Využití frameworku .NET pro zpracování obrazu a integraci hardwaru v desktopových aplikacích

Bc. Ondřej Černošek

Diplomová práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Ondřej Černošek**
Osobní číslo:	**A22268**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Kombinovaná**
Téma práce:	**Využití frameworku .NET pro zpracování obrazu a integraci hardwaru v desktopových aplikacích**
Téma práce anglicky:	**Utilization of the .NET Framework for Image Processing and Hardware Integration in Desktop Applications**

Zásady pro vypracování

1. Zpracujte přehled současného stavu dané problematiky.
2. Popište technologii .NET a vybrané algoritmy pro zpracování obrazu.
3. Vytvořte desktopovou aplikaci demonstrující zpracování obrazu a integraci hardwaru.
4. Na základě vytvořené aplikace zhodnoťte vhodnost frameworku .NET pro daný problém.
5. Zhodnoťte dosažené výsledky a možnosti dalšího vývoje do budoucna.

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

Seznam doporučené literatury:

1. GONZALEZ, Rafael C. a WOODS, Richard E. *Digital image processing*. Fourth edition. New York: Pearson, 2018. ISBN 978-1-292-22304-9.
2. FOWLER, Martin. *Patterns of enterprise application architecture*. The Addison-Wesley signature series. Boston: Addison-Wesley, 2003. ISBN 0321127420.
3. SKIENA, Steven. *The Algorithm Design Manual*. Second edition. Springer, 2008. ISBN 978-1849967204.
4. DOBEŠ, Michal. *Zpracování obrazu a algoritmy v C#*. Praha: BEN – technická literatura, 2008. ISBN 9788073002336.
5. GAMMA, Erich; HELM, Richard; JOHNSON, Ralph a VLISSIDES, John. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley professional computing series. Boston: Addison-Wesley, 1995. ISBN 0201633612.

Vedoucí diplomové práce: **Ing. Pavel Pokorný, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce: **5. listopadu 2023**
Termín odevzdání diplomové práce: **13. května 2024**

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

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

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

Prohlašuji,

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

Ve Zlíně, dne Ondřej Černošek v.r.

ABSTRAKT

Tato diplomová práce se zaměřuje na zkoumání a demonstraci možností využití frameworku .NET pro zpracování obrazu a integraci hardwaru v desktopových aplikacích. Hlavním cílem je vytvoření aplikace, která umožní komunikaci s kamerami, výkonným laserem a power meterem, a prostřednictvím této aplikace ukázat různé možnosti a metody implementace komunikace s hardwarovými zařízeními a práci se získanými daty.

Práce se zaměřuje na důkladnou analýzu a využití nástrojů pro zpracování obrazu ve frameworku .NET, s důrazem na metody zpracování obrazu pro detekci a analýzu dat z kamer, včetně rozpoznávání objektů. Dalším klíčovým aspektem je integrace hardwaru do aplikace, která zahrnuje analýzu a implementaci komunikačních protokolů pro komunikaci s kamerami, výkonným laserem a power meterem.

Klíčová slova: .NET, Windows Presentation Foundation, hardware, zpracování obrazu

ABSTRACT

This thesis focuses on exploring and demonstrating the possibilities of using the .NET framework for image processing and hardware integration in desktop applications. The main goal is to create an application that allows communication with cameras, a powerful laser and a power meter, and through this application to show different possibilities and methods of implementing communication with hardware devices and working with the acquired data.

The thesis focuses on a thorough analysis and use of image processing tools in the .NET framework, with an emphasis on image processing methods for the detection and analysis of camera data, including object recognition. Another key aspect is the integration of hardware into the application, which includes the analysis and implementation of communication protocols for communicating with cameras, a powerful laser and a power meter.

Keywords: .NET, Windows Presentation Foundation, hardware, image processing

Rád bych vyjádřil své upřímné poděkování Ing. Pavlovi Pokornému, Ph.D. za jeho podporu, odborné rady a trpělivost během tvorby této diplomové práce.

Také bych chtěl poděkovat svým blízkým za jejich trvalou podporu, pochopení a povzbuzení během mého studia. Bez jejich podpory by tato cesta nebyla možná.

Děkuji všem, kteří mi pomohli při tvorbě této práce, a všem, kteří mi věřili a podporovali mě na mé cestě.

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

OBSAH

ÚVOD

Tato diplomová práce bude zkoumat a demonstrovat možnosti využití frameworku .NET pro zpracování obrazu a integraci hardwaru v desktopových aplikacích. Hlavním cílem práce bude vytvoření aplikace, která bude komunikovat s kamerami, výkonným laserem a power meterem, a prostřednictvím této aplikace ukázat různé možnosti a metody implementace komunikace s hardwarovými zařízeními a práci se získanými daty.

Důkladná analýza a využití nástrojů pro zpracování obrazu ve frameworku .NET bude klíčovým prvkem práce. Budou zkoumány metody zpracování obrazu pro detekci a analýzu dat z kamer. Tyto funkce budou implementovány a optimalizovány v aplikaci.

Dalším důležitým aspektem práce bude integrace hardwaru do aplikace. Bude provedena analýza a implementace komunikačních protokolů pro komunikaci s kamerami, výkonným laserem a power meterem. To zahrnuje správu připojení, řízení zařízení a získávání dat z těchto zařízení. Zároveň bude kladen důraz na stabilitu, spolehlivost a efektivitu komunikace.

Důležitou částí diplomové práce bude práce se získanými daty. To zahrnuje jejich analýzu, vizualizaci a interpretaci. Bude provedena implementace algoritmů pro zpracování a výpočet dat získaných z kamery a power meteru. Tyto data budou prezentována uživateli pomocí uživatelského rozhraní aplikace.

Cílem této diplomové práce je ukázat široké možnosti využití frameworku .NET pro zpracování obrazu a integraci hardwaru v desktopových aplikacích. Vytvořená aplikace bude sloužit jako demonstrace těchto možností a bude poskytovat užitečné poznatky pro další výzkum a vývoj v této oblasti.

I. TEORETICKÁ ČÁST

1 OBRAZ

Obraz lze definovat jako dvoudimenzionální funkci f(x, y), kde x a y jsou prostorové (rovinové) souřadnice a amplituda f na libovolném páru souřadnic (x, y) se nazývá intenzita nebo úroveň šedi obrazu v tomto bodě. Digitální obraz je složen z konečného počtu prvků nazývaných pixel, které mají konkrétní umístění a hodnotu. [1][2]

Zrak je nejvyspělejším z našich smyslů, a proto není překvapující, že obrázky hrají klíčovou roli v lidském vnímání. Na rozdíl od lidí jsou obrazové stroje schopny pracovat s celým elektromagnetickým spektrem. Digitální zpracování obrazu zahrnuje široké spektrum aplikací, včetně ultrazvuku, elektronové mikroskopie a počítačem vytvořených obrazů. [1][2]

Není obecná shoda mezi autory ohledně toho, kde končí zpracování obrazu a začínají jiné oblasti, jako je analýza obrazu a počítačové vidění. Někdy je rozdíl definován tak, že zpracování obrazu zahrnuje disciplínu, ve které jsou jak vstupní, tak výstupní prvky procesu obrazy. Věří se však, že tato definice je omezující a poněkud umělou hranicí. [1][2]

Podle předchozích komentářů je logickým místem překryvu mezi zpracováním obrazu a analýzou obrazu oblast rozpoznávání jednotlivých oblastí nebo objektů na obraze. To, co nazýváme digitálním zpracováním obrazu v této práci, zahrnuje vedle procesů, jejichž vstupy a výstupy jsou obrazy, a zahrnuje také ty, které extrahují atributy z obrazů až po rozpoznání jednotlivých objektů. [1][2]

1.1 Zpracování obrazu

Zpracování obrazu je fascinující disciplínou, která se zabývá analýzou a manipulací digitálních obrazů, s cílem získat užitečné informace, nebo dosáhnout určitého výsledku. Tato oblast má široké uplatnění v různých odvětvích včetně lékařství, průmyslu, bezpečnosti, robotiky a mnoha dalších. Zde se podíváme na klíčové koncepty, metody a techniky zpracování obrazu.

1.1.1 Úvod do zpracování obrazu

Zpracování obrazu začíná sběrem dat pomocí digitálních zařízení, jako jsou fotoaparáty nebo kamery. Tyto zařízení zachycují obraz scény a převádějí ho na digitální formát, který se skládá z mřížky bodů nazývaných pixely. Každý pixel má svou hodnotu, která reprezentuje intenzitu světla nebo barvu daného místa na obrazovce.

Sběr dat je prvním krokem v procesu zpracování obrazu a zajišťuje převod fyzických objektů a scén do digitální podoby. Poté následuje samotné zpracování obrazu. Zde jsou data získaná během sběru analyzována a upravována pomocí různých algoritmů a technik. Cílem zpracování obrazu může být různorodý, od zlepšení kvality obrazu až po rozpoznávání objektů, změnu barev nebo aplikaci různých efektů.

1.1.2 Základní operace zpracování obrazu

Základní operace zpracování obrazu zahrnují manipulace s obrazy na základní úrovni, jako je změna velikosti, ořezání, rotace a změna jasu a kontrastu. Tyto operace jsou často prováděny pomocí lineární algebry a maticových operací, které umožňují efektivně transformovat pixely obrazu. Například změna velikosti obrazu může být prováděna pomocí transformace maticí, která řídí interpolaci mezi existujícími pixely, aby vytvořila nový rozměr obrazu. Ořezání obrazu je také často prováděno pomocí matematických operací, které vybírají určitou oblast obrazu a odstraní zbytek. Rotace obrazu je opět prováděna pomocí matice rotace, která aplikuje geometrickou transformaci na každý pixel obrazu. Změna jasu a kontrastu může být dosažena pomocí úpravy hodnot pixelů v obraze, což může být vyjádřeno jako maticová operace. [2]

1.1.3 Pokročilé techniky zpracování obrazu

Mezi pokročilé techniky zpracování obrazu patří detekce hran, segmentace, filtrace a extrakce příznaků. Detekce hran identifikuje prudké změny v intenzitě obrazu, což může naznačovat hrany objektů v obraze. Segmentace rozděluje obraz do jednotlivých částí nebo objektů na základě určitých kritérií, jako je barva nebo textura. Filtrace umožňuje odstranění šumu z obrazu nebo zvýraznění určitých oblastí zájmu. Extrakce příznaků se zabývá identifikací a popisem důležitých částí obrazu, které mohou sloužit jako vstup pro další analýzu nebo rozhodování. [2]

1.1.4 Aplikace zpracování obrazu

Zpracování obrazu má široké spektrum aplikací v různých odvětvích. V lékařství se využívá k diagnostice a monitorování chorob pomocí medicínských snímků. V průmyslu je používáno k automatizaci procesů, jako je kontrola kvality výrobků nebo sledování výrobních procesů. V bezpečnostních systémech se využívá k rozpoznávání obličejů, sledování pohybu a detekci neobvyklých událostí. V robotice je zpracování obrazu klíčové pro navigaci robotů a interakci s okolím.

Zpracování obrazu je tedy důležitou disciplínou s širokým spektrem aplikací a neustále se rozvíjejícími technikami a metodami. S rostoucím výkonem počítačů a pokrokem v oblasti algoritmů je možné očekávat stále větší pokrok v této oblasti a nové možnosti využití v praxi.

1.2 Využití zpracování obrazu

Dnes téměř není technické oblasti, kterou by digitální zpracování obrazu nějakým způsobem neovlivnilo. V této části je popsáno mnoho oblastí aplikací, z nichž každá využívá technik digitálního zpracování obrazu.

