami, tlačítky a formuláři není z mého pohledu tato knihovna vhodná. Na druhou stranu může být žádoucí při tvorbě multiplatformní aplikace, která má jen jednoduché rozhraní.



Obr. 4.2: Kivy Catalog [21]

4.2.2 Qt

Qt je knihovna napsaná v jazyce C++. Komunitou je velice často využívaná, proto lze v případě problémů nalézt množství online zdrojů k řešení různých problémů. Dokumentace je také velice rozsáhlá a dobře napsaná. Velká výhoda této knihovny je oficiální program *QtDesigner* [22], který umožňuje v jednoduchém uživatelském prostředí navrhnout téměř všechny prvky rozhraní pro novou aplikaci, zobrazit náhled a exportovat kód v C++.

Nadstavba, která se pro zmíněnou knihovnu používá při práci v jazyce Python nese název PyQt [23], opět existuje poměrně dobře zpracovaná dokumentace, avšak jednotlivé názvy metod, tříd a modulů jsou až na naprosté výjimky stejné jako v případě Qt, lze tedy s výhodou použít dokumentaci pro implementaci v C++. Při návrhu aplikace je velmi výhodné použít nástroj *Pyuic5 tool*, který dokáže přeložit uživatelské rozhraní z QtDesigner do jazyka Python. Nástroj je součástí standardní instalace PyQt. Implementace také umožňuje programátorovi použít QThread vláknový objekt této knihovny, avšak podpora standardních Python vláken je také na velice dobré úrovni. Jedinou podmínkou je, aby volání veškerých grafických změn v uživatelském rozhraní probíhalo z hlavního vlákna aplikace, jinak dochází k nahodilým pádům programu.

Pro tento projekt byla zvolena právě tato knihovna, jelikož dobře podporuje vícevláknové zpracování. Velkým bonusem je jednoduchý návrh uživatelského rozhraní díky aplikaci QtDesigner. Dokumentace knihovny je velice rozsáhlá a zkoumání jednotlivých položek a volba postupů je tak mnohem přímočařejší proces.

4.3 Výběr knihoven pro komunikaci s kamerami

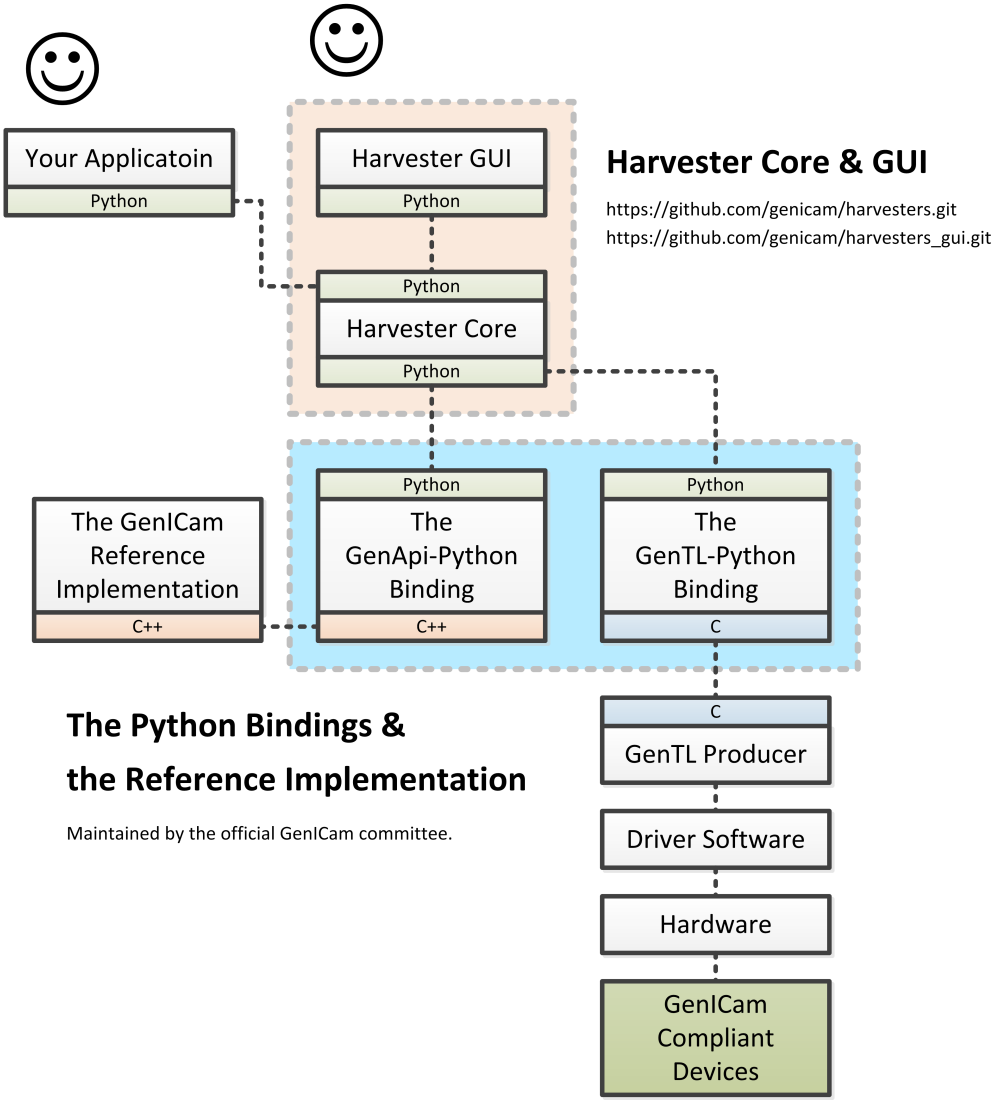
Aplikace musí podporovat různorodou škálu zařízení pracujících pod standardem GenICam a odvozenými standardy. První nabízející se možností, je použít oficiální Python wrapper udržovaný GenICam. Tato možnost by měla zajistit plynulé fungování prakticky všech kamer pracujících pod tímto standardem. Hlavním problémem tohoto řešení je hůře zpracovaná dokumentace a celková složitost využití nízkoúrovňového API. Přímo na stránce projektu [24] je však doporučeno pro koncovou aplikaci využít projektu Harvester.

4.3.1 Harvester

Knihovna Harvester přímo komunikuje s implementací GenAPI a uživateli poskytuje přístup ke kamerám a jejich konfiguraci skrze vysokoúrovňové API. Tato knihovna byla zvolena pro komunikaci s kamerami, jelikož umožňuje jednoduše přistupovat jak ke kamerám pod standardem USB3 Vision, tak GigE Vision. Pro správné fungování komunikace s kamerami je nutné knihovně poskytnout soubor definující transportní vrstvu GenICam (.cti soubory). Tyto soubory jsou poskytovány téměř všemi výrobci. Některé soubory transportní vrstvy jsou obecně použitelné, jiné jsou vázány pouze na kamery konkrétního výrobce. Harvester umožňuje navázání libovolného počtu těchto souborů.

Na hlavní stránce projektu [25] se můžeme dočíst, že ačkoliv knihovna přímo komunikuje se standardem GenICam, ne všechna zařízení fungují korektně. Problémy s komunikací bývají nejčastěji způsobeny přímým zásahem ze strany výrobce, kdy je softwarově zabráněno kameru využívat jinými než oficiálními nástroji. Nekompatibilita však může být způsobena také samotným Harvesterem a jeho interním fungováním.

Obrázek 4.3 popisuje, jak aplikace komunikuje s připojenou kamerou. Koncová aplikace komunikuje pouze s částí knihovny Harvester, která je nazvaná Harvester Core. Harvester Core zajišťuje sjednocení funkcí pro přístup k různým parametrům a rozhraním. Tento modul komunikuje se standardní knihovnou GenICam, která je oficiálně udržovaná vydavatelem standardu. Knihovna GenICam je sice z vnějšího pohledu psaná v jazyce Python, avšak vnitřně je napsána v jazyce C, skrze který také komunikuje s tzv. GenTL Producery. GenTL producer je soubor funkcí a ovladačů, které umožní modulu GenTL komunikovat s kamerou konkrétního výrobce a jeho specifickou implementací na úrovni transportní vrstvy [26]. Úloha modulu GenTL byla popsána již v kapitole 1.



Obr. 4.3: Blokové schéma komunikace pomocí knihovny Harvester [25]

Blok GenTL producer identifikuje kameru, jednoznačně ji označí a skrze unifikované rozhraní ji předá bloku GenICam. Kamera bývá jednoznačně definována následujícími parametry [27]:

- Producer - obsahuje informaci o GenTL Produceru, který byl použit pro přístup k zařízení.
- Interface - udává rozhraní, skrze které je zařízení připojeno. V případě ethernetových kamer jde například o IP a MAC adresu zařízení.
- Device - ukládá jméno kamery. Toto jméno se může mírně lišit v závislosti na GenTL Produceru, který identifikaci vytvořil.
- Stream - obsahuje informaci o všech datových přenosech mezi kamerou a koncovou aplikací.

Vzhledem k občasné nekompatibilitě mezi knihovnou a koncovým zařízením je nutné do aplikace zavést mechanismus, který umožní implementovat proprietární API dalších výrobců. Tím je nejen zajištěna plná kontrola nad kamerami pod standardem GenICam, ale je také relativně jednoduché do aplikace implementovat standardy úplně nové (např. ONVIF). Napojení jednoho takového API je v aplikaci demonstrováno pomocí API Vimba, které distribuuje a udržuje společnost *Allied Vision Technologies*.

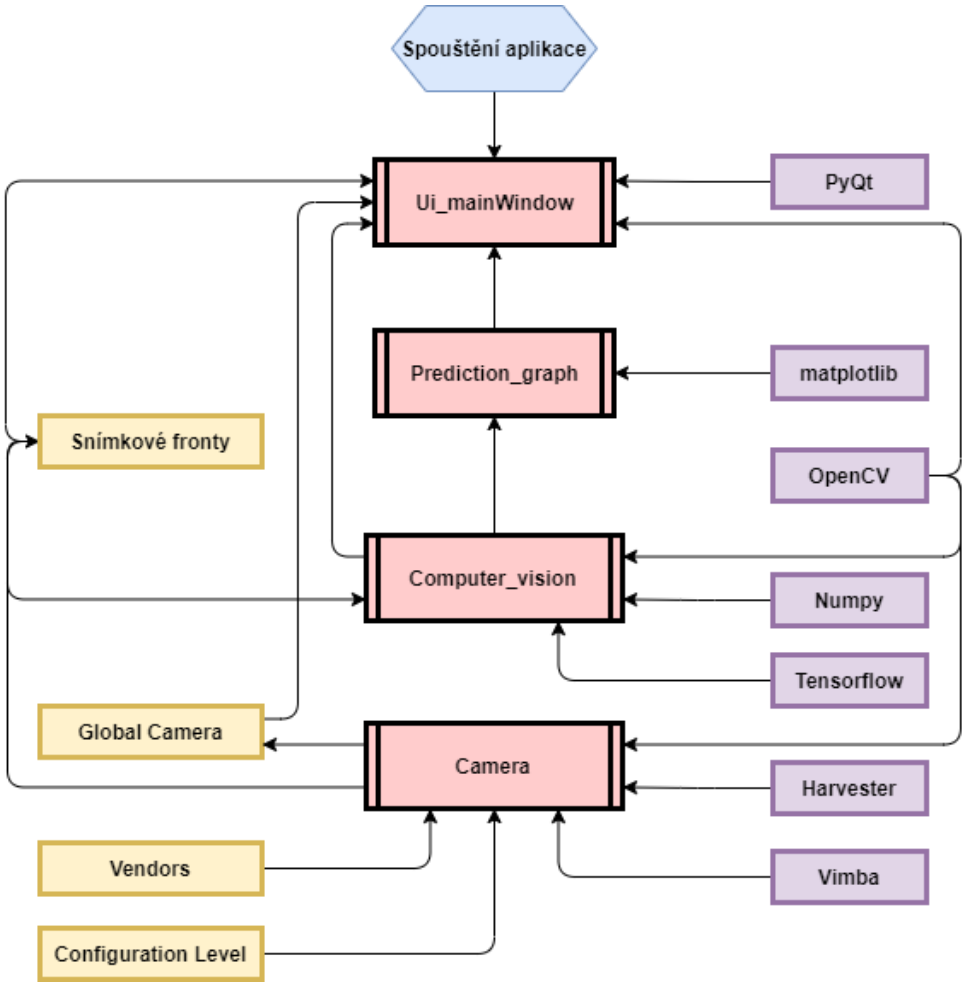
4.3.2 Vimba API

Jedná se o vysokoúrovňové API vytvořené společností *Allied Vision Technologies*, které umožňuje přímou komunikaci s kamerami tohoto výrobce. Knihovna stále interně komunikuje s GenICam a jednotlivé názvy parametrů jsou tudíž shodné jako v případě Harvesteru. Jednotlivá volání pro přístup k těmto funkcím se však liší. Implementovaný mechanismus tedy pro kameru určí, zda se jedná o tohoto výrobce a pokud ano, je upřednostněna implementace Vimba, pokud jde o jiného výrobce, bude použit Harvester. Obdobným způsobem lze implementovat jakékoliv další API nebo standardy, vždy však musí být dodrženy vstupně výstupní formát dat, který je popsán jak v kapitole 5.1.2, tak i v přiložené dokumentaci.

4.4 Základní architektura aplikace

Tato kapitola nastiňuje způsob, kterým spolu komunikují jednotlivé funkční bloky aplikace. Tyto bloky však obsahují množství dalších metod, které jsou detailněji popsány v příslušných kapitolách 5.1 a 5.2.

Těmito bloky je myšlena třída Camera, která zajišťuje komunikaci s kamerou a zprostředkování snímků. Snímky jsou dalším funkčním blokům aplikace předávány jedním z objektů globální fronty. Dalším blokem je třída Computer_vision, díky které je možné v aplikaci zahájit predikci dle načteného modelu nebo jeho učení. Centrální třídou celé aplikace je poté třída Ui_MainWindow, která definuje uživatelské rozhraní a jeho chování. Základní strukturu aplikace lze vidět na následujícím obrázku 4.4.



Obr. 4.4: Architektura aplikace

4.4.1 Hlavní okno aplikace

Část aplikace, která je po spuštění volána je definice uživatelského rozhraní neboli hlavní vlákno knihovny PyQt. Zde jsou vytvořeny instance všech prvků uživatelského rozhraní, dále je zavolána metoda, která inicializuje všechny vnitřní proměnné této třídy. Nakonec jsou na jednotlivé prvky napojeny metody, které jsou zavolány např. po stisknutí nebo zaškrtnutí boxu uživatelem. Veškeré ostatní metody v této třídě definují změny stavu prvků v uživatelském rozhraní, vykreslení snímku nebo třeba prezentaci parametrů přijatých z kamery. V této třídě neprobíhají žádné výpočty a akce, které by ovlivňovaly jiná data než ta, která jsou dále vykreslena. Pokud libovolný prvek vyžaduje komunikaci s kamerou, je volána metoda instance třídy *Camera*, stejně tak v případě požadavku na práci s Tensorflow jsou volány metody instance třídy *Computer_vision*. Jelikož provádění těchto metod je často buďto výpočetně náročné, nebo vyžaduje relativně dlouhé čekání (např. čekání na přijetí

snímku, zapsání snímku na disk nebo provedení učení), je nutné, aby většina těchto metod byla spouštěna ve vlastním vlákně. Pokud by tomu tak nebylo, potom by docházelo k situaci, kdy celá aplikace zamrzne, a to až do doby, kdy metoda dokončí svůj běh. V tu chvíli by nebylo možné např. zároveň přijímat snímky a vykreslovat náhled pro uživatele. Díky využití vláken může aplikace mezi jednotlivými úkoly efektivně přepínat a uživatel tak má dojem, že běží několik procesů současně.

4.4.2 Komunikace s kamerami

Třída *Camera* je odpovědná za veškerou komunikaci s kamerami. Po zvolení kamery si vnitřně uchovává informaci o této zvolené kameře. Informace může mít různou podobu v závislosti na právě použitém API. Implementace Vimba si uchovává informaci o názvu kamery, s kterou pokaždé vytvoří komunikační kanál pomocí kontextového manageru. Naproti tomu Harvester si vytvoří tzv. *image acquifier* objekt, který drží vytvořený po celou dobu připojení kamery a uzavírá jej pouze při volání metody *disconnect_camera*. Společně s informací o kameře samotné je uložena také informace o zvoleném API a ve všech navazujících metodách je poté na základě této informace volen přístup k řešení požadované problematiky.

4.4.3 Provádění strojového učení a klasifikace

Třída *Computer_Vision* obsahuje metody a proměnné potřebné pro veškerou komunikaci s Tensorflow skrze API Keras. Metody zajišťují načtení modelu (cesta je metodě předávána z uživatelského rozhraní) a identifikaci jeho parametrů. Načtení trénovacích dat probíhá v metodě *process_dataset*. Zde jsou data upravena do podoby kompatibilní s načteným modelem. Většinou se jedná o změnu rozměrů a převod barevného obrazu do šedotónového spektra. Mezi další metody této třídy se řadí spuštění učení, které zároveň zajišťuje zpětné volání do hlavního vlákna pro zobrazení průběhu. Dále je implementována metoda pro provedení predikce. V neposlední řadě jsou vytvořeny obslužné metody pro uložení aktuálního stavu modelu.

V programu je také vytvořena třída *Prediction_graph*. Třída je definována jako Matplotlib graf s vazbou na knihovnu PyQt. Do této třídy jsou předávány výsledky predikce obrazu a do grafu samotného jsou zaneseny výsledky predikce až 5 tříd s nejvyšší aktivací. Třída *Ui_MainWindow* si z této třídy vytvoří instanci grafu, kterou poté jakožto další widget vykresluje do uživatelského rozhraní.

4.5 Dokumentace

Aby byla aplikace kompletní, musí obsahovat dokumentaci kódu. V rámci této práce byl zvolen přístup, kdy je celá aplikace, a tudíž i dokumentace, psána v anglickém jazyce. Pro vytvoření dokumentace byl použit nástroj Doxygen. Tento nástroj zajišťuje automatické generování dokumentace přímo ze zdrojových souborů. Výsledná dokumentace je ve formě LaTeX souborů a také ve formě interaktivní html stránky.

Díky dokumentaci je v budoucnu mnohem jednodušší v aplikaci opravit případné chyby, optimalizovat některé postupy nebo implementovat zcela novou funkcionalitu. Dokumentace případnému programátorovi poskytne ucelenou představu o tom, jak aplikace vnitřně funguje a jaké jsou očekávané vstupy a výstupy jednotlivých metod. Vnitřní chování a způsob provádění metod se z dokumentace programátor nedozví. K tomuto účelu slouží komentáře, které přímo v kódu popisují funkčnost jednotlivých bloků a jejich využití uvnitř metod. Jelikož jsou veškeré texty anglicky, není rozšíření aplikace omezeno pouze na programátory ovládající český jazyk.

4.6 Systémové požadavky

Pro zajištění plynulého chodu aplikace je nutné nainstalovat veškeré knihovny obsažené v souboru *requirements.txt*. Navíc je nutné nainstalovat Vimba SDK [17] a případně vývojové sady dalších výrobců v závislosti na použitých kamerách. Aplikace byla z technických důvodů otestována pouze s kamerou *Allied Vision Technologies Manta G-125B* což je kamera pracující pod standardem GigE Vision. Dále bylo zjištěno, že výrobce *The Imaging Source* v současné době neposkytuje soubory transportní vrstvy a připojení těchto kamer tudíž nelze garantovat. Dle vyjádření technické podpory budou tyto soubory dostupné až později během tohoto roku. V případě rozšíření aplikace je také možné využít implementace API jako v případě API Vimba (kapitola 4.3.2), avšak API *The Imaging Source* pro Python je v současnosti také v rané fázi vývoje [28].

5 Vnitřní struktura aplikace

Cílem této práce bylo navržení modulárního programu s uživatelským rozhraním. Tento program má být určen pro práci s kamerami pod standardy USB3 Vision a GigE Vision. Modulární aplikace pracuje tak, že nehledě na typ kamery sama určí vhodný komunikační standard. Uživateli jsou poté kamery pod různými standardy a od různých výrobců prezentovány s jednotným rozhraním. Aby byl kód a výsledný program co nejuniverzálnější a rozšiřitelný, byl celý, včetně dokumentace, psán v anglickém jazyce. Pro co zajištění co nejjednodušší rozšiřitelnosti bylo dodržováno formátování podle PEP8 [11]. Kód byl vhodně komentován a dokumentován. Dále je kód přehledně rozdělen do dílčích souborů dle jejich funkce v rámci výsledného programu.