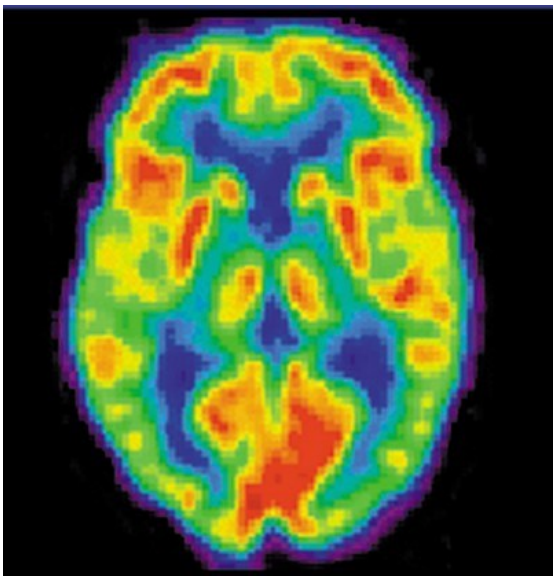
Jedním z nejjednodušších způsobů, jak vyvinout základní pochopení rozsahu aplikací zpracování obrazu, je kategorizovat obrázky podle jejich zdroje (např. rentgenové snímky, vizuální, infračervené atd.). Hlavním zdrojem energie pro dnešní obrázky je elektromagnetické spektrum energie. Další důležité zdroje energie zahrnují akustiku, ultrazvuk a elektronickou energii (ve formě elektronů používaných v elektronové mikroskopii). Syntetické obrazy, používané k modelování a vizualizaci, jsou generovány počítačem. V této části je stručně popsáno, jak jsou obrazy v těchto různých kategoriích generovány a v jakých oblastech jsou používány. [1][2]

Z obrázků založené na záření elektromagnetického spektra jsou nejznámější, zejména obrázky v rentgenovém a vizuálním pásmu spektra. Elektromagnetické vlny lze představit jako šířící se sinusoidální vlny s různými vlnovými délkami, nebo je lze chápat jako proud hmotných částic, každá se pohybující ve vlnovém vzoru a pohybující se rychlostí světla. Každá hmotná částice obsahuje určité množství (nebo svazek) energie. Každý svazek energie se nazývá foton. Jestliže jsou spektrální pásma seskupena podle energie na foton, získáme spektrum, které sahá od gama paprsků (nejvyšší energie) na jednom konci až po rádiové vlny (nejnižší energie) na druhém. Pásma jsou zobrazena stínovaná, aby bylo patrné, že pásma elektromagnetického spektra nejsou výrazně oddělená, ale plynule přecházejí od jednoho k druhému. [1][2]

1.2.1 GAMA-PAPRSKOVÉ ZOBRAZENÍ

Hlavními použitími obrazů založených na gama paprscích jsou nukleární medicína a astronomická pozorování. V nukleární medicíně se vstříkne do pacienta radioaktivní izotop, který vydává gama paprsky při rozpadu. Obrázky jsou vytvořeny z emisí zachycených detektory gama paprsků. Princip je stejný jako u rentgenové tomografie. Místo použití externího zdroje rentgenové energie je pacientovi podán radioaktivní izotop, který vydává

pozitrony při svém rozpadu. Když se pozitron setká s elektronem, oba jsou anihilováni a vydávají se dva gama paprsky. Ty jsou zachyceny a vytvoří se tomografický obraz pomocí základních principů tomografie. [1][2][3]

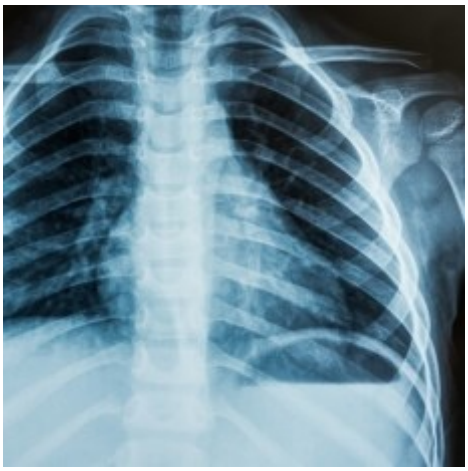


Obrázek 1 – Snímek Gama zobrazení [3]

1.2.2 RENTGENOVÉ ZOBRAZENÍ

Rentgenové paprsky patří mezi nejstarší zdroje elektromagnetického záření používaného pro zobrazování. Nejznámějším použitím rentgenových paprsků je lékařská diagnostika, ale jsou také rozsáhle používány v průmyslu a dalších oblastech, jako je astronomie. Rentgenové paprsky pro lékařské a průmyslové zobrazování jsou generovány pomocí rentgenové trubice, což je vakuová trubice s katodou a anodou. Katoda je zahřáta, což způsobuje uvolnění volných elektronů. Tyto elektrony proudí rychlostí k pozitivně nabité anodě. Když elektrony narazí na jádro, uvolňuje se energie ve formě rentgenového záření. Energie (průniková schopnost) rentgenových paprsků je ovládána napětím aplikovaným na anodu a proudem aplikovaným na vlákno katody. Intenzita rentgenových paprsků se mění absorpcí při průchodu pacientem a výsledná energie padající na film ji vyvolává, podobně jako u světla na fotografickém filmu. V digitální radiografii jsou digitální obrazy získávány jedním ze dvou způsobů: (1) digitalizací rentgenových filmů; nebo (2) necháním rentgenových paprsků, které procházejí pacientem, padnout přímo na zařízení (jako je fosforová deska), které přeměňuje rentgenové paprsky na světlo. Signál světla je pak zachycen světlocitlivým digitalizačním systémem. [1][2][3]

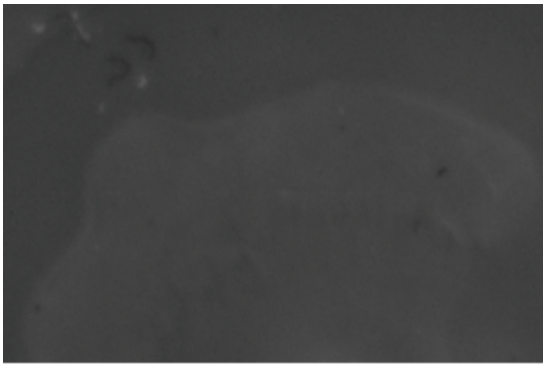
Dalším důležitým použitím rentgenových paprsků v lékařském zobrazení je počítačová axiální tomografie (CT). Kvůli svému rozlišení a 3D schopnostem CT skeny revolučně změnily medicínu od okamžiku, kdy se poprvé objevily na začátku 70. let. Každý CT obraz je "plátek" vzatý kolmo skrz pacienta. Několik plátků se generuje, když je pacient pohybován ve směru podélném. Soubor těchto obrázků tvoří 3D reprezentaci vnitřku těla, kde podélné rozlišení je úměrné počtu snímků plátků Průmyslové CT skeny jsou užitečné, když lze díly proniknout rentgenovými paprsky, například v plastových sestavách, a dokonce i velkých tělesech, jako jsou motory na tuhé pohonné látky. [1][2]



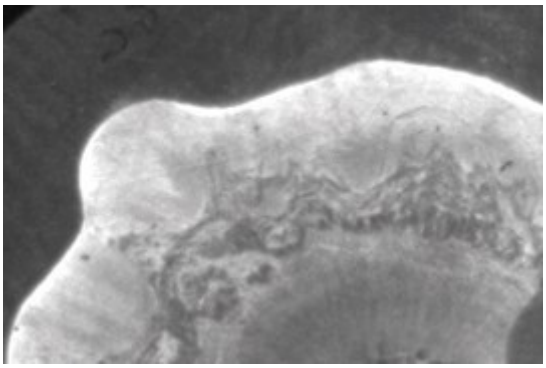
Obrázek 2 – Rentgenový snímek [4]

1.2.3 ZOBRAZENÍ V UV PÁSMU

Aplikace ultrafialového "světla" jsou různorodé. Zahrnují litografii, průmyslovou inspekci, mikroskopii, lasery, biologické zobrazování a astronomická pozorování. Ultrafialové světlo se používá ve fluorescenční mikroskopii, jedné z nejrychleji rostoucích oblastí mikroskopie. Samotné ultrafialové světlo není viditelné, ale když foton ultrafialového záření narazí na elektron v atomu fluoreskujícího materiálu, zdvihne elektron na vyšší energetickou úroveň. Následně vzrušený elektron relaxuje na nižší úroveň a emituje světlo ve formě fotonu nižší energie v oblasti viditelného světla. Důležité úkoly provedené fluorescenčním mikroskopem jsou použít excitační světlo k ozáření připraveného vzorku a následně oddělit mnohem slabší zářící fluorescenční světlo od jasnějšího excitačního světla. Takto se pouze emisní světlo dostane k oku nebo jinému detektoru. Výsledné fluoreskující oblasti září proti tmavému pozadí s dostatečným kontrastem pro detekci. Čím tmavší je pozadí nezářícího materiálu, tím efektivnější je nástroj. Fluorescenční mikroskopie je vynikající metoda pro studium materiálů, které lze přivést k fluorescenci. [1][2]



Obrázek 3 – Snímek bez osvětlení UV [5]



Obrázek 4 – Snímek s osvětlením UV [5]

1.2.4 ZOBRAZENÍ V VIDITELNÉM A INFRAČERVENÉM PÁSMU

S ohledem na to, že viditelné pásmo elektromagnetického spektra je nejznámější ve všech našich aktivitách, není překvapující, že zobrazení v tomto pásmu převyšuje všechny ostatní, co se využití v aplikacích týče. Infračervené pásmo je často používáno v kombinaci s vizuálním zobrazením. Využívá se v aplikacích ve světelné mikroskopii, astronomii, dálkovém průzkumu, průmyslu a v právních složkách. [1][2]

1.2.5 ZOBRAZENÍ V RADIOFREKVENČNÍM PÁSMU

Jednou z největších výhod radiofrekvenčních paprsků oproti viditelnému světlu je, že mohou proniknout skrze plynné nebo dokonce i pevné objekty. To jim umožňuje zobrazovat objekty, které nejsou viditelné v optickém spektru nebo které by byly zastíněny. [1][2]

1.2.6 ZOBRAZENÍ V MIKROVLNNÉM PÁSMU

Mikrovlnné zobrazování, známé také jako radary, bylo rozšířené v průmyslu, na vojenských zařízeních a ve vědeckém výzkumu. V poslední době bylo mikrovlnné zobrazování

věnováno velké množství pozornosti pro jeho schopnost proniknout nejen do atmosféry, ale také do povrchu Země. Radar lze použít nejen k identifikaci objektů, ale také k určení jejich vzdálenosti, rychlosti a směru pohybu. Různé barvy na obrázku označují různé vlastnosti povrchu, včetně vegetace, vody a zástavby. Mikrovlnné zobrazování se stává stále více užitečným nástrojem pro průzkum Země a sledování změn v krajině. Mikrovlnné paprsky mohou proniknout do oblaků a jsou schopny "vidět" skrze ně, což je výhoda při průzkumu a monitorování povrchu. Celkově tyto dva příklady ilustrují, jak se mikrovlnné zobrazování využívá ve velké škále aplikací od vojenského průzkumu po sledování klimatických podmínek a zemědělských oblastí. [1][2]

1.2.7 ZOBRAZENÍ V MÉČOVÉM PÁSMU

Měřená rychlost oběžných drah a hmotnost planet vedly k mnoha objevům v oblasti astronomie. Jedním z klíčových nástrojů pro tuto práci bylo používání měření elektromagnetického záření ze světla s nejdelší vlnovou délkou, což je záření v mikrovlnném pásu elektromagnetického spektra. Antény citlivé na mikrovlnné záření umožnily astronomům studovat radiový signál vysílaný planetami a hvězdami. Radioměření Slunce v mikrovlnném pásu poskytují informace o sluneční aktivitě, které mohou být využity k predikci slunečních erupcí a následných vlivů na Zemi, včetně geomagnetických bouří. Radioměření vysílané Saturnem poskytuje informace o atmosféře a magnetosféře planety. Mikrovlnná radiometrie umožnila detailní studium mnoha objektů ve vesmíru a přinesla důležité poznatky do oblasti astronomie. [1][2]

1.3 ELEMENTY VIZUÁLNÍ PERCEPCE

I když oblast digitální zpracování obrazu stojí na základech matematiky, lidská intuice a analýza často hrají roli při volbě jedné techniky před druhou a tato volba často probíhá na základě subjektivních, vizuálních úsudků.

Proto je vhodné rozvíjet porozumění základním charakteristikám lidské vizuální percepce jako první krok v naší cestě skrze tuto knihu. Zejména nás zajímají elementární mechanismy toho, jak jsou obrazy utvářeny a vnímány lidmi. Zajímá nás pochopení fyzikálních omezení lidského vidění ve smyslu faktorů, které jsou také používány při naší práci s digitálními obrazy. Faktory jako je to, jak se lidské a elektronické zobrazovací zařízení porovnávají ve smyslu rozlišení a schopnosti přizpůsobit se změnám v osvětlení, nejsou pouze zajímavé, ale jsou také důležité z praktického hlediska. [1][2]

1.3.1 TVOŘENÍ OBRAZU V OKU

V běžné fotografické kameře má objektiv pevnou ohniskovou vzdálenost. Zaostření na různé vzdálenosti se dosahuje změnou vzdálenosti mezi objektivem a obrazovou rovinou, kde se nachází film (nebo obrazový čip v případě digitální kamery). [1][2]

V lidském oku je tomu naopak; vzdálenost mezi středem čočky a obrazovým snímačem (sítnicí) je pevná a ohnisková vzdálenost potřebná k dosažení správného zaostření se získává změnou tvaru čočky. [1][2]

1.3.2 JASOVÁ ADAPTACE A DISKRIMINACE

Jelikož digitální obrázky jsou zobrazeny jako sady diskrétních intenzit, schopnost oka rozlišovat mezi různými úrovněmi intenzity je důležitým hlediskem při prezentaci výsledků zpracování obrazu. Rozsah úrovní světelné intenzity, ke kterým se lidský vizuální systém může přizpůsobit, je obrovský. Celkový rozsah rozlišitelných úrovní intenzity, které oko může současně diskriminovat, je poměrně malý ve srovnání s celkovým rozsahem adaptace. Pro danou sadu podmínek je aktuální úroveň citlivosti vizuálního systému nazývána jasovou adaptační úrovní, která může odpovídat například jasu. [1][2]