Aplikace je postavená ze dvou hlavních částí. První částí je *backend*, který je zodpovědný za přímou komunikaci aplikace s kamerami a také za komunikaci s knihovnou Tensorflow skrze Keras API. *Frontend* část (uživatelské rozhraní) se potom stará o vykreslení uživatelského rozhraní a obsluhu veškerých uživatelských akcí. Další důležitou funkcí této části programu je volat funkce *backendu* a návratové hodnoty zprostředkovat uživateli v čitelné formě.

5.1 Backend

Backend zajišťuje komunikaci s kamerami, čtení konfigurací a komunikaci s dalšími moduly. Úkolem *backendu* je data a volání metod různých knihoven sjednotit a vytvořit tak rozhraní, s kterým bude *frontend* komunikovat vždy stejně i v případě implementace nových standardů pro kamery nebo například změny zpracování dat knihovnou Tensorflow.

5.1.1 Zpracování dat

Hlavním prvkem modulární povahy této aplikace je knihovna Harvester [26]. Tato knihovna jednoduše zpřístupňuje funkce standardu GenICam a standardů z něj odvozených. Jak již bylo uvedeno dříve, kód je napsán tak, aby umožňoval jednoduchou a rychlou implementaci API dalších výrobců. Začlenění API sice rozšíří možnosti aplikace a zvýší stabilitu při práci s kamerami konkrétních výrobců, nezajistí však podporu dalších standardů.

Ačkoliv je to nad rámec zadání této práce, tak pro implementaci knihovny nebo API pracující s dalším standardem by bylo nutné kromě standardních úprav pro nové API dále modifikovat následující metody uvnitř třídy *Camera*:

- *get_camera_list()*
- *___translate_selected_device()*

Funkcionalita pro komunikaci s kamerou je implementována v rámci třídy *Camera*, která se nachází v souboru *camera.py*. Tato třída implementuje metody pro spojení s kamerou, čtení parametrů a příjem snímků. Dále je implementována funkce ukládání snímků a konfigurace na disk.

Zpracování snímků pomocí standardního sekvenčního programu je problematická úloha, jelikož snímky z kamery přicházejí velice rychle a jsou v nezpracované podobě. Ukládání těchto snímků je naproti tomu velice pomalý proces, který je dále zpomalen médiem, na které data ukládáme a také časem potřebným pro zpracování těchto dat. Tento problém lze vyřešit implementací vícevláknového zpracování [29].

Pro tento konkrétní případ je vhodné vícevláknové zpracování v takzvaném producer-consumer vztahu. Tento způsob návrhu vychází ze skutečnosti, že jedno vlákno rychle produkuje data, zatímco druhé slouží pouze pro jejich zpracování. Pro zajištění správné funkčnosti je nutné použít mechanismus, který dokáže pracovat asynchronně. K tomuto účelu byla využita fronta, která je již jako modul součástí Pythonu.

Vlákno produkující snímky vždy vkládá přijímané snímky do fronty, která je globálně přístupným objektem v rámci celého programu. Zpracování snímků z fronty probíhá různě v závislosti na použité funkci. Pokud je příjem dat spuštěn za účelem živého náhledu, jsou snímky z fronty čteny metodou implementovanou ve zdrojovém souboru uživatelského rozhraní viz kapitola 5.1.2. Jsou-li snímky ukládány na disk, je použita metoda *start_recording* a snímky jsou následně ukládány na definované místo v rámci počítače. Dále je v uživatelském rozhraní možné nastavit délku záznamu a také formátování názvu ukládaných snímků. Tyto informace jsou poté metodě předány při jejím volání.

Jako unifikovaný formát snímků pro uložení do fronty byl zvolen formát *OpenCV Image* z knihovny OpenCV [30]. Formát byl zvolen, protože knihovna OpenCV je široce používaná a moduly, které se snímky nadále pracují tento formát nativně podporují. Jde tedy o řešení, které je z pohledu budoucí rozšiřitelnosti rozumnější než například definice speciálního formátu vícerozměrného pole. Konverze snímků do tohoto formátu je navíc velice jednoduchá díky knihovně OpenCV samotné nebo například pomocí explicitně definované metody jako v případě Vimba API.

5.1.2 Komunikace s kamerami

Veškerá komunikace s kamerami probíhá ve třídě Camera. Pro využití plného potenciálu kamery je vhodné použít API vydávané přímo výrobcem kamery. Ne vždy je však toto API dostupné, proto je do programu zakomponována knihovna Harvester. Funkce této knihovny byla rozebrána v kapitole 4.3.

Implementace nových API a standardů probíhá výhradně ve třídě *Camera* v souboru *camera.py*. Pro zavedení nového API je důležité v metodách *get_camera_list* a *select_camera* implementovat detekci a volbu kamery. Ve všech ostatních metodách této třídy potom stačí pouze definovat novou *elif* větev, která rozhoduje o volbě implementace na základě obsahu proměnné *vendor* (výrobce). V této větvi je poté možné již libovolně implementovat další API a standardy. Je pouze nutné dodržet jednotný formát výstupních dat definovaný v dokumentaci metod. Obsah proměnné *vendor* měníme vždy při volbě kamery v *select_camera*. Proměnná je nečíselného datového typu a nový záznam pro implementované API zavedeme v souboru *vendors.py*.

V následujících sekcích bude proveden bližší rozbor metod této třídy, jejich funkce v rámci programu a ve vhodných případech bude také blíže popsáno jejich vnitřní fungování. Pro konkrétní informace týkající se korektních vstupů a výstupů je však vhodné využít projektové dokumentace, kterou lze nalézt na přiloženém CD nebo ji lze vygenerovat pomocí nástroje Doxygen přímo z projektového repozitáře [31].

Třída Camera obsahuje kromě metod také řadu proměnných, které jsou důležité pro její správné fungování. Tyto proměnné jsou nadefinovány v rámci metody __init__, kde je jim také přiřazena výchozí hodnota.

5.1.3 Rozbor metod třídy Camera

get_camera_list

Jelikož všechny standardy implementované v rámci této práce spadají pod GenICam a jsou tudíž do určité míry podporovány skrze Harvester, byl zvolen přístup, kdy modul Harvester vždy zprostředkovává rozpoznání připojených kamer a tyto kamery poté vrací jakožto seznam, kdy každý záznam nese informaci o ID kamery, jejím modelu a výrobci. Správná funkce této metody je podmíněna tím, že uživatel třídě poskytne cestu ke vhodným souborů transportní vrstvy. Pro zajištění funkce se standardy úplně novými je nutno implementovat samostatný mechanismus, který bude dodržovat výše nastíněný formát.

select_camera

Vstupem této metody je ID kamery, kterou si uživatel zvolil. Toto ID je poskytováno kamerou samotnou, takže je stejné jak při použití Harvesteru, tak i jiného API. Jelikož Harvester detekuje všechny kamery a například Vimba API pouze kamery *Allied Vision*, index v seznamu nalezených kamer nemusí souhlasit. Proto je volána metoda *_translate_selected_device*, která tento problém řeší. Po zjištění korektního indexu nebo jiné proměnné, která kameru definuje v daném API je proměnná uložena do *active_camera* a dále, pokud je to vhodné, je s kamerou navázána komunikace.

Tato metoda také zajišťuje úpravu MTU (maximum transmission unit) pro ethernetové kamery. MTU definuje maximální velikost rámce, který může procházet sítí a zároveň být zpracován síťovou kartou hostujícího počítače. pokud by MTU nebylo adekvátně upraveno docházelo by k zahazování rámců, které jsou větší než stanovená hodnota. Standardně je maximální definována velikost rámce 1500B, avšak dle konfigurace sítě může být větší [32].

get_parameters

Tato velice důležitá metoda zajišťuje načtení všech dostupných parametrů z kamery a jejich prezentaci uživateli. Jelikož může opět nastat situace, kdy různé implementace prezentují data z kamery různými způsoby, byl definován přesný formát dat pro jednotlivé parametry. Každý parametr je samostatným datovým typem dictionary (slovník), který obsahuje následující klíče:

- *name*
  - Obsahuje název parametru použitý pro přístup k jeho dalším vlastnostem.
- *attr_name*
  - Název parametru ve formě vhodné pro prezentaci v uživatelském rozhraní. Pokud název není nalezen nebo definován je do atributu vložena stejná hodnota jako do *name*.
- *attr_value*
  - Obsahuje aktuální hodnotu parametru.
- *attr_enabled*
  - Pokud je parametr určen pouze pro čtení je tento atribut nastaven na False.
  - Příkladem je například informace o verzi firmwaru kamery.
- *attr_type*
  - Definuje datový typ parametru.
  - V rámci GenICam jsou definovány následující typy: Int, Float, Enum, Boolean, String a Command.

- *attr_enums*
  - Pokud je parametr datového typu enumeration (výčet), jsou do této položky vloženy všechny platné výčtové možnosti.
- *attr_cat*
  - Každý parametr náleží určité kategorii nebo dokonce několika podkategoriím, do tohoto atributu metoda vloží string definující kategorii ve tvaru: *kategorie/podkategorie/...*
- *attr_range*
  - Definuje rozsah platných hodnot pro daný parametr a je ukládán jako list dvou hodnot z nichž první je minimum a druhá maximum.
- *attr_increment*
  - Definuje krok, s kterým jsou hodnoty inkrementovány případně dekrementovány.
- *attr_max_length*
  - Atribut je využit u stringových parametrů a omezuje maximální délku textového řetězce, který kamera pro tento parametr přijme.
- *attr_tooltip*
  - Jde o textový řetězec s krátkou nápovědou nebo popisem parametru, který se uživateli zobrazí po najetí kurzorem na název tohoto parametru.

Z popisů jednotlivých atributů je zřejmé, že ne všechny budou mít definovanou hodnotu pro všechny parametry (např. parametr typu Int nebude obsahovat atribut attr_enums). Pokud atribut neexistuje je mu pouze přiřazena hodnota **None**.

Podle zvolené konfigurační úrovně jsou čteny pouze parametry spadající pod úroveň *beginner, expert* nebo *guru*. Konfigurační úroveň každého parametru definuje dokument SFNC [2].

read_param_value

Použití metody *get_parameters* je vhodné v případech, kdy je nutné vyčíst veškeré parametry z kamery. Těchto parametrů však mohou být desítky a jejich vyčtení a zpracování může zpomalovat běh programu. Pokud je název parametru již znám a je požadována pouze informace o jeho aktuální hodnotě je mnohem vhodnější použít metodu *read_param_value*. Tato metoda vysílá požadavek pouze pro jeden konkrétní parametr a místo slovníku s veškerými informacemi o parametru vrací jen jeho aktuální hodnotu. Tuto metodu je vhodné použít například pro průběžnou aktualizaci hodnot parametrů vypsaných v uživatelském rozhraní.

set_parameter

Jak již název napovídá tato metoda umožňuje uživateli třídy nastavit parametr v kameře na požadovanou hodnotu. Pokud je hodnota z jakéhokoliv důvodu neplatná a nelze ji nastavit, zůstane parametr nezměněn a metoda vrátí hodnotu False, v opačném případě je vráceno True.

Následující metody se volají při požadavcích na spuštění náhledu, záznamu nebo na jiné obrazové operace s kamerou.

start_acquisition a stop_acquisition

Jak již bylo řečeno v kapitole 5.1.1 pro plynulý příjem snímků je nutné záznam spouštět ve vlastním vlákně (metoda *_frame_producer*). O toto se stará metoda *start_acquisition*, která zároveň nastaví příznak *acquisition_running* na True. Příznak třídě říká, že probíhá záznam. Opačnou úlohu zastává metoda *stop_acquisition*. Metoda má za úkol zastavit probíhající vlákno, korektně ukončit záznam a příznak *acquisition_running* nastavit zpět na False. Toto se děje v případě, že záznam již byl spuštěn. Pokud záznam neběží, metoda nedělá nic.

_frame_producer

Tato metoda musí být vždy spouštěna ve vlastním vlákně, jelikož by jinak dlouhodobě blokovala běh celé aplikace. Dle použitého API je spuštěn kontinuální přenos snímků z kamery. Některá API, jako je například Vimba, vyžadují definování metody pro zpracování přijatých snímků (*_frame_handler_vimba*). Harvester je snímky schopen zpracovávat přímo v metodě, kde byl spuštěn jejich příjem. V obou případech je však cílem metody přijímat snímky, převést je do definovaného formátu OpenCV image, přiřadit k nim informaci o barevném modelu a výsledný snímek vložit do snímkové fronty. Zároveň musí při svém běhu metoda libovolným způsobem kontrolovat, zda je již nastaven příznak pro ukončení příjmu snímku a v definovaný okamžik příjem vhodně ukončit.

start_recording a stop_recording

Pokud uživatel požaduje nejen živý náhled z kamery, ale také ukládání jednotlivých snímků na pevný disk, bude volána metoda start_recording a pro následné ukončení záznamu potom stop_recording. Tyto metody volají výše zmíněné start_acquisition a stop_acquisition, ale navíc vytváří vlákno s voláním metody _frame_consumer, která má za úkol zpracovat veškeré snímky ve frontě. Příznak záznamu je proměnná is_recording. Metoda _frame_consumer má za úkol pojmenovat výstupní soubory

dle definované předlohy a opatřit je aktuálním datem, časem, pořadovým číslem nebo jejich kombinací. Předlohu pro název poskytuje uživatel. V rámci této aplikace jde o textový řetězec v definovaném poli v uživatelském rozhraní.

get_single_frame

Pokud uživatel požaduje pouze jeden obrazový snímek, je zbytečné spouštět standardní záznam. Místo toho je volána buďto přímo metoda pro přijetí jednoho snímku nebo je záznam spuštěn a po přijetí snímku okamžitě zastaven. Oproti předchozím metodám není snímek vkládán do frame_queue, ale je ve formátu OpenCV image přímo návratovou hodnotou metody.

save_config a load_config

Metoda pro uložení, respektive načtení konfigurace kamery. Pokud implementované API umožňuje čtení .xml souboru z kamery, potom je konfigurace uložena v tomto formátu a stejně tak je z něj poté načítána. Bohužel, knihovna Harvester v době psaní této práce sice umožňuje načítání konfigurace kamery ve formátu .xml, ukládání však Harvester potažmo GenICam implementace pro Python zatím nepodporuje. Tato skutečnost vedla k vytvoření vlastního formátu pro uložení konfigurace kamery, který sice není standardní, ale pro použití v rámci aplikace je plně dostačující. Samotné konfigurační soubory jsou totiž uložené přímo v kameře a mimo názvy parametrů a dalších prvků definovaných GenICam nemají plně standardizovaný formát. Je proto obtížné vytvořit metodu, která by byla obecně použitelná. Podpora pro export .xml souborů by měla být součástí některé z budoucích verzí implementace GenICam. Jakmile se tak stane, bude metoda aktualizována pro použití .xml konfiguračních souborů exportovaných přímo z dané kamery.

Další metody v této třídě jsou již spíše pomocné a jejich přesná funkce je popsaná v dokumentaci. Tyto metody se starají o přidání a odebrání souborů transportní vrstvy, překlad ID kamery na index pro dané API, provedení příkazu pro parametr typu Command, odpojení Harvesteru a v neposlední řadě také o plné odpojení kamery a uvedení objektu do výchozího stavu.

5.1.4 Provádění strojového učení

Třída *Computer_vision* slouží pro provádění strojového učení za použití knihovny Tensorflow společně s vysokoúrovňovým API Keras, které je přímo součástí distribuce Tensorflow. V rámci třídy jsou nadefinovány proměnné, které si drží aktuálně načtený model, informaci o rozměrech vstupní a výstupní vrstvy a také soubor dat

v podobě vhodné pro přímé použití s načtenou sítí. Rozbor důležitých metod této třídy je uveden níže.

K celé této třídě je ve stejném souboru computer_vision.py nadefinována také třída *Gui_callback*, která definuje komunikaci mezi Tensorflow a uživatelským rozhraním v průběhu učení. Zpětným voláním jsou předávána data o aktuální přesnosti a ztrátě sítě společně s informací o celkovém postupu učení. Tato třída vznikla úpravou standardní implementace zpětného volání definovaného přímo v Tensorflow.

process_dataset

Před tím, než je uživateli umožněno spustit samotné učení, je nutné zvolený soubor dat převést do vhodné formy. Metoda si společně s metodou _create_training_data zjistí vstupní rozměry načtené sítě. Za pomoci knihovny OpenCV postupně změní rozměry veškerých obrázků ve složce s daty a takto upravená data si společně s informací o kategorii uloží do seznamů jako hodnoty vzoru a požadované hodnoty. Aby metody pro přípravu učících dat fungovaly správně je nutné, aby ve zvoleném adresář byly podadresáře, z nichž každý obsahuje sadu obrázků jedné kategorie. Metoda na svém vstupu přijímá referenci na list, do kterého jsou ukládány informace o průběhu zpracování. Tím je umožněno zpětné volání do hlavního vlákna aplikace a prezentování průběhu uživateli.

train

Jakmile jsou učící data připravena, je uživateli umožněno spustit proces učení právě pomocí této metody. Dle hodnoty definované uživatelem je soubor dat rozdělen na trénovací a validační část a poté je spuštěno samotné učení pomocí metody *fit*, která je definována přímo v API Keras. Počet učících cyklů je dán vstupním argumentem metody.

classify

Pokud je uživatelem načtena síť, která je již naučena nebo byla doučena přímo v rámci programu a uživatel chce otestovat její odezvu na datech přijímaných v reálném čase z kamery, potom je volána tato metoda. Na základě výstupu funkce predict (Keras API) je do objektu *Prediction_graph* poslán výstup sítě a v rámci objektu je vybráno až 5 tříd s nejvyšší aktivací. Třídy jsou potom do grafu vykresleny a prezentovány uživateli.

5.2 Frontend

V této kapitole je proveden rozbor metod a celkového fungování uživatelského rozhraní. V rámci třídy *Ui_MainWindow* je nadefinováno velké množství metod, ale podstatná část slouží pouze pro kontrolu stavu příznaků a následné volání jiné, složitější, metody, která je již popsána jinde. Z tohoto důvodu jsou zde popsány jen rozsáhlejší metody nebo metody jejichž řešení je z technického pohledu zajímavé nebo nestandardní. Ostatní metody a jejich popis je samozřejmě součástí dokumentace na přiloženém CD.

5.2.1 Rozbor metod

setupUi a retranslateUi

setupUi a *retranslateUi* jsou metody, které jsou generovány pomocí *pyuic5 tool* přímo z QtDesigner viz kapitola 4.2.2. Tyto metody vytváří samotné uživatelské rozhraní, definují vztahy mezi jednotlivými prvky a jejich popis. Jakmile třída obsahuje tyto metody, program lze již spustit a nahlédnout na uživatelské rozhraní, aby jednotlivé prvky po stisknutí vyvolávaly očekávané akce, je nutné je na tyto akce napojit, to je z důvodu přehlednosti kódu provedeno ve vlastní metodě *connect_actions*.

connect_actions

Knihovna Qt pro každý element uživatelského rozhraní definuje tzv. signály, které jsou vyslány po změně stavu daného prvku. Mezi tyto signály patří například stisknutí tlačítka, změna záložky nebo třeba vložení textu do textového pole. Právě na tyto signály jsou napojena volání jednotlivých metod pro obsluhu uživatelského rozhraní.

tab_changed

Pomocná metoda, která je napojena na změnu některé z hlavních záložek uživatelského rozhraní. Jakmile uživatel přepne záložku je z důvodu šetření výpočetních prostředků vhodné přestat vykonávat některé metody. Příkladem může být živá predikce pomocí konvoluční sítě, kterou je bezvýznamné provádět v době, kdy uživatel nemá možnost vidět její výstup.

preview

Kdykoliv je vyslán požadavek na zahájení náhledu, je zavolána metoda *preview*. Tato metoda kontroluje, zda již není náhled spuštěn, nastavuje informační text ve stavové řádce, a nakonec také spouští samotný náhled společně s metodou *show_preview*,

která poté přijaté snímky vykresluje. Jak již bylo uvedeno dříve, obě tyto metody musí probíhat ve vlastním vlákně.

show_preview

tato metoda probíhá v cyklu po celou dobu živého náhledu. v každém cyklu je vyčten nejaktuálnější snímek ze snímkové fronty a je vykreslen do uživatelského rozhraní. Jelikož kamera může snímat velice rychle je nutné starší snímky zahodit, jinak by se náhled mohl stále více zpožďovat. Dále je také zbytečné vykreslovat snímky na vyšší frekvenci, než je obnovovací frekvence monitoru. Bylo tedy využito *win32api* [33], které umožňuje pomocí pár příkazů zjistit obnovovací frekvenci monitoru a omezit tak rychlost provádění vykreslovací smyčky.

Uvnitř samotného cyklu je poté zjištěn barevný formát a podle zvoleného typu přiblížení je obraz buďto procentuálně zmenšen, respektive zvětšen nebo je provedeno jeho roztažení na velikost náhledového okna.