Schopnost oka rozlišovat změny v intenzitě světla při jakékoli konkrétní úrovni adaptace je velmi zajímavá. Klasický experiment používaný k určení schopnosti lidského vizuálního systému pro rozlišování jasu spočívá v tom, že subjekt pozoruje plochou, jednotně osvětlenou oblast dostatečně velkou na to, aby zaujala celé zorné pole. Tato oblast je obvykle difuzní, například z neprůhledného skla, osvětlená zezadu zdrojem světla I s proměnlivou intenzitou. K této oblasti je přidáno zvýšení osvětlení, I, ve formě krátkodobého záblesku, který se jeví jako kruh ve středu jednotně osvětlené oblasti. Pokud není I dostatečně jasný, subjekt řekne "ne", což znamená, že žádná vnímatelná změna není. Když se I zesiluje, subjekt může dát pozitivní odpověď "ano", což znamená vnímatelnou změnu. Nakonec, když je I dostatečně silný, subjekt bude reagovat "ano" po celou dobu. Množství I/Ic, kde Ic je zvýšení osvětlení, které je 50 % času rozlišitelné s pozadím osvětlením I, se nazývá Weberův poměr. Malá hodnota I/Ic znamená, že je rozlišitelná malá procentuální změna intenzity. To představuje "dobrou" diskriminaci jasu. Naopak, velká hodnota I/Ic znamená, že je třeba velká procentuální změna intenzity pro to, aby byla změna vnímána okem. To představuje "špatnou" diskriminaci jasu. [1][2]

Pokud je pozadí osvětlení konstantní a intenzita druhého zdroje, místo záblesku, je nyní povolena postupně se měnit od nikdy nevnímané po vždy vnímanou, typický pozorovatel

může rozeznat celkem jedno až dvě desítky různých změn intenzity. Tento výsledek je hrubě spojen s počtem různých intenzit, které člověk může vidět v libovolném bodě nebo malé oblasti v monochromatickém obraze. To neznamená, že by obraz mohl být reprezentován tak malým počtem hodnot intenzity, protože když se oko pohybuje po obraze, průměrné pozadí se mění, což umožňuje detekci jiné sady postupných změn při každé nové úrovni adaptace. Celkovým výsledkem je, že oko je schopno širšího rozsahu celkového rozlišení intenzity. [1][2]

2 KOMUNIKACE S HW

2.1 Sériová linka

Datová komunikace je důležitá pro komunikaci mezi počítači, pro samostatná zařízení, jako je např. GPS, v místní síti (LAN) a na internetu. Pokaždé, když chcete odeslat nebo přijmout online zprávu, odeslat nebo přijmout SMS (krátkou zprávu), odeslat nebo přijmout e-mail, přistupovat na webovou stránku. Na internetu nebo stahovat jakékoli informace z internetu, je zapotřebí datová komunikace. Pokud připojíte k počítači zařízení, jako je například zařízení GPS, paměťové zařízení USB, mobilní telefon nebo fotoaparát, je třeba zajistit datovou komunikaci, je rovněž nutná datová komunikace. Počítač s možností externí komunikace se často označuje jako uzel. A sada uzlů musí používat stejnou strukturu dat a komunikačních médií, aby spolu mohly komunikovat. [6]

Tato struktura je definována dokumentem protokolu, který popisuje podrobnosti, jak používat komunikační média a jak interpretovat informace o datech. Protokol je soubor pravidel, která se snaží strukturovat odesílání a přijímání informací. To lze přirovnat ke dvěma lidem, kteří se snaží komunikovat, také potřebují mít soubor pravidel, aby si rozuměli, jako když používají jazyk, kterým si oba rozumí, nemluvit současně a nebýt od sebe příliš daleko. [6][7]

2.1.1 Fyzická vrstva

Fyzická vrstva může být drátová, optická nebo bezdrátová (Stallings 2001) a popisuje způsob připojení komunikačních médií, jak přenášet data na komunikačních médiích a jak zpracovávat datový tok na komunikačním médiu. Kabelové připojení bude buď paralelní, nebo sériové. Paralelní využívá pro každý bit jeden vodič, zatímco sériové zpracovává každý bit v časových slotech. na stejném vodiči. Paralelní komunikace je rychlejší, ale sériová je levnější a lze ji používat na větší vzdálenosti. Takže se nejčastěji používá sériová komunikace. Optická komunikace používá jako komunikační nosič světlo (laser) místo elektrického proudu jako u kabelového spojení. Bezdrátové mohou být protokoly jako Bluetooth a WiFi. (Stallings 2001), které jako komunikační nosič používají rádiové frekvence. Tato kapitola se dále zaměří na pouze na drátová připojení pro sériovou datovou komunikaci. [6][7]

2.1.2 Asynchroní seriová komunikace

Sériová datová komunikace může být synchronní nebo asynchronní, přičemž asynchronní komunikace je nejčastěji používaná.

V synchronní komunikaci probíhá nepřetržitý tok datových bitů a pro synchronizaci odesílatele a přijímače je nutný externí synchronizační signál. Tento synchronizační signál vyžaduje použití dalšího vodiče.

Asynchronní komunikace rozdělí datový tok na znaky a přidá ke každému znaku v datovém toku synchronizační bity. Externí synchronizační signál není potřeba, nicméně přenos dat trvá o něco déle, protože spolu s datovými bity musí být přenášeny i synchronizační bity.

Protokoly RS-232C, RS-422 a RS-485 jsou některé z příkladů asynchronních protokolů často používaných v počítačových systémech. Před zahájením přenosu datového proudu je nutné nastavit sadu komunikačních parametrů:

- RS-232C má obvykle pouze jeden vysílací a jeden přijímací uzel.
- RS-422 má obvykle jeden vysílací uzel a několik přijímacích uzlů.
- RS-485 má obvykle několik jak vysílacích, tak přijímacích uzlů.

Počet datových bitů a stop bitů lze konfigurovat, zatímco jeden start bit je pevný. Mezi konfigurovatelné komunikační parametry patří:

Baud rate: Definuje rychlost přenosu, počet bitů za sekundu. Běžné hodnoty baud rate jsou 300, 600, 1200, 2400, 4800, 9600, 19200, 38400 a 115200.

Datové bity: Definuje počet bitů v každém přenášeném znaku. Počet datových bitů může být 5, 6, 7 nebo 8.

Kontrola parity: Jednobitový součet, který může být žádný, sudý nebo lichý. Sudá nebo lichá parita přidá k datovému proudu další bit před stop bitem.

Stop bity: Počet bitů za datovými bity a případným paritním bitem. Počet stop bitů může být 1, 1,5 nebo 2.

Start bit se používá pro synchronizaci, protože přijímače nevědí, kdy kterýkoli uzel vysílá. Start bit obvykle mění úroveň elektrického napětí nosné vlny z vysoké na nízkou, což znamená, že budou přenášena nová data. Start bit také synchronizuje systém baud rate pro přijímač, protože systém baud rate se používá k určení, kdy má být přenesen nový bit a kdy jej přijímač může přečíst. [6][7]

Paritní bit se přidá za datové bity a před stop bity, pokud je nastavena sudá nebo lichá parita. Tento extra bit bude 0 nebo 1, aby splňoval konfiguraci parity.

Pokud jsou datové bity 1101 1111 bude počet jedniček 7, což je lichý počet bitů. Pokud je zvolena lichá parita, bude extra paritní bit 0, nebo pokud je zvolena sudá parita, bude extra paritní bit 1.

Kontrola paritního bitu může být užitečná při komunikaci v hlučném prostředí nebo na velké vzdálenosti.

Převod mezi datovým bajtem a datovými bity a zahrnutí komunikačních parametrů zajišťuje hardwarový čip nazvaný univerzální asynchronní přijímač a vysílač (UART). UART bude řídit baud rate, počet datových bitů, volitelnou kontrolu parity a zahrne start a stop bity. Rámec tvoří datové bity a je součástí celého paketu. [7]

UART bude také detekovat chyby rámce, což znamená jakékoli chyby v počtu datových bitů. Důvodem může být nesprávná konfigurace baud rate nebo nesprávný počet datových bitů.

UART také obsahuje zásobník typu FIFO (First In First Out) pro přijímací část, například pro 16 bajtů, kromě kontroly parity a chyb rámce. FIFO zásobník umožňuje sériovému portu přijímat více znaků, než je softwarový ovladač schopen přečíst.

Řízení UART zajišťují ovladače operačního systému a většina programovacích jazyků má hotové knihovny pro komunikaci s těmito ovladači. Tento dokument se zaměří na knihovnu sériového portu v jazyce C#.

2.1.3 Komunikační protokoly

Než se pustíme do programování sériového portu, prozkoumáme dva asynchronní sériové protokoly, které budou použity jako příklady v programovací části. Jedná se o protokol Národní asociace pro mořskou elektroniku (NMEA) 183, který se často používá pro GPS (Global Position System), a diagnostiku palubního počítače (OBD), která se používá jako síť ve většině vozidel vyrobených po roce 2000. [6][7]

NMEA 183 je jednosměrný protokol, takže vysílač cyklicky vysílá všechny potřebné informace a každý přijímač musí tyto informace poslouchat a ukládat pouze potřebné údaje. Vysílačem tedy může být pouze jeden uzel, ostatní uzly budou přijímače. [6]

Protokol OBD je obousměrný protokol, kde všechny uzly mohou být vysílačem nebo přijímačem. Každý uzel může odeslat požadavek do sítě a uzel s informacemi odpoví v rámci specifického časového rámce.

Než se však ponoříme hlouběji do těchto dvou různých protokolů, je třeba krátce uvést připojení k sériovému portu. [6]

Sériový port

Starší počítače měly jeden nebo několik portů RS232C nebo RS485, zatímco moderní počítače mají pouze porty USB. Existuje však mnoho adaptérů USB na sériový port.

Tři nejdůležitější piny konektoru sériového portu jsou pin pro příjem dat (RD) (rx: pin 2), pin pro odesílání dat (TD) (tx: pin 3) a zemnící pin (GND) (pin 5). Pin 2 je připojení Rx (příjem) a pin 3 je připojení Tx (odesílání).

Jedním ze způsobů testování takového adaptéru je propojení pinu 2 a pinu 3 a provedení testovací volby v softwaru, která spočívá pouze v odeslání testovacího textu a kontrole jeho příjmu. Tato testovací možnost se často nazývá testováním smyčky a otestuje jak hardware, tak software daného uzlu. [6][7]

Protokol NMEA

Protokol NMEA 183 je cyklický protokol, kde vysílací uzel periodicky odesílá sadu zpráv. Tento protokol se používá v mnoha GPS zařízeních. Jedním z nich je Pharos iGPS180.

Jedná se o samostatné zařízení napájené z USB portu (5 V). Protokol je založen na standardu RS-232C, používá přenosovou rychlost 4800 baudů, 8 datových bitů, bez parity a jeden stop bit. GPS zařízení má integrovanou anténu a pro přenos informací o poloze využívá protokol NMEA. [7]

OBD

Standard diagnostiky palubního počítače (OBD-2) je povinný pro všechny vozy prodávané ve Spojených státech od roku 1996 a jedná se o protokol pro síť senzorů, akčních členů a řídicích jednotek uvnitř automobilů. Evropská diagnostika palubního počítače (EOBD) je evropský ekvivalent amerického standardu OBD II, který platí pro benzínové vozy prodávané v Evropě od roku 2001 (a naftové vozy o tři roky později). [9]

Automobily využívají sběrnici CAN (Controller Area Network) pro komunikaci mikrokontrolérů a zařízení bez nutnosti serveru, takže sběrnice CAN je místní síť (LAN) pro

všechna zařízení instalovaná v moderním automobilu. Protokol OBD definuje port připojení, signály a příkazy.

Moderní automobil obsahuje průmyslový řídicí systém s několika distribuovanými řídicími systémy, jako je řídicí jednotka motoru, řídicí jednotka brzdového systému (ABS), řídicí jednotka převodovky, řídicí jednotka odpružení a řídicí jednotka klimatizace. Jednou z možností externí komunikace s touto sítí (LAN) je použití portu OBD.

Protokol CAN je složitý protokol, jehož implementace v softwaru může být obtížná. Lepší volbou je použití rozhraní ELM327, které je převodníkem protokolu a převádí mezi příkazy CAN a typem příkazů AT (Attention). Příkazy AT se běžně používají pro komunikaci s modemy, kde příkazy obvykle začínají znakem AT. Implementace ELM327 přizpůsobila sadu těchto příkazů (Elm 2016). Port OBD a rozhraní ELM327.

2.1.4 Programování sériového portu

Programování sériového portu v C# je pomocí standardní knihovny sériového portu poměrně jednoduché. Stačí do kódu zahrnout knihovnu sériového portu, a poté lze sériový port používat jako textový soubor, což znamená, že port lze otevřít, číst z něj, zapisovat do něj a zavřít jej. [9][7]

Otevření portu je exkluzivní, takže v daném okamžiku může specifický sériový port otevřít pouze jedna aplikace. Proto je důležité zkontrolovat, zda je odpověď na otevření platná. Z téhož důvodu nezapomeňte uzavřít všechny sériové porty, které již vaše aplikace nepoužívá, a to i při přechodu z jednoho sériového portu na jiný. [9]

2.1.5 Knihovna sériového portu

Knihovna sériového portu obsahuje sadu předdefinovaných metod, které lze použít k manipulaci s libovolným sériovým portem na počítači. Knihovna umožňuje získat seznam dostupných sériových portů, vybrat jeden z dostupných sériových portů, a otevřít sériový port s vhodnými komunikačními parametry, jako jsou baudová frekvence, datové bity, parita a stop bity, před manipulací s jakýmikoli informacemi protokolu. [9][7]

2.1.6 Knihovna sériového portu v C#

Knihovna sériového portu v jazyce C# umožňuje vývojářům komunikovat sériovými porty v operačním systému Windows. Knihovna je součástí .NET Frameworku a je dostupná ve všech verzích jazyka C#.