Dále je proveden výpočet snímků za sekundu. Výpočet je pro lepší přesnost prováděn každých 30 cyklů, což je pro běžná zobrazovací zařízení 1-2x za sekundu. Původně byl výpočet prováděn přímo z obnovovací frekvence a z počtu celkově přijatých snímků (tedy vykreslených i zahozených), ukázalo se však, že v případě velkého zatížení programu, například v průběhu učení konvoluční sítě, nemá toto vlákno přístup k výpočetním prostředkům v přesně definovaných časech a výpočet se tak stal nepřesným a nestabilním. Jako řešení byl zvolen přístup s využitím modulu *time*, který obsahuje funkci *monotonic_ns*. Tato funkce vrací čas v nanosekundách nevázaný na žádný konkrétní výchozí bod. Díky této metodě je tak možné provádět výpočet snímků za sekundu dostatečně přesně i ve velkém zatížení. Modul *time* samozřejmě obsahuje množství dalších metod, které mohly být zvoleny, avšak *monotonic_ns* je pro danou aplikaci nejvhodnější, protože vrací neformátované číslo a použití funkce je díky tomu nenáročné na další výpočty.

Jakmile jsou všechny výše uvedené výpočty provedeny, přistoupí metoda k vykreslování snímku. Již v kapitole rozebírající knihovnu PyQt (kap.4.2.2) bylo zmíněno, že veškeré změny uživatelského rozhraní musí probíhat v hlavním vlákně aplikace, to platí i pro aktualizaci vykresleného snímku. Zvolené řešení uloží připravený snímek do proměnné *image_pixmap* a zavolá metodu *preview_callback*. Tato metoda má za úkol pouze vygenerovat signál na prvku, který není v uživatelském rozhraní vykreslen a jeho reakce je již napojena na samotné vykreslení snímku (metoda *update_image*). Tento přístup umožňuje přijímat a zpracovávat snímky ve vlastním vlákně a nenarušovat tak plynulý běh programu, ale zároveň zajišťuje, že vykreslení je provedeno v hlavním vlákně a nedochází tudíž k nahodilým pádům aplikace.

single_frame

Oproti kontinuálnímu záznamu, tato metoda mnohem méně zatěžuje systém, z toho důvodu není nutné spouštět příjem jednoho snímku ve vlastním vlákně, ale stačí jej přijmout v hlavním vlákně aplikace. Navíc se může stát, že v případě použití vlákna pro takto jednoduchou metodu by výsledek byl pomalejší díky režii spojené s tvorbou a ukončováním vlákna. Zbytek této metody pracuje obdobně jako vykreslování snímku v případě souvislého náhledu a bližší informace lze tedy nalézt v popisu metody *show_preview*.

load_parameters

Čtení parametrů v rámci aplikace probíhá dvěma způsoby. První způsob je použit při prvním otevření záložky camera configuration a také při každé změně konfigurační úrovně. Tyto metody zahrnují načtení všech parametrů. V první řadě metoda *show_parameters* odstraní všechny již přítomné parametry z uživatelského rozhraní a poté cyklicky přidává nové parametry, dokud nejsou přidány všechny. Na základě rozboru formátu pro předávání parametrů, který byl popsán ve třídě *Camera* (viz kapitola 5.1.4), metoda zařadí každý parametr do kategorie, nebo kategorii vytvoří, pokud zatím neexistuje. Dále je vytvořen štítek s názvem parametru a je mu přiřazena kontextová nápověda. Obdobným způsobem jsou pak vytvořeny prvky pro hodnotu parametru vždy dle jeho typu. Zároveň je na každý prvek obsahující hodnotu navázána metoda třídy *Camera* pro změnu této hodnoty. Štítky jsou uloženy ve slovníku *feat_labels* a prvky obsahující hodnoty ve slovníku *feat_widgets*. K hodnotám lze přistupovat pomocí názvu v atributu *name*. Jakmile je parametr tímto způsobem připraven, je navázán do příslušné kategorie strukturovaného seznamu. Tento přístup zajišťuje dynamické přidávání prvků a nezáleží tak na standardu nebo konkrétních názvech parametrů, jediná důležitá věc je, aby byl dodržen definovaný formát slovníku s parametry.

update_parameters

Z uživatelského pohledu je pohodlnější, když jsou veškeré parametry aktuální a není nutné starat se o jejich ruční aktualizaci. Proto byla implementována metoda *update_parameters* a další pomocné metody, které zajišťují, že každé 4 sekundy bude provedena aktualizace hodnot parametrů. Pro provádění této metody by bylo zbytečné znovu načítat veškeré informace o parametrech, jelikož jediné, co se v čase může měnit, jsou jejich hodnoty. Tato metoda tedy zažádá objekt třídy *Camera* postupně o aktuální hodnotu každého z parametrů. Hodnota v uživatelském rozhraní je přepsána, pokud se neshoduje s novou vyčtenou hodnotou. Opět musí být

brán zřetel na provádění aktualizace v hlavním vlákně aplikace, metody jsou tudíž napojeny na signál časovače, který je volá každé 4 sekundy.

Třída obsahuje značné množství dalších metod, ale jak již bylo zmíněno dříve, jde převážně o metody kontrolující data nebo mající přímou vazbu na metody jiné třídy. Mezi takové metody se mimo jiné řadí *preprocess_dataset*, *update_processing*, *train_model*, *update_training* a *predict*, které po kontrole svých vstupů pouze zavolají metody ve třídě *Computer_vision*, které již byly popsány dříve v kapitole 5.1.4. Ostatní, zde neuvedené metody, jsou popsány v dokumentaci nebo v kódu samotné aplikace.

6 Aplikace z uživatelského pohledu

Detailní informace týkající se vnitřního fungování aplikace byly rozebrány v předchozích kapitolách. Tato kapitola nahlíží na aplikaci z pohledu uživatele, popisuje jednotlivé prvky uživatelského rozhraní a nastiňuje možnosti, které jsou uživateli k dispozici. Informace obsažené v této kapitole jsou v anglickém jazyce také součástí nápovědy přímo v programu.

6.1 Spuštění aplikace

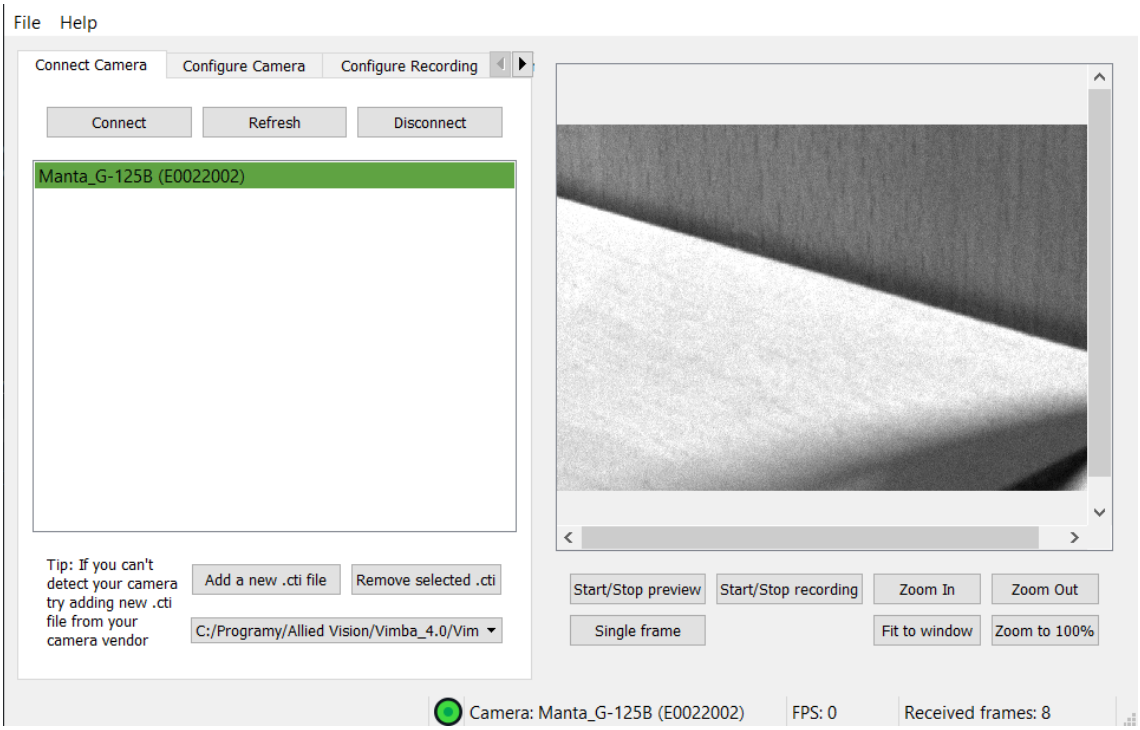
Jak již bylo popsáno v teoretickém rozboru, aplikace v jazyce Python potřebují ke svému běhu interpreter, který aplikaci překládá za běhu. Nevýhodou tohoto přístupu je nejen to, že počítač, na kterém aplikace běží musí mít interpreter nainstalován, ale také je nutno nainstalovat veškeré navázané knihovny.

Tento problém lze vyřešit vytvořením spustitelného souboru, který všechny tyto prvky obsahuje přímo v sobě. V případě této aplikace byl pro vytvoření spustitelného souboru zvolen PyInstaller [34]. Díky tomuto programu lze vytvořit spustitelný soubor a aplikaci tak přenášet mezi různými počítači. Pro korektní vygenerování souboru byl použit postup popsaný v úvodu dokumentace, která je součástí přiloženého CD.

Tento postup lze využít pro sestavení aktuální i budoucích verzí aplikace. Z kapacitních důvodů jsou součástí CD pouze zdrojové soubory a aplikaci je tak nutné spustit buďto jako Python skript nebo ji sestavit dle přiloženého návodu a spustit jako obyčejnou aplikaci v prostředí Windows.

6.2 Úvodní obrazovka

Na následujícím obrázku 6.1 lze vidět uživatelské rozhraní aplikace. Rozhraní bylo navrženo tak, aby působilo jednoduchým a přehledným dojmem. V pravé části je vždy dostupné okno s náhledem kamery a ovládacími prvky. Levá část potom obsahuje aktuálně vybranou záložku a možnost mezi záložkami přepínat. Bližší popis jednotlivých záložek bude uveden dále. Spodní část programu obsahuje jednoduchou stavovou lištu, která uživateli poskytuje některé důležité informace o stavu aplikace a kamery.



Obr. 6.1: Uživatelské rozhraní aplikace

6.3 Náhled kamery

V této oblasti se vykresluje aktuální snímek přijímaný z kamery. V rámci aplikace je tato funkce realizována pomocí knihovny OpenCV. Přijímané snímky se ve třídě Camera převádějí na jednotný formát obrazu OpenCV, který je poté vykreslován do uživatelského rozhraní.