Knihovna sériového portu se skládá z následujících tříd:

SerialPort: Tato třída představuje sériový port.

SerialStream: Tato třída umožňuje číst a zapisovat data do sériového portu.

Parity: Tato třída definuje různé typy parity, které lze použít sériovou komunikací.

StopBits: Tato třída definuje různé typy stop bitů, které lze použít sériovou komunikací.

2.2 Programování síťové komunikace v jazyce C#

2.2.1 Úvod

Datová komunikace je důležitá pro komunikaci mezi počítači, jak na lokálních sítích (LAN) (intranetu), tak na internetu. Předchozí kapitola se zabývala komunikací sériového portu, využívající sériové porty pro externí komunikaci. V této kapitole bude zaměření na síťovou datovou komunikaci, což znamená komunikaci dat přes síť. Datová komunikace sériového portu má omezený počet vysílačů a přijímačů na stejném komunikačním médiu, a obvykle má také omezenou délku komunikačního média. Maximální délka je často závislá i na baudové frekvenci pro sériovou datovou komunikaci. Síťová datová komunikace využívá složitější strukturu s hardwarovými zařízeními, jako jsou přepínače, routery, brány a ochranné zdi (firewally), k propojení síťových systémů, směrování zpráv mezi těmito síťovými systémy a využívání ochranných zdí k určení, které zprávy mohou být přijaty každou podsítí.

To znamená, že síťový protokol bude složitější než protokol sériového portu, nicméně model OSI může být také použit k strukturování funkcionality těchto protokolů. Protokoly sériového portu závisí především na nejnižší vrstvě v modelu OSI, zatímco síťové protokoly budou záviset na několika vrstvách v tomto modelu protokolu. Přesto musí síťová komunikace stále záviset na nějakém druhu sériové datové komunikace, například protokolu RS-485 používaného v mnoha síťových protokolech, jako fyzická vrstva. Fyzická vrstva může stále používat buď kabelové, optické nebo bezdrátové fyzické připojení pro datovou komunikaci.

2.2.2 Model OSI

Model OSI je abstraktní model používaný k strukturování libovolného komunikačního protokolu do funkcionality sedmi vrstev. Komunikace sériového portu popsaná v předchozí kapitole popisuje funkcionalitu fyzické vrstvy modelu OSI, zatímco síťová datová

komunikace je popsána funkcionalitou prvních čtyř vrstev: fyzická, datová linka, síťová a transportní vrstva. [8][10]

Protokol TCP/IP využívá funkcionality protokolu Internet Protocol (IP) pro komunikaci se uzly v síti, kde hlavní funkčnost je logické adresování. Uzel v jedné lokální síti připojené k internetu může komunikovat s jiným uzlem připojeným k jiné síti na internetu pomocí protokolu IP. Protokol IP bude využívat buď protokol Ethernet nebo Token Ring k manipulaci s funkcionalitou dvou nejnižších vrstev modelu OSI. Protokol Ethernet je dnes nejčastěji používaným protokolem pro manipulaci s internetovou komunikací na těchto nejnižších vrstvách. [8][10]

2.2.3 Protokol TCP/IP

Protokol TCP/IP je síťový protokol a je založen na konceptu klient/server. Tento koncept definuje, že musí existovat existující server s nějakými informacemi, ke kterým klient chce získat přístup. V datové komunikaci se používá socket k navázání spojení mezi dvěma nebo více uzly (počítači s komunikačními schopnostmi), a tyto sockety budou použity k odesílání dat mezi uzly. Pro protokol TCP/IP bude server nejprve vytvářet posluchačský socket a každý klient vytvoří socket k připojení k posluchačskému socketu konkrétního serveru. Pro protokol TCP/IP budou IP adresy použity jak serverem, tak klienty k vytvoření socketu pro komunikaci, takže IP adresa serveru by měla být známa. Když je spojení socketu navázáno, uzly mohou vyměňovat informace pomocí spojení socketu (metody čtení a zápisu). Až je spojení naplněno, oba systémy vytvoří datový proud pro výměnu informací čtením a zápisem. Upozorňujeme, že se jedná o systém s multitaskingem, protože obě aplikace budou běžet nezávisle na sobě. Ukončení spojení probíhá uzavřením datového proudu a socket spojení na obou koncích. [8][10]

2.3 Implementace kamerové podpory

Implementace kamerové podpory je množina tříd, které umožňují komunikaci s kamerami různých výrobců. Speciálním druhem je pak OfflineSource, který slouží k neustálému zobrazení buď sekvence, nebo statického obrazu. Všechny kamery pak implementují společné rozhraní IImageSource. Tímto je docíleno, že jsme odstíněni od konkrétního SDK kamery, ale všechna komunikace je zprostředkována přes toto rozhraní.

Načítání informací z CameraListu je implementováno ve třídě BaseCamera, což je předek pro všechny třídy, které implementují rozhraní mezi IImageSource a konkrétním SDK výrobce. Toto je důvod, proč třída OfflineSource nemá předka třídu BaseCamera.

Bázová třída BaseImageSource implementuje předávání vyčtených obrazových dat do tříd NewImage pro předání obrazových dat.

2.3.1 Podporovaná funkcionalita kamer

Podporovanou funkcionalitou kamer se myslí možnosti vyčítání a nastavení parametrů. Mezi toto funkcionalitu patří:

- Binning
- Ořez
- Změna bitové hloubky
- Vyčtení základních informací o kameře
- Nastavení expozičního času
- Nastavení zesílení
- Black Level

2.3.2 Použití kamerové podpory

Pro zobrazení streamu, ať už z kamery nebo pomocí *OfflineSource*, slouží komponenta ***VideoWindow***.

Načtení připojených kamer nebo obrázku je *ImageSourceCore* je možné pomocí dotazu na kolekci zdrojů *Instance.AvalaibleCameras* na objektu *ImageSourceCore*. Při tomto dotazu se vždy vrátí alespoň jeden zdroj, a to *OfflineSource* pro načtení obrázku.

Připojení zdroje k *VideoWindow* je možné pouze v případě, kdy není připojen jiný zdroj. Stav připojení je uložen v propertě *IsConnected*. Samotné připojení zdroje je realizováno pomocí volání metody *ConnectToCamera*. Pokud je aktuální zdroj odpojen, tzn., není zapnuté grabování, volá se metoda *Open* zdroje. V případě zdroje typu *OfflineSource* je uživatel vyzván k zadání cesty a velikosti pixelu kamery.

3 TECHNOLOGIE PRO VÝVOJ DESKTOPOVÝCH APLIKACÍ

V následujících odstavcích jsou popsány vybrané technologie pro vývoj desktopových a webových aplikací.

3.1 Programovací jazyky využívané pro vývoj desktopových/webových aplikací

3.1.1 C#

Je výkonný a moderní programovací jazyk, který kombinuje sílu a flexibilitu. Byl vyvinut společností Microsoft a je často používán pro vývoj široké škály aplikací, od desktopových programů a webových služeb až po mobilní aplikace. C# je objektově orientovaný jazyk s možností zabudovaného podpory pro událostmi řízené programování a generické typy, což umožňuje vývojářům psát čistý a efektivní kód. Díky svému silnému typování a rozsáhlé sadě knihoven nabízí C# vysokou úroveň bezpečnosti a produktivity. S jazykem C# je možné pracovat v různých vývojových prostředích, jako je například Visual Studio, což poskytuje vývojářům široké možnosti pro vytváření inovativních a robustních aplikací pro různé platformy. [11][12]

3.1.2 XAML

XAML (Extensible Application Markup Language) je deklarativní jazyk používaný především pro návrh uživatelského rozhraní v různých platformách a technologiích, zejména v ekosystému Microsoft. Jeho síla spočívá v tom, že umožňuje oddělit design uživatelského rozhraní od logiky aplikace, což usnadňuje práci designérům a vývojářům. XAML je často využíván ve spojení s jazykem C# při vývoji aplikací pro platformy jako Windows, Xamarin, UWP (Universal Windows Platform) nebo WPF (Windows Presentation Foundation). Díky své strukturované syntaxi a možnosti vytvářet hierarchické struktury UI prvků je XAML oblíbeným nástrojem pro vytváření atraktivních a interaktivních uživatelských rozhraní. Otevřenost a rozšiřitelnost jazyku XAML umožňuje vývojářům přizpůsobit si uživatelské rozhraní podle svých potřeb a vytvořit tak jedinečné a poutavé aplikace. [13][14]

3.1.3 Swift

Jazyk Swift je moderní, výkonný a bezpečný programovací jazyk vyvinutý společností Apple pro vývoj aplikací pro iOS, macOS, watchOS a tvOS. Byl navržen tak, aby byl uživatelsky přívětivý. Swift kombinuje vlastnosti tradičních objektově orientovaných jazyků

s moderními funkcemi, jako jsou generické typy a funkcionální programování. Své místo si získal i díky jednoduchosti použití a rychlosti kompilace, což zvyšuje produktivitu vývojářů. Swift je open source jazyk, což znamená, že komunita může přispívat k jeho rozvoji a inovacím. Díky těmto vlastnostem se Swift stal preferovanou volbou pro vývoj mobilních a desktopových aplikací v ekosystému Apple. [15]

3.2 Popis vybraných .NET frameworků pro vývoj desktopových aplikací

.NET je robustní softwarová platforma vyvinutá společností Microsoft, která poskytuje kompletní prostředí pro vývoj, běh a správu široké škály aplikací a služeb. Jeho flexibilita a rozmanitost umožňují vývojářům vytvářet aplikace pro různé platformy a scénáře, včetně desktopových, webových, mobilních a cloudových prostředí. [9]

Jádro platformy .NET je postaveno kolem Common Language Runtime (CLR), což je běhové prostředí, které provádí .NET kód a poskytuje klíčové funkce, jako je automatická správa paměti, správa výjimek a bezpečnostní mechanismy. CLR umožňuje integraci různých programovacích jazyků, jako je C#, Visual Basic.NET, F# a C++/CLI, což dává vývojářům široký výběr možností při výběru nejvhodnějšího jazyka pro jejich projekty. [9]

V následujících odstavcích jsou uvedeny vybrané .NET frameworky pro vývoj desktopových aplikací.

3.2.1 Universal Windows Platform (UWP)

Je framework vyvinutý společností Microsoft, který umožňuje vývoj aplikací pro širokou škálu zařízení s operačními systémy Windows, včetně nejnovějších verzí jako Windows 10 a 11, platformy Xbox a Windows Mixed Reality, která slouží pro zařízení jako Microsoft HoloLens. Tento framework byl poprvé představen v roce 2015 a rychle se stal oblíbenou volbou pro vývojáře díky své flexibilitě a širokému spektru podporovaných zařízení. [9]

Při vývoji aplikací pro UWP se často využívá vývojové prostředí Visual Studio od společnosti Microsoft, které nabízí bohaté možnosti pro tvorbu, ladění a nasazování aplikací. UWP umožňuje psát aplikace v různých programovacích jazycích, což zahrnuje populární jazyky jako C#, C++, Visual Basic a JavaScript pro backendové části aplikace. Pro frontendové části jsou k dispozici technologie jako XAML, HTML, DirectX a WinUI, což umožňuje vývojářům vytvářet moderní uživatelská rozhraní a interaktivní zážitky pro uživatele. [9]

Díky podpoře různých zařízení a operačních systémů poskytuje UWP vývojářům možnost dosáhnout širokého publika a zároveň optimalizovat své aplikace pro konkrétní platformy a zařízení. Tím se zajišťuje konzistentní uživatelská zkušenost a zvyšuje se efektivita vývoje pro různé cílové platformy. [9]

3.2.2 Windows UI Library 3 (WinUI 3)

Je nativní komponenta uživatelského rozhraní vyvinutá společností Microsoft a je součástí balíku Windows App SDK. Tento balík poskytuje standardizovanou sadu API rozhraní a nástrojů určených pro tvorbu desktopových aplikací pro operační systém Windows 10 a novější verze. [9]

WinUI 3, jako nástupce UWP frameworku, nabízí širokou škálu možností pro tvorbu moderních desktopových aplikací. To zahrnuje nejen vývoj desktopových aplikací, ale také podporu pro aplikace UWP. Programování aplikací v WinUI je možné v jazycích jako C#, C++, Visual Basic a JavaScript. Tento framework nabízí vývojářům flexibilitu a moderní nástroje, aby mohli vytvářet aplikace, které splňují nejnovější standardy a požadavky uživatelů Windows platformy. [9]

3.2.3 Windows Presentation Foundation (WPF)

WPF, neboli Windows Presentation Foundation, představuje vývojový framework určený pro tvorbu desktopových aplikací a je nedílnou součástí .NET frameworku. V aplikacích využívajících WPF je uživatelské rozhraní definováno pomocí jazyka XAML, zatímco pro zpracování logiky aplikace slouží jazyk C#.

Původně byl WPF navržen jako náhrada za starší framework WinForms. Avšak jeho přijetí mezi vývojáři probíhalo pomalu, zejména kvůli jeho rozsáhlosti a výrazným odlišnostem oproti WinForms, se kterými byli vývojáři zvyklí pracovat. Nicméně postupně se situace změnila. Vývojáři se začali adaptovat na nové prostředí a nové pracovní postupy, a XAML přestal být pouze další komplikovanou částí, kterou je třeba naučit se obsluhovat. Stal se pohodlným a výkonným nástrojem pro vytváření uživatelských rozhraní.