6.4 Ovládání kamery

Tato část uživatelského rozhraní implementuje základní funkce pro ovládání kamery. Mezi tyto funkce se řadí: spustit náhled, zastavit náhled, spustit a ukončit nahrávání, sejmout jeden snímek, přiblížit nebo oddálit obraz.

Funkce spuštění a zastavení náhledu a sejmutí jednoho snímku jsou přímo navázány na odpovídající metody třídy *Camera*. Tlačítka pro spuštění a zastavení nahrávání se řídí parametry, které jsou uživatelem definovány v rámci záložky Recording Configuration.

Přiblížení a oddálení obrazu lze realizovat pomocí tlačítek *Zoom In* a *Zoom Out*, tyto funkce pouze modifikují obraz zobrazovaný v rámci náhledu kamery. Pokud chce uživatel zobrazit náhled v nezměněné formě, tedy aby 1 pixel snímku kamery odpo-

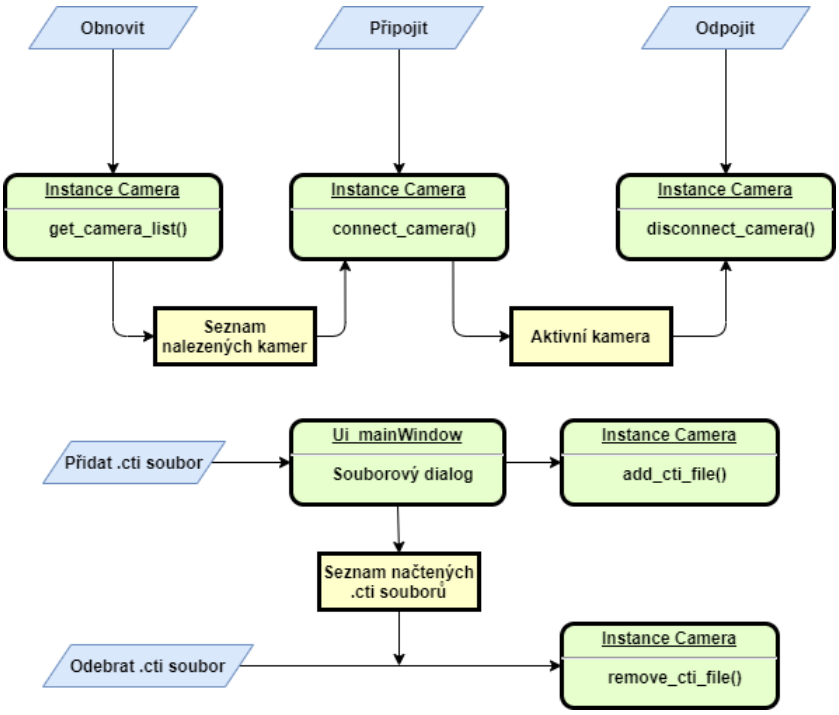
vídal 1 pixelu na obrazovce, potom použije tlačítko *Zoom 100 %*. Pro přizpůsobení snímku oknu náhledu slouží tlačítko *Zoom Fit*.

6.5 Záložky

V levém horním rohu okna aplikace se nachází výběr záložek. V současné verzi aplikace se zde nachází záložky: Connect Camera, Camera Configuration, Recording Configuration a Tensorflow. Bude-li v budoucnu aplikace rozšiřována, budou nové funkce přidány jako záložky právě v této části.

6.5.1 Connect Camera

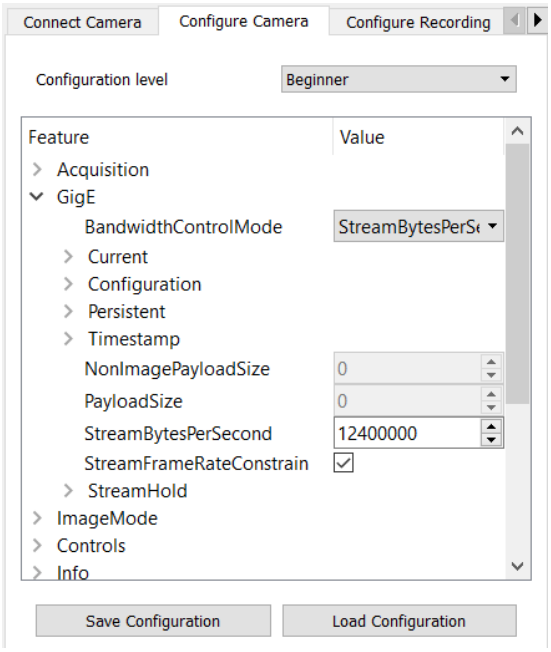
Tato záložka je zobrazena po spuštění aplikace. Zobrazení této záložky umožní okamžitě začít pracovat s kamerami. Zde má uživatel možnost vyhledat kamery a z výsledného seznamu vybrat tu, ke které se chce připojit. Záložku lze vidět na obrázku obr. 6.1 a její architektura je uvedena níže na obrázku 6.2.



Obr. 6.2: Vývojový diagram záložky Connect Camera

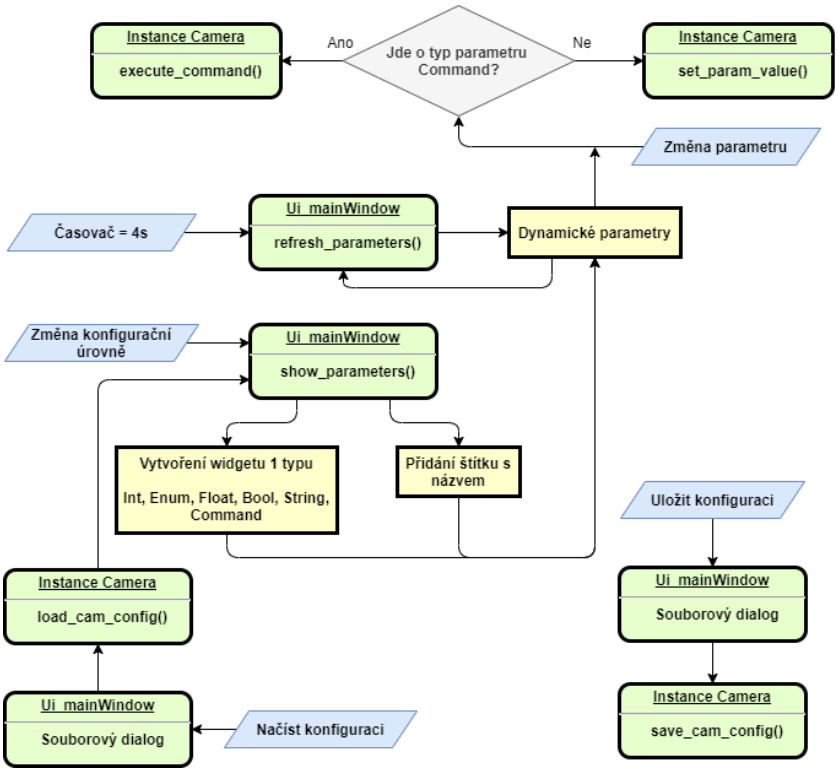
6.5.2 Configure Camera

Záložka Configure Camera plní svou funkci pouze při korektně připojené kameře. Pomocí metod, implementovaných ve třídě *Camera*, jsou jednotlivé parametry čteny přímo z kamery a jsou v definovaném tvaru posílány do uživatelského rozhraní. Zde se tyto parametry vypíší jako dvojice název parametru a hodnota. Podle typu hodnoty je možné vybrat si parametr ze seznamu, vložit vlastní numerickou hodnotu nebo zaškrtnout políčko v případě logického parametru.



Obr. 6.3: Záložka konfigurace kamery

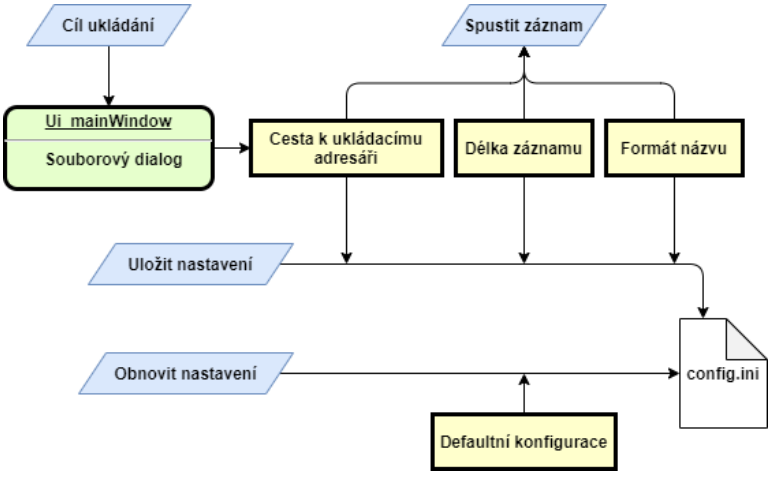
Kamery většiny výrobců si neuchovávají svou konfiguraci po odpojení napájení, případně mají jen omezenou paměť pro uložení několika málo uživatelských konfigurací. Z tohoto důvodu je velice důležitá implementace funkcí pro uložení konfigurace na disk. Samozřejmě je implementována také funkce pro nahrání této konfigurace zpět do kamery.



Obr. 6.4: Vývojový diagram záložky Configure Camera

6.5.3 Configure Recording

V této záložce probíhá veškerá konfigurace záznamu prováděná uživatelem. Náhled této záložky je na obrázku 6.6 a odpovídající vývojový diagram potom na obrázku 6.5.



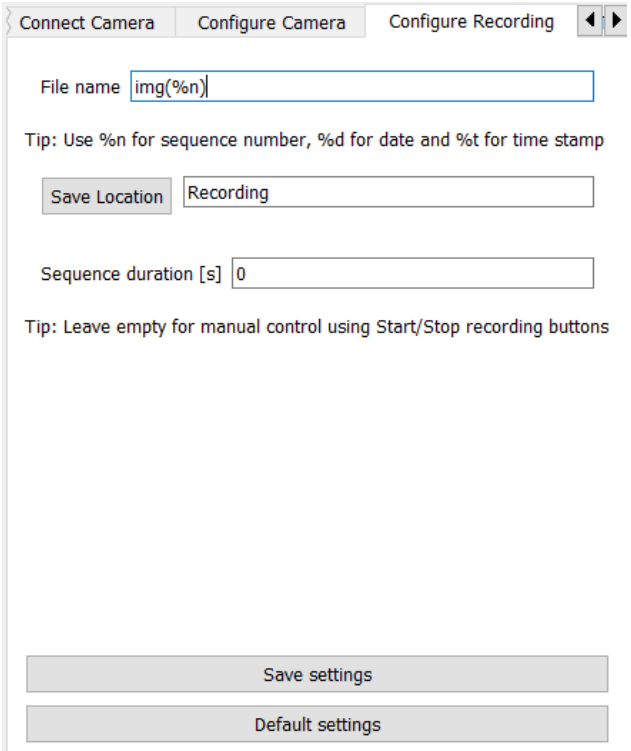
Obr. 6.5: Vývojový diagram záložky Configure Recording

Do pole *File name* uživatel zapíše název ukládaných souborů. Pod tímto polem se nachází nápověda pro správný formát číslování nebo přidání data a času do názvu jednotlivých snímků sekvence. Vstup pole je omezen pouze na znaky, které jsou v systémech Windows podporovány pro názvy souborů, nelze tedy vložit žádný ze skupiny znaků < > : "/ \| ? *.

Tlačítko *Save location* otevře systémové okno pro vybrání cíle uložení dat. Jakmile uživatel vybere umístění, je tato cesta vypsána do vedlejšího pole, kde má uživatel možnost ji dále modifikovat.

Do pole *Sequence* je možno zadat délku nahrávání. Pokud je pole ponecháno prázdné, bude nahrávání ovládáno přímo pomocí tlačítek *Start a Stop recording*.

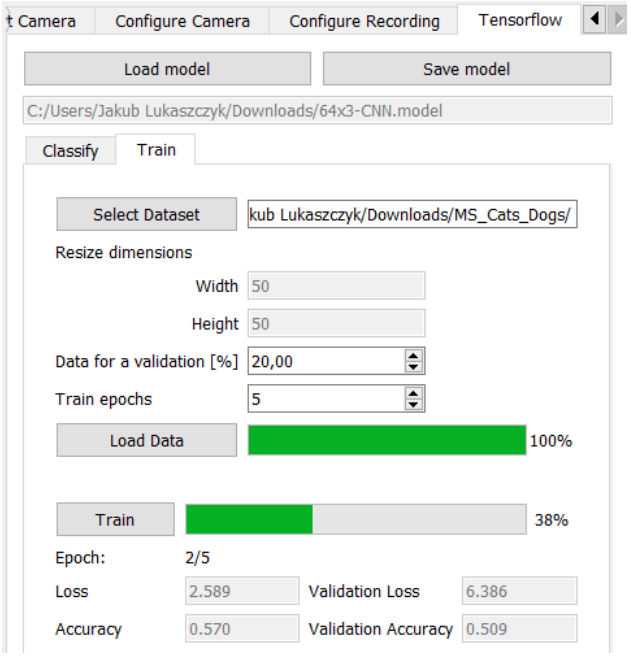
Posledními možnostmi v dolní části záložky jsou uložení konfigurace záznamu a obnovení výchozího nastavení. Tato konfigurace je ukládána do konfiguračního souboru aplikace a uživatel nemá možnost změnit její umístění.



Obr. 6.6: Záložka konfigurace záznamu

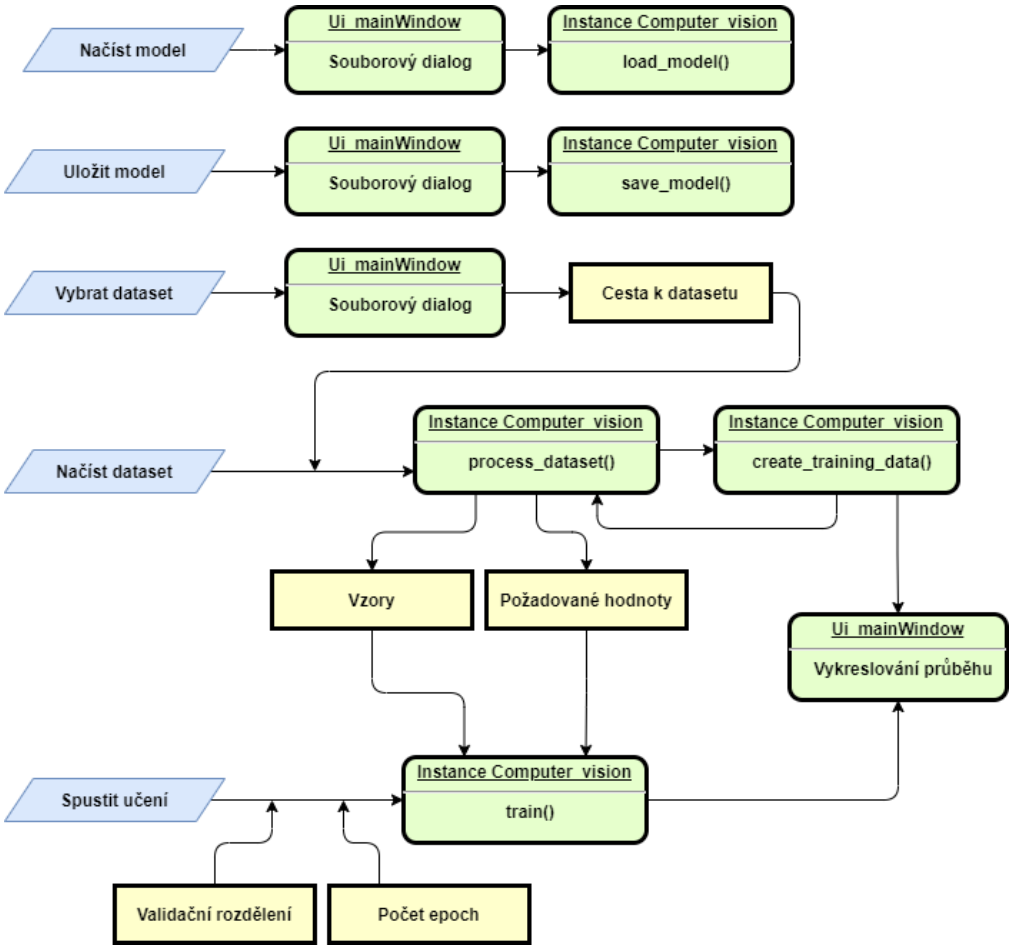
6.5.4 Tensorflow

Tato záložka umožňuje uživateli využít umělé neuronové sítě, jak pro klasifikaci, tak i pro učení. Před jakýmkoliv použitím je nutno načíst model vygenerovaný a zkompilovaný pomocí API Keras (tlačítko *Load Model*). Poté, pokud uživatel chce síť otestovat na živém náhledu, zvolí záložku classify, která mu ukáže aktivaci 5 nejpravděpodobnějších kategorií. Pokud má více kategorií nulovou aktivaci nebo je jejich celkový počet menší než 5, může se v grafu zobrazit méně výstupních kategorií.



Obr. 6.7: Záložka Tensorflow

Pro účely učení sítě je připravena záložka Train, kde je uživatel vyzván ke zvolení souboru učících dat (adresář, který pro každou kategorii obsahuje podadresář, ve kterém jsou vzory náležící této kategorii). Před zahájením procesu učení je nutné data načíst a také zvolit počet epoch pro učení společně s hodnotou pro rozdělení na učící a validační část. Ve spodní části záložky je potom v průběhu učení uživatel informován o výsledcích, které síť aktuálně podává. Bližší představu o této záložce jistě poskytne přiložený obrázek 6.7 a vývojový diagram na obrázku 6.8.



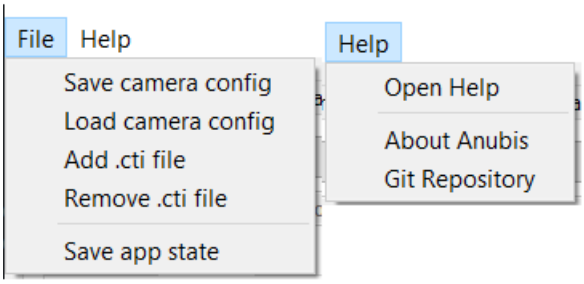
Obr. 6.8: Vývojový diagram záložky Tensorflow

6.6 Stavová lišta a položky menu

V levé části dolní lišty se zobrazuje aktuálně prováděná akce. Díky této funkci má uživatel přehled o tom, co aplikace aktuálně zpracovává. Na pravé straně se poté nachází informace o aktuálním počtu snímků za sekundu a celkovém počtu přijatých snímků.

Obr. 6.9: Stavová lišta

V levé horní části se nachází dvě položky menu tak, jak je u aplikací v OS Windows zvykem. Položka *File* umožňuje uložit stav aplikace nebo přidat, respektive odebrat soubory transportní vrstvy. Pod položkou *Help* lze kromě očekávané nápovědy nalézt také položku *o programu* a odkaz na Git repozitář, který obsahuje vždy nejaktuálnější verzi zdrojových souborů aplikace.



Obr. 6.10: Položky menu

Závěr

V bakalářské práci byl proveden průzkum programů pro ovládání kamer. Z tohoto průzkumu vyšlo najevo, že dnes standardně dostupné programy poskytují základní funkce pro ovládání kamery a nastavení parametrů. Některé programy jako například Vimba Viewer (kapitola 3.1) umožňují uživateli modifikovat veškeré dostupné parametry pro danou kameru. Jiné programy, mezi které se řadí také IC Capture (kapitola 3.2), umožňují modifikaci pouze omezené množiny těchto parametrů. Nevýhodou některých programů je jejich vázanost na výrobce. Uživatel je poté nucen přepínat mezi několika programy s často velmi rozdílným rozhraním.

Výstupem práce je program, který skrze jednoduché uživatelské rozhraní umožňuje přístup ke kamerám pod standardy USB3 Vision a GigE Vision. Aplikace je navržena tak, aby sama rozpoznala, o které rozhraní se jedná a dle toho zvolila vhodný způsob komunikace. Pro kamery, které mají nestandardní implementaci nebo pracují pod zcela jiným standardem byla aplikace připravena tak, aby ji bylo možné jednoduše rozšířit úpravou jasně definovaných metod a s minimálním zásahem do kódu uživatelského rozhraní. Díky těmto postupům byla zajištěna modularita a rozšiřitelnost na dostatečné úrovni.

Samotné uživatelské rozhraní je navrženo pomocí QtDesigner a v přiložené dokumentaci je popsáno, které metody měnit v případě přidání nových prvků uživatelského rozhraní. Tímto postupem lze rozšířit uživatelské rozhraní, tentokrát bez zásahu do logické části programu.