S nástupem WPF přišly také nové návrhové vzory, především Model-View-ViewModel (MVVM), který je variací jiných existujících vzorů oddělujících view a data, jako jsou MVC a MVP. Tyto vzory nejenom zjednodušily práci vývojářů, ale především nastavily standard pro interakci mezi view a daty.

WPF rovněž položil základy pro další technologie postavené na podobných principech, jako je Silverlight (pro vývoj webových klientů v .NET), Windows Phone 7 (mobilní operační systém od Microsoftu využívající variantu Silverlight) a v poslední době Windows 8 a Windows Phone 8. Všechny tyto platformy jsou postaveny na podobných konceptech jako XAML, vlastnosti závislostí, šablony, styly a vazby, což jasně ukazuje sílu a význam WPF. [9][13]

Výhody WPF:

- **Široká integrace** – Před příchodem WPF musel vývojář, který chtěl pracovat s 3D grafikou, videem a dalšími multimediálními prvky, ovládat několik různých technologií a integrovat je do svého programu. S příchodem WPF se tato situace změnila, protože WPF poskytuje konzistentní programovací model a úzkou integraci, která umožňuje práci s těmito prvky pod jednotným rámcem.
- **Hardwarová akcelerace** – WPF je postaven na 3D technologii, což znamená, že veškerý obsah ve WPF aplikaci (ať už 2D, 3D, text nebo grafika) je převeden na 3D trojúhelníky, textury a další objekty Direct3D a následně je renderován hardwarově. Tímto způsobem WPF využívá výkon hardwaru pro dosažení plynulejší grafiky a lepšího výkonu.
- **Deklarativní programování** – Zatímco deklarativní programování není unikátní vlastností pouze WPF (jelikož i starší Win16/Win32 aplikace využívaly deklarativní zdrojové skripty), WPF přinesl deklarativní programování na novou úroveň díky jazyku Extensible Application Markup Language (XAML). Kombinace WPF a XAMLu je podobná používání HTML k definování uživatelského rozhraní, ale s větší flexibilitou a možnostmi. XAML se v rámci WPF používá nejen pro definici uživatelského rozhraní, ale i pro reprezentaci 3D modelů a dalších prvků.
- **Kompozice a přizpůsobení** – Prvky ovládacích prvků ve WPF lze kombinovat a přizpůsobovat různými způsoby. Například uživatel může vytvořit ComboBox a naplnit jej animovanými tlačítky nebo menu obsahujícími živé videoklipy. To umožňuje uživatelům upravovat ovládací prvky bez nutnosti psaní značného množství kódu. [13]

3.3 Popis vybraných alternativních frameworků pro vývoj desktopových aplikací

V následujících odstavcích jsou uvedeny vybrané frameworky pro vývoj desktopových aplikací, které byly zařazeny na základě hodnocení a doporučení z článku od společnosti Solace Infotech [16] a některé další frameworky.

3.3.1 SwiftUI

Je inovativní framework vyvinutý společností Apple pro vývoj aplikací určených pro širokou škálu operačních systémů této společnosti, včetně MacOS, iOS, watchOS a dalších. Tento framework byl uveden na trh v roce 2019 a rychle získal popularitu mezi vývojáři díky své jednoduchosti a vysoké produktivitě. [17]

Jako hlavní programovací jazyk pro vývoj aplikací v SwiftUI se používá Swift, což umožňuje vývojářům vytvářet moderní a efektivní kód. Jedním z významných benefitů SwiftUI je jeho těsná integrace přímo do vývojového prostředí Xcode od společnosti Apple, což zjednodušuje proces vývoje a ladění aplikací. [17]

Další výhodou tohoto frameworku je jeho nativní podpora pro různá zařízení a operační systémy od Apple, což umožňuje vývojářům vytvářet aplikace s konzistentním uživatelským rozhraním a chováním napříč různými zařízeními. Díky těmto vlastnostem poskytuje SwiftUI robustní prostředky pro vývoj moderních aplikací, které efektivně využívají možnosti a funkce nabízené platformami od společnosti Apple. [17]

3.3.2 Qt

Jde o open-source framework navržený pro vývoj multiplatformních aplikací, které jsou schopny běžet na různých operačních systémech, včetně Linuxu, MacOS, Windows, Androidu, iOS a dalších, jako je například QNX. Při vývoji aplikací v tomto frameworku je nejběžněji využíván programovací jazyk C++, ale podporovány jsou i další jazyky, jako například Python nebo Qt QML. Qt framework nabízí řadu výhod, mezi něž patří jeho rychlost a schopnost pracovat na více operačních systémech, což umožňuje vývojářům snadněji dosáhnout multiplatformního pokrytí a šíře publiku pro své aplikace. [18]

3.3.3 Electronjs

Electron.js, open-source framework, spatřil světlo světa v roce 2013 díky iniciativě vývojáře Cheng Zhao v rámci společnosti GitHub. Po tříletém intenzivním vývoji bylo v roce 2016

vydáno první stabilní API ve verzi 1.0. Tento framework přináší vývojářům široké možnosti pro tvorbu aplikací, a to díky podpoře jazyků HTML, CSS a JavaScript. Což umožňuje vytvářet moderní a uživatelsky přívětivé desktopové aplikace. Jedním z klíčových aspektů Electron.js je jeho multiplatformní povaha, což znamená, že umožňuje vývoj aplikací pro různé operační systémy, jako jsou Windows, MacOS a Linux. Tato univerzálnost je pro vývojáře obzvláště cenná, neboť minimalizuje náklady a úsilí spojené s vytvářením a udržbou různých verzí aplikací pro každý operační systém zvlášť. Díky Electron.js mají vývojáři možnost rychle a efektivně nasadit své aplikace na trh a oslovit široké spektrum uživatelů bez ohledu na jejich preferovaný operační systém. Tímto způsobem framework přináší výrazné zjednodušení vývojového procesu a zvyšuje konkurenceschopnost vytvářených aplikací na trhu. [19]

3.3.4 Swing

Swing je framework vyvinutý společností Oracle jako součást Java Foundation Classes. Poskytuje prostředky pro vytváření uživatelských rozhraní pro Java aplikace. Podobně jako Electron.js, Swing je navržen tak, aby byl multiplatformní, což umožňuje vývoj desktopových aplikací pro různé operační systémy. Jeho hlavním programovacím jazykem pro vývoj aplikací je Java. [20]

3.3.5 Cocoa

Tento framework je speciálně navržen pro tvorbu nativních, objektově orientovaných aplikací určených pro operační systém MacOS od firmy Apple. Pro vývoj aplikací v rámci tohoto frameworku je běžně využíváno vývojového prostředí Xcode a programovacích jazyků Swift a Objective-C. Avšak existuje možnost použít i další vývojová prostředí a programovací jazyky, což umožňuje vývojářům flexibilitu při tvorbě aplikací pro MacOS. Tento framework poskytuje uživatelům prostředky pro efektivní vývoj aplikací, které jsou plně integrovány do ekosystému Apple a optimalizované pro běh na zařízeních s operačním systémem MacOS. [21]

4 NÁVRHOVÉ VZORY

Návrhové vzory jsou užitečné nástroje, které nám usnadňují vytvoření přehledných a opakovaně použitelných řešení pro náš projekt. Tyto vzory nám typicky ukazují, jak by měly jednotlivé objekty vzájemně spolupracovat nebo jak by měly fungovat. Je důležité si uvědomit, že tyto vzory nejsou pevně dané pravidlo, ale spíše doporučení. I když není nezbytné se jimi striktně řídit, je to často prospěšné.

V softwarovém inženýrství existují tři hlavní kategorie návrhových vzorů. První skupina, Creational patterns, se zabývá vytvářením objektů a jejich instancí. Patří sem například Singleton Pattern, který zajišťuje existenci jediné instance třídy. Druhá skupina, Structural patterns, se soustředí na organizaci objektů a tříd do efektivních struktur, často využívající kompozici existujících objektů. Třetí skupina, Behavioral patterns, se zaměřuje na interakce mezi třídami a definuje způsoby, jak třídy komunikují a řeší různé scénáře chování. Tyto kategorie poskytují užitečný rámec pro návrh a implementaci softwarových systémů, což pomáhá vytvářet efektivní a snadno udržovatelný kód. [23]

4.1 MVVM

MVVM, zkráceně Model-View-ViewModel, je návrhový vzor navržený k oddělení programové logiky od uživatelského rozhraní. Jeho hlavním cílem je rozdělit kód do modulů, což usnadňuje práci, umožňuje snadné úpravy a nahrazování jinými moduly. Díky tomu je kód přehlednější a lépe udržovatelný. [11][23]

Tento návrhový vzor je běžně využíván pro vývoj aplikací v rámci platformy Windows Presentation Foundation (WPF). Následující odstavce popisují jednotlivé části tohoto vzoru s využitím informací z dostupných zdrojů a článků. [11][23]

4.1.1 Model

Model představuje datový model aplikace, který obsahuje informace a logiku potřebnou pro zpracování dat. Tento komponent reprezentuje podnikovou logiku aplikace a nezávisí na uživatelském rozhraní. V MVVM je Model tedy zodpovědný za uchování a manipulaci s daty aplikace.

4.1.2 View

View je vizuální komponenta aplikace, která zobrazuje uživatelské rozhraní a interaguje s uživatelem. To může zahrnovat okna, dialogové okna, prvky ovládání a další. V rámci

MVVM se View zaměřuje pouze na prezentaci dat a neobsahuje žádnou logiku týkající se zpracování dat.

4.1.3 ViewModel

ViewModel je spojovacím prvkem mezi Modelem a Viewem. Obsahuje prezentaci dat, která jsou vyžadována uživatelským rozhraním, a zprostředkovává komunikaci mezi Modelem a Viewem. ViewModel obsahuje také logiku potřebnou pro interakci s uživatelem a zpracování událostí. V MVVM se ViewModel často implementuje jako rozhraní, které je propojeno s odpovídajícím Viewem pomocí datové vazby.

Tento přístup k vývoji aplikací umožňuje snadnější správu kódu, oddělení logiky od prezentace a zlepšuje testovatelnost aplikace. Využití MVVM je běžnou praxí při vývoji aplikací pro platformu WPF, díky čemuž je možné dosáhnout efektivního a strukturovaného vývoje softwaru.

4.2 MVC

MVC, neboli Model-View-Controller, je návrhový vzor, který byl navržen za účelem oddělení datového modelu aplikace (Modelu), uživatelského rozhraní (View) a řídící logiky (Controlleru). Jeho hlavním cílem je rozdělení kódu do modulů, což usnadňuje práci, umožňuje snadné úpravy a nahrazování jinými moduly, a tím zlepšuje přehlednost kódu.

4.2.1 Model

Model představuje jádro návrhového vzoru MVC a obsahuje data a funkce určené pro manipulaci s těmito daty. Jeho úkolem je udržovat stav aplikace a provádět operace nad těmito daty. Důležité je, aby Model neobsahoval žádnou logiku týkající se zobrazení dat uživateli a aby byl nezávislý na uživatelském rozhraní.

4.2.2 View

View slouží k vizuálnímu zobrazení dat Modelu uživateli. Může se jednat o grafické prvky, tabulky, diagramy a další komponenty. Jeho hlavní funkcí je zobrazovat data uživateli a starat se o jejich vzhled. View by měl být pasivní a neobsahovat žádnou logiku týkající se zpracování dat. [23]

4.2.3 Controller

Controller je prostředníkem mezi View a Modelem. Jeho úkolem je reagovat na události vyvolané uživatelem v View a provádět odpovídající akce. Například, pokud View vyvolá událost, Controller na ni reaguje a provádí příslušné akce, například volání metod v Modelu. Controller je zodpovědný za řízení toku dat a provádění business logiky aplikace. [11]

Tímto způsobem umožňuje návrhový vzor MVC efektivní správu kódu, oddělení zobrazení od logiky a zlepšení testovatelnosti aplikace. MVC je široce využívaným přístupem při vývoji aplikací, a to nejen v rámci desktopových aplikací, ale také v webovém vývoji a dalších oblastech softwarového inženýrství. [11]

4.3 SOLID

Tyto principy, známé jako SOLID, jsou základními zásadami návrhu softwarové architektury, které definoval Robert C. Martin. Každý z těchto principů přináší důležité směrnice pro tvorbu a údržbu softwarových systémů. Následující odstavce popisují jednotlivé principy a jejich význam, jak jsou definovány v uvedených zdrojích.

4.3.1 Single Responsibility Principle (SRP):

Princip jediné zodpovědnosti říká, že každá třída by měla mít pouze jednu důvodnou zodpovědnost. Tímto způsobem je zachována jednoduchost, čitelnost a údržba kódu je snazší. Každá třída by měla být zaměřena na jednu oblast funkcionality, což umožňuje snadnější rozšiřování a úpravy.

4.3.2 Open/Closed Principle (OCP):

Princip otevřenosti a uzavřenosti zdůrazňuje, že třídy by měly být navrženy tak, aby bylo možné jejich chování rozšiřovat bez nutnosti úprav v existujícím kódu. To znamená, že by měly být otevřené pro rozšíření, ale uzavřené pro modifikaci. Tím je minimalizováno riziko poškození existující funkcionality při úpravách.

4.3.3 Liskov Substitution Principle (LSP):

Princip substituce Liskovové říká, že instance odvozené třídy by měly být plně nahraditelné instancemi základní třídy, aniž by to ovlivnilo správnost programu. To znamená, že by se měly chovat stejně a vracet stejné výsledky jako základní třída, aniž by uživatel musel vědět o konkrétní implementaci.

4.3.4 Interface Segregation Principle (ISP):

Princip segregace rozhraní stanovuje, že by rozhraní měla být co nejmenší a třídy by neměly být nuceny implementovat metody, které nepotřebují. Tímto způsobem se minimalizuje vazba mezi třídami a zvyšuje se modularita a flexibilita systému.

4.3.5 Dependency Inversion Principle (DIP):

Princip inverze závislostí stanovuje, že by třídy měly záviset na abstrakcích, nikoliv na konkrétních implementacích. Tím se snižuje závislost mezi třídami a umožňuje snadnější změny a testování kódu.

Tyto principy poskytují směrnice pro tvorbu robustních, flexibilních a snadno udržovatelných softwarových systémů. Jejich dodržování přispívá k vytváření kvalitního a efektivního kódu, který je odolný vůči změnám a snadno rozšiřitelný.

.

5 BEZPEČNOST

Zabezpečení aplikací je důležité, obzvláště u webových aplikací připojených k internetu, jelikož jsou vystaveny možnému útoku odkudkoliv.

5.1 Nejčastěji vyskytující se bezpečnostní rizika

V následující části jsou popsány vybraná bezpečnostní rizika, která byla vybrána podle článku ze stránky OWASP [26]. Při popisu bezpečnostních rizik se čerpá s článků [27][28].

5.1.1 Denial of service (DoS) a Distributed denial of service (DDoS)

Jedná se o útok, kdy se útočníci snaží přetížit server. V důsledku přetížení může docházet k zpomalení odpovědí serveru nebo dokonce k úplnému znemožnění přístupu uživatelům. Proti útokům typu DDoS lze přijmout opatření, jako je zabezpečení serverů, kontrola vstupů (například omezení maximálního počtu dotazů nebo maximální velikosti nahrávaných souborů), kontrola přístupu a další.

5.1.2 Cross site scripting (XSS)

Bezpečnostní hrozba tkví v možnosti, že útočník vloží do webové aplikace na straně klienta skripty, jež by umožnily přístup k citlivým údajům nebo by mohly usilovat o získání těchto informací od ostatních uživatelů. Toto riziko lze eliminovat s pomocí moderních frameworků, které nabízejí určitou úroveň ochrany, například skrze kódování výstupů.

5.1.3 Server side request forgery

Bezpečnostním rizikem je situace, kdy webová aplikace nedokáže ověřit platnost URL adres, které uživatel poskytne před stažením dat ze zdroje. Tato zranitelnost může být zneužita k útokům, jako je vložení škodlivých nebo nebezpečných odkazů, které mohou vést k různým bezpečnostním problémům, včetně stahování nebezpečného obsahu, phishingových útoků nebo injekce škodlivého kódu. Aby se tomuto riziku předešlo, je důležité provádět validaci URL adres, které uživatelé poskytnou, a to buď pomocí regulárních výrazů nebo jiných mechanismů ověřování, aby se zajistilo, že jsou adresy bezpečné a legitimní. Tímto způsobem lze minimalizovat riziko zneužití a zajistit bezpečnost webové aplikace.

5.1.4 SQL Injection (SQi)

Tato metoda spočívá v hledání slabých míst v způsobu, jakým databáze provádí vyhledávací dotazy, a následném snaze útočníka dostat se k citlivým informacím. Jedním z opatření, která lze přijmout k prevenci této hrozby, je používání parametrizovaných SQL dotazů, které využívají parametry místo řetězců.

5.1.5 Broken access control

Bezpečnostní riziko spočívá v skutečnosti, že útočníkovi může být umožněn neautorizovaný přístup a práva, často se prezentujíc za administrátora systému. Takový útok může mít zničující důsledky, umožňuje útočníkovi manipulovat s daty, měnit nastavení systému a potenciálně poškodit důvěrnost, dostupnost a integritu informací. Aby se předešlo této situaci, je nezbytné provádět správné řízení přístupů na základě uživatelských rolí, což zahrnuje přidělování přístupových práv pouze těm uživatelům, kteří je skutečně potřebují pro svou práci. Kromě toho je nezbytné provádět pravidelné kontroly a testování autorizace, aby se zajistilo, že jsou dodržovány stanovené bezpečnostní politiky a že není narušena integrita systému. Tyto opatření společně pomáhají minimalizovat riziko neoprávněného přístupu a ochránit citlivé informace a zdroje před útoky.

II. PRAKTICKÁ ČÁST

6 ÚVOD K PRAKTICKÉ ČÁSTI

Tato část diplomové práce se zaměřuje na prezentaci a analýzu aplikace vyvinuté za účelem demonstrace technik zpracování obrazu a integrace hardwarových zařízení s využitím .NET Frameworku. Tato aplikace slouží jako multifunkční nástroj, umožňující uživatelům připojit různá zařízení, včetně kamer, power meterů a laserových systémů, a efektivně s nimi pracovat přímo z uživatelského rozhraní.

V důsledku rostoucí potřeby v oblasti sběru a analýzy obrazových dat se aplikace zaměřuje na usnadnění procesu integrace a ovládání těchto zařízení, aby uživatelé mohli rychle a přesně získat požadované informace. Hlavním cílem je vytvořit prostředí, ve kterém uživatelé mohou snadno manipulovat s obrazovými daty a provádět analýzy obrazových dat bez potřeby hlubších znalostí programování nebo zpracování obrazu.

Jedním z hlavních prvků aplikace je modul zpracování obrazu, který umožňuje uživatelům provádět různé analýzy obrazových dat. To zahrnuje měření charakteristik spotů na obraze, výpočet velikosti, kruhovitosti a vzdálenosti mezi nimi. Dále je umožněno ukládat všechna naměřená data do databáze pro další analýzu a archivaci.

Následující kapitoly se budou podrobně zabývat architekturou aplikace, implementačními detaily, testováním a výsledky experimentů, které nám poskytnou ucelený pohled na výkonnost a možnosti této aplikace.

7 NÁVRH APLIKACE

V následující části jsou uvedeny funkční a nefunkční požadavky spolu s případy užití.

7.1 Funkční požadavky

- Získávání obrazových dat z kamer: Aplikace bude schopna aktivně získávat obrazová data z připojených kamer.
- Nastavení parametrů kamery: Uživatelé budou moci nastavovat různé parametry kamer, jako je expozice, zaostření nebo kontrast, prostřednictvím aplikace.
- Detekce objektů: Implementace algoritmů umožní detekci objektů v obrazových datech.
- Analýza obrazu: Aplikace bude schopna provádět různé analýzy obrazových dat.
- Ukládání a zpracování dat: Aplikace bude umožňovat ukládání obrazových dat a jejich následné zpracování, například pro historické analýzy nebo trénování modelů strojového učení.
- Zobrazení obrazu: Aplikace bude poskytovat uživatelům prostředky pro zobrazení obrazových dat v reálném čase nebo v různých formátech (např. fotografie, video).

7.2 Nefunkční požadavky

- Uživatelské rozhraní: Uživatelské rozhraní bude navrženo tak, aby bylo intuitivní a snadno ovladatelné pro uživatele.
- Výkon a rychlost: Aplikace bude optimalizována pro rychlou odezvu a plynulý provoz, což je důležité zejména při zpracování obrazu v reálném čase.
- Odezva systému: Systém bude muset poskytovat rychlou odezvu na uživatelské interakce, aby byla zajištěna efektivní práce s hardwarem a zpracování obrazu.

7.3 Případy užití aplikace pro zpracování obrazu a komunikaci s hardwarem

V této kapitole jsou popsány základní případy užití aplikace vytvořené jako součást praktické části diplomové práce, které popisují základní funkcionalitu aplikace.

Tabulka 1 – Scénář pro vytvoření nového měření

Název: Vytvoření nového měření		
ID: UC001		
Charakteristika: Případ popisující vytvoření nového měření		
Primární aktér: Uživatel		
Vedlejší aktéři: Žádní		
Vstupní podmínky: Uživatel již vytvořil PN, systém je připojený k databázi.		
Výstupní podmínky: Do databáze bude přidán nový záznam, Aktuální měření v aplikaci se změní na právě vytvořené.		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Uživatel	Uživatel klikne na File -> New Measurement
2	Systém	Aplikace zobrazí vyskakovací okno s formulářem pro vyplnění potřebných údajů.
3	Uživatel	Uživatel vyplní formulář a následně kline na OK
4	Systém	Po úspěšném vyplnění formuláře aplikace vytvoří v databázi nový záznam měření a jako aktuální měření v aplikace nastaví právě vytvořené.

Tabulka 2 – Scénář Vytvoření nového Part Numberu (PN)

Název: Vytvoření nového PN
ID: UC003
Charakteristika: Případ popisující vytvoření nového měření
Primární aktér: Uživatel
Vedlejší aktéři: Žádní
Vstupní podmínky: Systém je připojený k databázi.

Výstupní podmínky: Do databáze bude přidán nový záznam.		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Uživatel	Uživatel klikne na File -> New PN
2	Systém	Aplikace zobrazí vyskakovací okno s formulářem pro vyplnění potřebných údajů.
3	Uživatel	Uživatel vyplní formulář a následně kline na OK
4	Systém	Po úspěšném vyplnění formuláře aplikace vytvoří v databázi nový záznam.

Tabulka 2 – Scénář popisující otevření měření

Název: Otevření měření		
ID: UC005		
Charakteristika: Případ popisující otevření měření		
Primární aktér: Uživatel		
Vedlejší aktéři: Žádní		
Vstupní podmínky: Uživatel již vytvořil měření, systém je připojený k databázi.		
Výstupní podmínky: V aplikaci se změní aktuállní měření na právě zvolené.		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Uživatel	Uživatel klikne na File -> Open Measurement
2	Systém	Aplikace zobrazí vyskakovací okno se seznamem validních měření.
3	Uživatel	Uživatel zvolí měrení a následně kline na OK
4	Systém	Po úspěšném zvolení měření aplikace nastaví zvolené měření jako aktuální a zobrazí užavateli dostupné procedury pro toto měření.

Tabulka 3 – Scénář popisující otevření nastavení

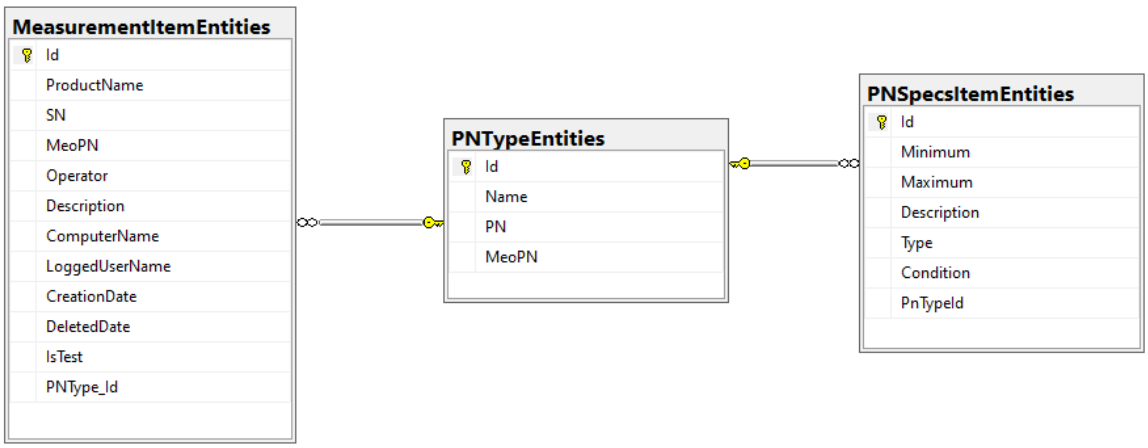
Název: Otevření nastavení		
ID: UC005		
Charakteristika: Případ popisující otevření nastavení		
Primární aktér: Uživatel		
Vedlejší aktéři: Žádní		
Vstupní podmínky: Uživatel má spuštěnou aplikaci.		
Výstupní podmínky: V aplikaci se změní aktuállní měření na právě zvolené.		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Uživatel	Uživatel klikne na Settings
2	Systém	Aplikace zobrazí vyskakovací okno se záložkami pro jednotlivé skupiny nastavení.

Tabulka 4 – Scénář pro připojení kamery

Název: Připojení kamery		
ID: UC005		
Charakteristika: Případ popisující postup pro připojení kamery		
Primární aktér: Uživatel		
Vedlejší aktéři: Žádní		
Vstupní podmínky: Uživatel má spuštěnou aplikaci a kameru připojenou k PC.		
Výstupní podmínky: Připojá kamera k aplikaci.		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Administrátor	Uživatel klikne na File -> Change image source
2	Systém	Aplikace zobrazí vyskakovací okno se seznamem dostupných kamer.
3	Uživatel	Uživatel zvolí kameru a následně kline na OK
4	Systém	Po úspěšném zvolení kamery aplikace připojí zvolenou kameru.