Aplikace umožňuje nalezení připojených kamer, živý náhled a ukládání snímků. Pro manipulaci s náhledem byla implementována funkce pro přiblížení obrazu nebo jeho přizpůsobení náhledovému oknu. Uživateli bylo umožněno nastavit délku záznamu a také formát názvu ukládaných snímků. Změna konfigurace kamery je možná skrze vlastní záložku a umožňuje konfiguraci ve třech úrovních tak, jak jsou definovány standardem GenICam. Pro každý parametr je identifikován jeho typ a na základě toho je zvolen vhodný prvek uživatelského rozhraní pro jeho reprezentaci. Tuto konfiguraci lze uložit do souboru a také ji zpětně nahrát do kamery.

Pro provádění strojového učení byla použita knihovna Tensorflow a její vysokoúrovňové API Keras. Do aplikace je možné načíst již hotový a zkompilovaný model neuronové sítě. S tímto modelem je možné provádět klasifikaci v reálném čase, kdy je uživateli ve formě grafu prezentován výstup pěti tříd s nejvyšší aktivací na daný snímek. Aplikaci lze také využít pro učení načteného modelu, a to jednoduše zvolením adresáře s vhodně formátovaným souborem obrazových dat a spuštěním učení. Takto doučený model lze samozřejmě uložit.

Součástí aplikace je také dokumentace a nápověda v anglickém jazyce. Dokumentace, společně se znalostí architektury, popsané v této práci, je důležitou součástí v

případě budoucího rozšiřovaní aplikace. Znalost architektury také minimalizuje riziko nevědomé modifikace části programu přepsáním zdánlivě nesouvisející metody.

Další vývoj

Aplikace byla otestována na kameře *Manta G-125B* a jeví se být plně funkční. Testování s více kamerami nemohlo být z technických důvodů provedeno. Jakmile to situace dovolí, bude provedeno testování s dalšími kamerami od různých výrobců a běh aplikace se současně připojenými více kamerami. Případné úpravy kódu budou součástí aktualizace projektového repozitáře [31].

Mezi další možná rozšíření výstupu této práce se řadí převážně potenciál pro implementaci nových standardů a podpora většího množství výrobců. Dále by v případě rozšiřování aplikace bylo vhodné provést refaktorování kódu a jednotlivé třídy rozdělit na menší logické celky, případně implementovat automatické testy převážně pro část backend. Provádění automatických testů pro uživatelské rozhraní by již bylo komplikované, jelikož tato část programu silně závisí na interakci uživatele s vykresleným uživatelským rozhraním.

Literatura

[1] DIERKS, Fritz. GenICam Standard: Generic Interface for Cameras. In: *EMVA: European Machine Vision Association* [online]. 2009-11-06 [ver. 2.0] [cit. 2020-12-23]. Dostupné z: https://www.emva.org/wp-content/uploads/GenICam_Standard_v2_0.pdf

[2] CAREY, Eric, Stephane MAURICE, Vincent ROWLEY, Thies MÖLLER a Karsten I. CHRISTENSEN. GenICam Standard Features Naming Convention. In: *EMVA: European Machine Vision Association* [online]. 2019-05-27 [ver. 2.5] [cit. 2020-12-23]. Dostupné z: https://www.emva.org/wp-content/uploads/GenICam_SFNC_v2_6.pdf

[3] GigE Vision. In: *Atesystem: Focused on Detail* [online]. Česká republika [cit. 2020-10-14]. Dostupné z: https://eshop.atesystem.cz/clanek/177/gige-vision

[4] Gigabit Ethernet and GigE Vision. In: *Basler: the power of sight* [online]. [cit. 2020-10-14]. Dostupné z: https://www.baslerweb.com/en/vision-campus/interfaces-and-standards/gigabit-ethernet/

[5] NAGGATZ, Marcel, Henrik RICHTER a Eric CAREY. GigE Vision Mechanical Supplement. In: *Aia Vision Online: Global Association for Vision Information* [online]. 2018-09-25 [cit. 2020-11-04]. Dostupné z: https://www.visiononline.org/vision/download-gige-vision-mechanical-supplement-free

[6] Everything you need to know about USB3 Vision. In: *OEM* [online]. Švédsko, 2013 [cit. 2020-10-17]. Dostupné z: http://media.oem.se/aut/oem_aut/oem_aut_se/pdf/imaging_vision_12-2013.pdf

[7] USB 3.0 Interface and USB3 Vision Standard. In: *Basler: the power of sight* [online]. [cit. 2020-10-16]. Dostupné z: https://www.baslerweb.com/en/vision-campus/interfaces-and-standards/usb3/

[8] MCCURRACH, Bob, Eric GROSS, Mark BREAUX a Thomas HOPFNER. USB3 Vision. In: *Aia Vision Online: Global Association for Vision Information* [online]. Michigan, 2019-09-01 [cit. 2021-03-14]. Dostupné z: https://www.visiononline.org/vision-standards-details.cfm?id=167&type=11

[9] FREEMAN, Jonathan. What is an API? Application programming interfaces explained. In: *InfoWorld* [online]. 2019-08-08 [cit. 2020-12-02]. Dostupné z: https://www.infoworld.com/article/3269878/what-is-an-api-application-programming-interfaces-explained.html

[10] *Clean Code: A Handbook of Agile Software Craftsmanship*. Upper Saddle River, NJ, Prentice Hall: Pearson, 2008. ISBN 978-01-323-5088-4.

[11] ROSSUM, Guido van, Barry WARSAW a Nick COGHLAN. PEP 8 – Style Guide for Python Code. In: *Python* [online]. 2001-07-05 [cit. 2020-11-10]. Dostupné z: https://www.python.org/dev/peps/pep-0008/

[12] SUMIT, Saha. A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way. In: *Towards data science* [online]. 15 December 2018 [cit. 2021-09-03]. Dostupné z: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[13] BROWNLEE, Jason. A Gentle Introduction to Pooling Layers for Convolutional Neural Networks. In: *Machine Learning Mastery: Making Developers Awesome at Machine Learning* [online]. 22 April 2019 [cit. 2021-03-09]. Dostupné z: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/

[14] JAI SDK and Control Tool. *JAI: See the possibilities* [Software]. 2020-04-01 ver. 3.0.7 [cit. 2020-10-14]. Dostupné z: https://www.jai.com/support-software/jai-software

[15] IC Capture: image acquisition. *The Imaging Source: Technology Based on Standards* [Software]. 2020-05-11 ver. 2.5.1525.3931 [cit. 2020-12-22]. Dostupné z: https://www.theimagingsource.com/support/downloads-for-windows/end-user-software/iccapture/

[16] Pylon Viewer: součást pylon 6.1.1 Camera Software Suite Windows. *Basler: the power of sight* [Software]. 2020-05-12 ver. 6.2.0.8205 [cit. 2020-10-14]. Dostupné z: https://www.baslerweb.com/en/sales-support/downloads/software-downloads/

[17] VIMBA: The SDK for Allied Vision Cameras. *Allied Vision* [Software]. ver. 4.2 [cit. 2020-12-22]. Dostupné z: https://www.alliedvision.com/en/products/software.html

[18] KRUG, Steve. *Don't make me think, revisited: a common sense approach to web usability*. [Berkeley]: New Riders, [2014]. Voices that matter. ISBN 978-0-321-96551-6.

[19] STICKA, Tyler. Designing Button States. *Cloud Four* [online]. Portland, Oregon, c2007-2021, 13. 3. 2018 [cit. 2021-4-29]. Dostupné z: https://cloudfour.com/thinks/designing-button-states/

[20] *Kivy: Cross-platform Python Framework for NUI Development* [online]. [cit. 2021-5-1]. Dostupné z: https://kivy.org/

[21] Kivy Catalog. *Kivy: Cross-platform Python Framework for NUI Development* [online]. [cit. 2021-5-1]. Dostupné z: kivy.org/doc/stable/examples/gen___demo___kivycatalog___main___py.html

[22] Qt Designer. *Qt* [online]. ver. 5.9.7 [cit. 2020-12-19]. Dostupné z: https://www.qt.io/download-open-source?hsCtaTracking=9f6a2170-a938-42df-a8e2-a9f0b1d6cdce%7C6cb0de4f-9bb5-4778-ab02-bfb62735f3e5

[23] What is PyQt? *Riverbank Computing* [online]. [cit. 2020-12-19]. Dostupné z: https://www.riverbankcomputing.com/software/pyqt/

[24] THE GENICAM COMMITTEE. GenICam. *The Python Package Index* [online]. June 10 2020 [cit. 2021-04-03]. Dostupné z: https://pypi.org/project/genicam/

[25] Harvesters: 1.2.8. *Python package index* [online]. 2020-07-22 [cit. 2020-12-23]. Dostupné z: https://pypi.org/project/harvesters/

[26] *Harvester's Documentation* [online]. [cit. 2020-12-22]. Dostupné z: https://harvesters.readthedocs.io/en/latest/

[27] Image Acquisition Interface for GenICam GenTL Compliant Cameras. *MVTec* [online]. 2017-11-28 [cit. 2020-12-23]. Dostupné z: https://www.mvtec.com/products/interfaces/documentation/view/1303-standard-13-mvtecdoc-genicamtl

[28] , TIS-Stefan. Python implementation for TIS camera on Windows. *GitHub* [online]. c2021, 2001-05-02 [cit. 2021-5-9]. Dostupné z: https://github.com/TheImagingSource/tiscamera/issues/247

[29] ANDERSON, Jim. An Intro to Threading in Python. In: *Real Python* [online]. 2019-03-25 [cit. 2020-12-21]. Dostupné z: https://realpython.com/intro-to-python-threading/

[30] *OpenCV* [online]. Palo Alto (California), c2021 [cit. 2021-04-14]. Dostupné z: https://opencv.org/

[31] LUKASZCZYK, Jakub. GIT Repozitář. *GitHub* [online]. c2021 [cit. 2021-5-1]. Dostupné z: https://github.com/xlukas10/anubis

[32] COHEN, Dor. MTU and MSS: What You Need to Know. In: *Imperva* [online]. San Mateo (California), c2021, January 3 2017 [cit. 2021-04-14]. Dostupné z: https://www.imperva.com/blog/mtu-mss-explained/

[33] MICROSOFT. Win32api. In: *Microsoft* [online]. Redmont (Washington), c2021 [cit. 2021-04-14]. Dostupné z: https://docs.microsoft.com/en-us/windows/win32/api/

[34] PyInstaller. *PyInstaller* [online]. c2005-2019 [cit. 2021-4-28]. Dostupné z: https://www.pyinstaller.org/

Seznam příloh