8 DATABÁZE

V tomto projektu byla vytvořena databáze pro ukládání dat ze systému. Databáze byla vytvořena pomocí Entity Frameworku metodou Code-First. To znamená, že nejprve vytvoříme třídy v naší aplikaci, a na základě těchto tříd Entity Framework automaticky vytvoří odpovídající databázové tabulky. Následující odstavce se zabývají detailním popisem této databáze a jednotlivými entitami v ní obsaženými.



Obrázek 5 – Diagram databáze

8.1 MeasurementItemEntity

Je nejpodstatnější entita v databázi, jedná se o entitu měření do které se ukládají záznamy o jednotlivých měřeních, kde každý záznam má následující parametry: Id: Jedná se o unikátní identifikátor záznamu měření. Slouží k jednoznačné identifikaci každého záznamu v databázi.

Name: Název měření, který může poskytnout užitečný popis nebo identifikaci konkrétního typu měření.

SerialNumber: Seriové číslo příslušného zařízení, ke kterému je měření prováděno. Pomáhá identifikovat zařízení spojené s daným měřením.

Vazba na PNType entitu: Odkazuje na typ měření (PNType), což může být například elektrický proud, napětí nebo jiná fyzikální veličina.

ComputerName: Název počítače nebo zařízení, na kterém bylo měření provedeno. Pomáhá sledovat, kde bylo měření provedeno.

Description: Popis měření, který může obsahovat další podrobnosti nebo informace o kontextu měření.

UserName: Uživatelské jméno osoby, která provedla měření. Poskytuje informaci o tom, kdo je zodpovědný za dané měření.

CreationDate: Datum a čas, kdy byl záznam měření vytvořen. Pomáhá sledovat časové razítko měření.

DeletedDate: Datum a čas, kdy byl záznam měření odstraněn.

IsTest: Boolovská hodnota, která označuje, zda je měření pouze testovací nebo má reálný význam. Užitečné pro filtrování dat.

Výsledky jednotlivých měření: Skutečné hodnoty nebo výsledky samotného měření, které jsou ukládány alespoň jednou spojené s každým záznamem měření.

8.2 PNTypeEntity

Entita specifikující typ měření. Obsahuje informace o typu měření, má vazby na konkrétní specifikace.

8.3 PNSpecItemEntity

Entita do které se ukládají specifikace pro konkrétní PN typy, má následující parametry:

Id: Unikátní identifikátor záznamu specifikace pro konkrétní PN typ. Umožňuje jednoznačně identifikovat každou specifikaci v databázi.

Minimum: Minimální hodnota, kterou může výsledek měření daného PN typu nabývat a stále splňovat specifikaci.

Maximum: Maximální hodnota, kterou může výsledek měření daného PN typu nabývat a stále splňovat specifikaci.

Vazba na PNType entitu: Odkazuje na konkrétní typ PN (Product Number), pro který jsou specifikace definovány. Toto spojení umožňuje aplikovat specifikace na správné PN typy.

Popis: Popis specifikace, který může obsahovat další informace o tom, co specifikace zahrnuje nebo jaká je její povaha.

Podmínka: Určuje, jakým způsobem je specifikace vyhodnocována vzhledem k výsledkům měření. Například může určovat, že výsledek měření musí být větší než Minimum, menší než Maximum, mezi Min a Max atd. Tato podmínka definuje kritéria, která musí být splněna, aby byla specifikace považována za dodrženou.

9 TVORBA APLIKACE

Součástí této diplomové práce je desktopová aplikace vytvořená pomocí frameworku WPF, jejímž hlavním úkolem je prezentace možností využití .NET frameworků pro zpracování obrazu a komunikaci s hardwarovými zařízeními v průmyslových aplikacích, aplikace dokáže komunikovat s kamerami, zpracovávat obraz, detekce spotů v obraze, vyhodnocování informací z obrazových dat, komunikace s hardwarem, komunikace s databází, ukládání a načítání dat.

Aplikace obsahuje procedury, které jsou uživateli zpřístupněny po vytvoření nového měření (nebo otevření již existujícího). Jednotlivé procedury jsou určeny pro získávání dat z hardwaru a pro vyhodnocování těchto dat. Aplikace umožňuje uživateli připojit a ovládat hardware v podobě laseru, power meteru a kamery. Aplikace také umožňuje uživateli ukládat naměřená data do databáze a načítat je. Aplikace také umožňuje uživateli použít namísto kamery obrázek uložený v počítači a vyhodnocovat data na tomto obrázku.

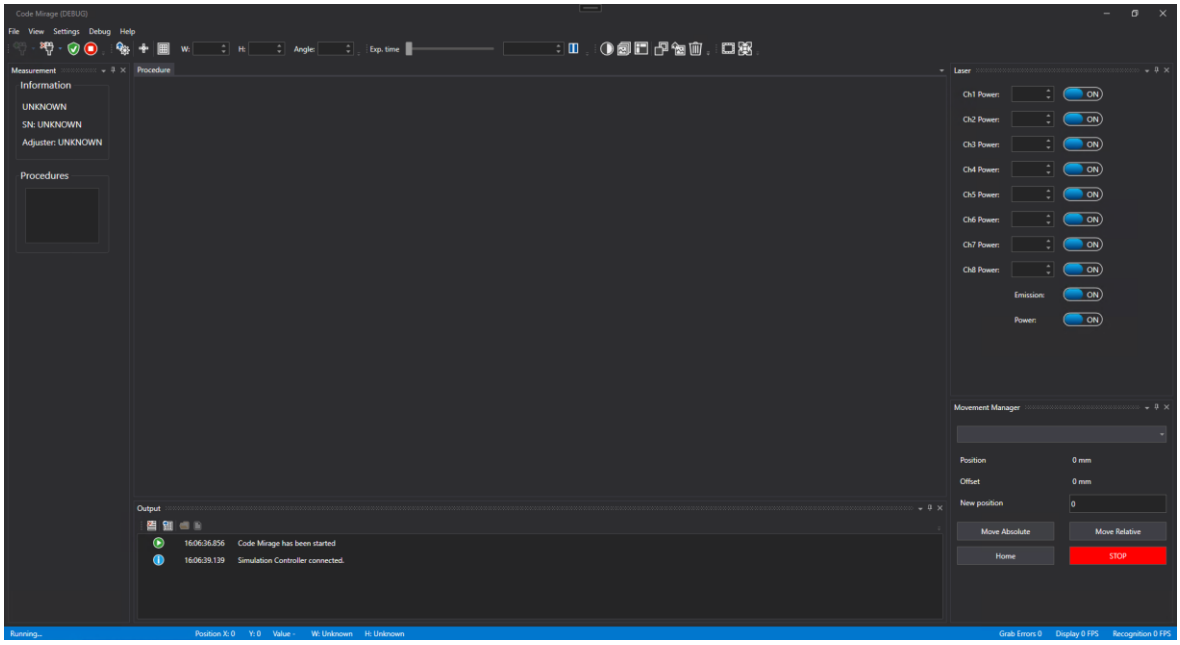
9.1 Uživatelské rozhraní

Po spuštění aplikace se uživateli zobrazí úvodní obrazovka, na této obrazovce uživatel může vidět:

Toolbar pomocí kterého může uživatel přistupovat k oknům pro tvorbu nových měření, otevírání již existujících měření, nastavení aplikace nebo připojování hardwaru.

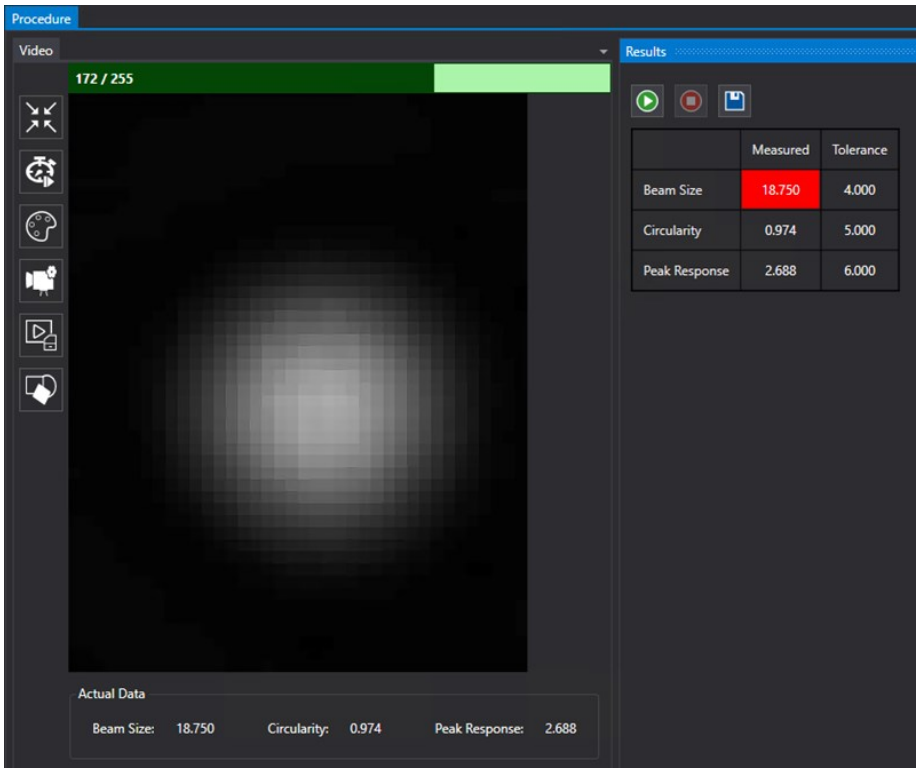
V levé části je okno měření, které vypisuje informace o aktuálním zvoleném měření a nabídku procedur dostupných pro aktuální měření, nabídka procedur se aktualizuje po vytvoření nového měření nebo po otevření již existujícího a liší se v závislosti na PNTypu tohoto měření.

V Pravé části aplikace jsou vidět okna sloužící pro ovládání hardwaru (pokud je připojen).



Obrázek 6 – Uživatelské rozhraní

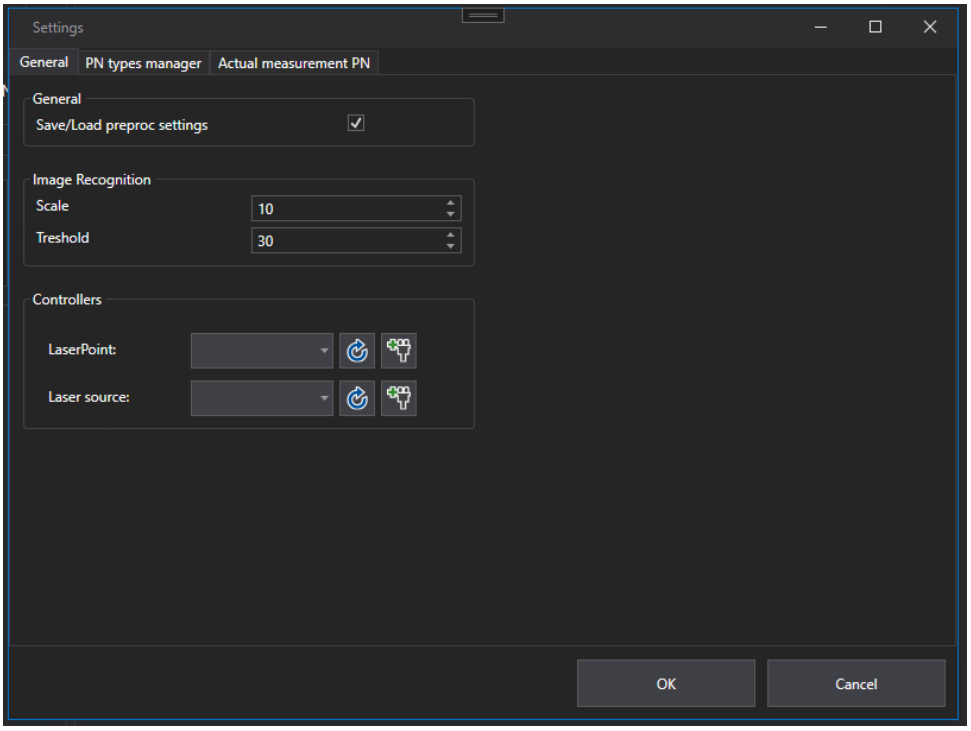
Po vytvoření nebo otevření měření se uživateli zobrazí dostupné procedury, po kliknutí na jednu z procedur aplikace zobrazí okno procedury, které se může skládat z video okna, okna s výsledky a tlačítek pro spuštění a zastavení měření nebo uložení naměřených dat.



Obrázek 7 – Vzhled okna procedury

9.1.1 Okno nastavení

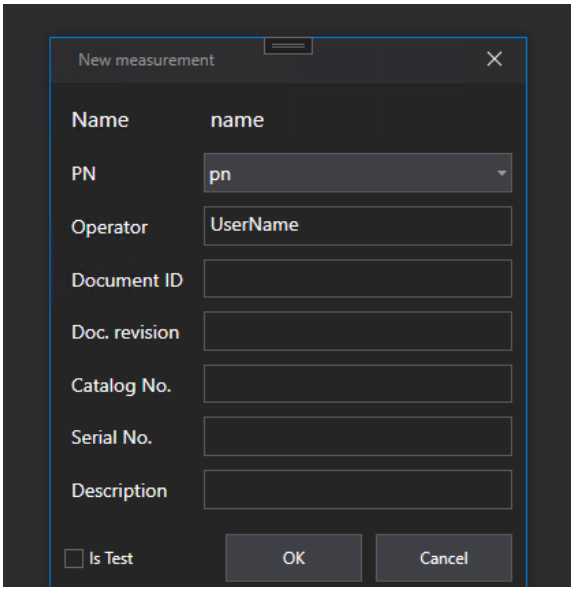
Pomocí okna nastavení může uživatel připojovat/odpojovat hardwarová zařízení. V záložce PNType může uživatel také vytvářet nové Pntypy nebo editovat stávající.



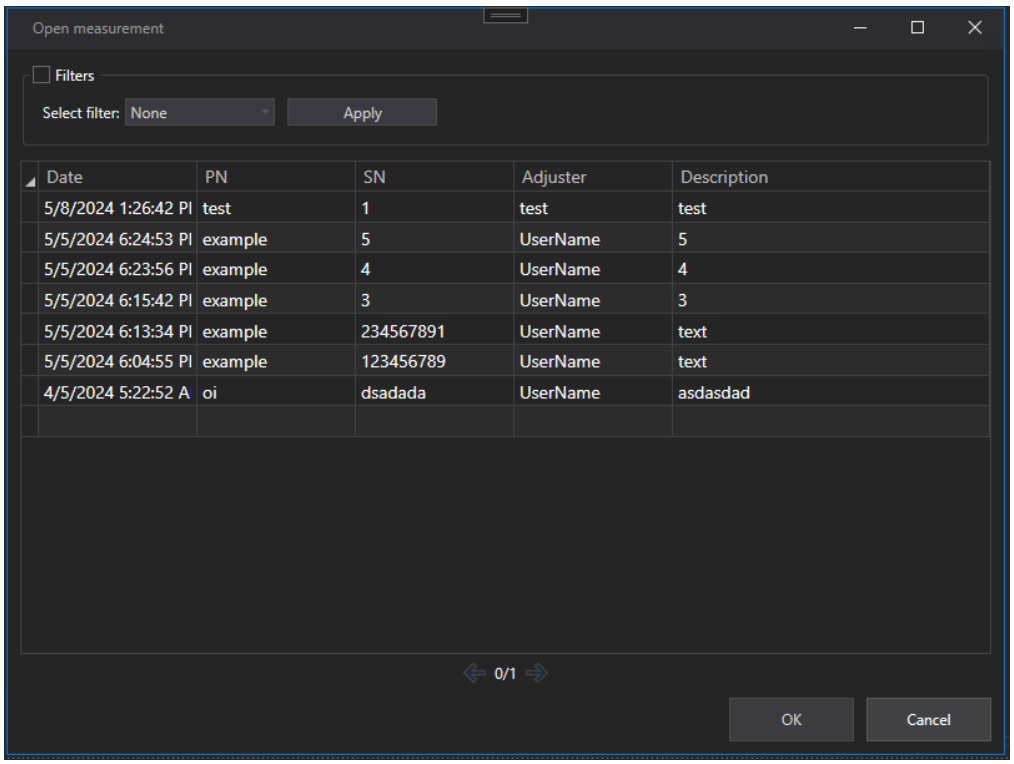
Obrázek 8 – Okno nastavení

9.1.2 Okna pro vytvoření a otevření měření

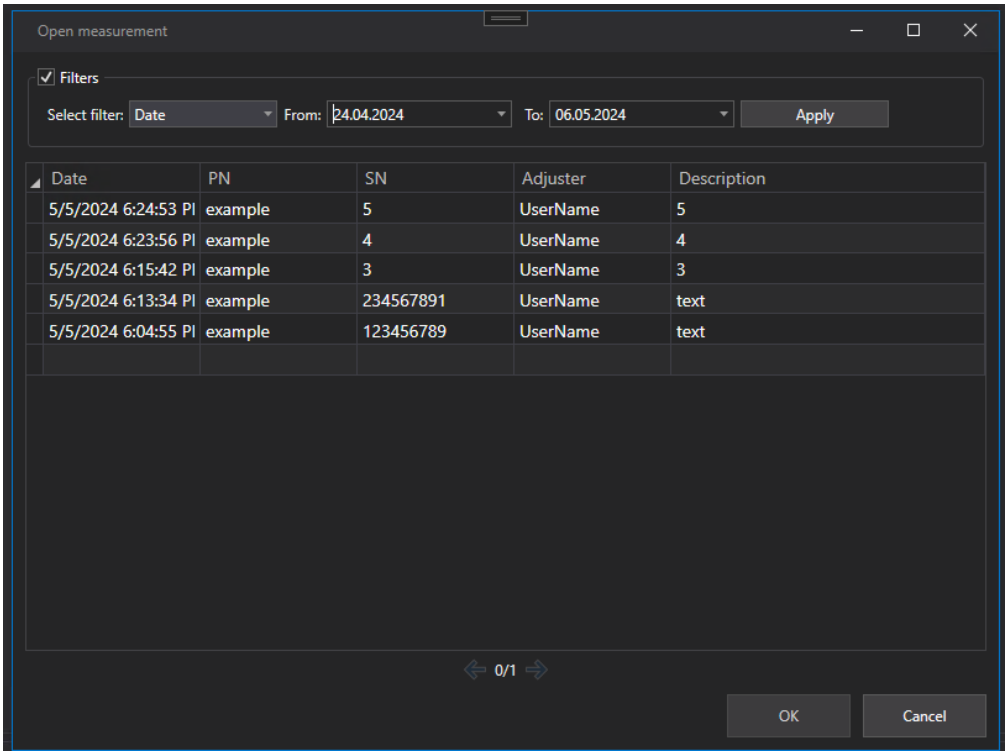
Pomocí těchto oken může uživatel vytvářet nová měření nebo otevírat již existující.



Obrázek 9 – Onko pro vytvoření nového měření



Obrázek 10 – Okno pro otevření již existujícího měření



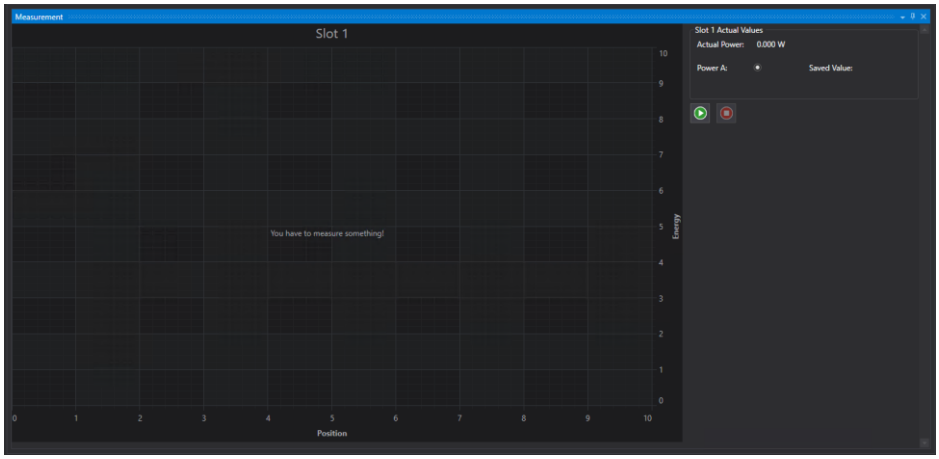
Obrázek 11 – Ukázka možnosti filtrování záznamů

9.2 Měřící procedury

Jednou z nejdůležitějších částí aplikace jsou měřící procedy, to jsou třídy jejichž úkolem je vyhodnocování získaných dat, ať už se jedná o data obrazová nebo o data vyčtená z hardwaru. Aplikace obsahuje tři procedury, které jsou popsány v následujících odstavcích.

9.2.1 IRTMeasurement Procedura

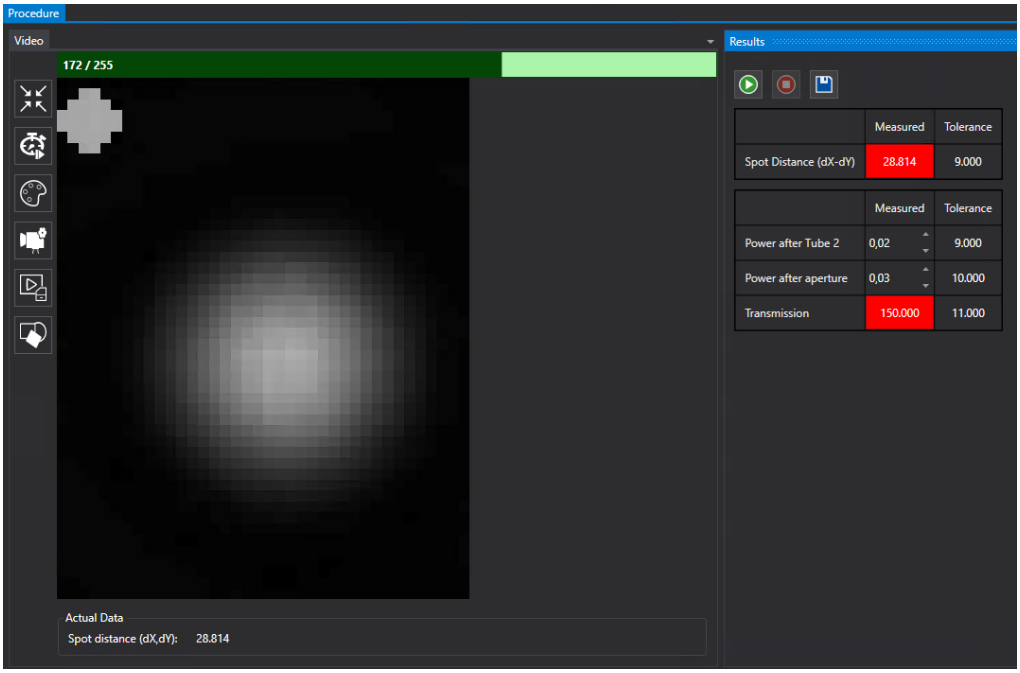
Měřící procedura určená pro měření stability výkonu laseru v čase. Procedura po spuštění měří pomocí power meteru výkon laseru po určitou dobu v určitých intervalech. Při každém změřeném bodu je aktuální hodnota vykreslena do grafu. Pro tvorbu grafů bylo využito knihovny SciChart. Po uplynutí času pro měření je vypočítaná průměrná hodnota výkonu laseru a je porovnána s očekávanou hodnotou.



Obrázek 12 – Okno IRT procedury

9.2.2 Procedura pro měření vzdálenosti spotů

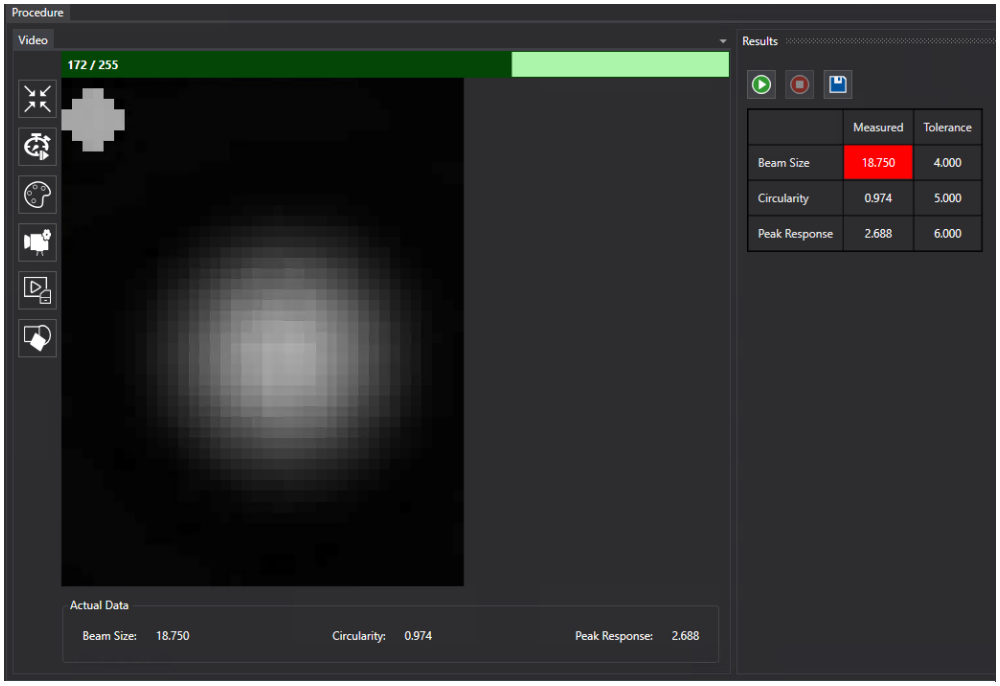
Procedura která najde dva spoty v obraze a vypočítá jejich vzájemnou vzdálenost, spuštění procesu provede to že se výsledná vzdálenost zapíše do tabulky výsledků a porovná se specifikací, pokud je vzdálenost větší než specifikace, políčko s výsledkem se podbarví červeně.



Obrázek 13 – Okno procedury pro vyhodnocení vzdálenosti spotů

9.2.3 Procedura pro měření vlastností spotu

Procedura která najde největší spot v obraze a vypočítá jeho vlastnosti jako jsou velikost, která určuje jak velký daný spot je, nebo centricita, která udává jak moc je spot kruhový. Spuštěním procesu se naměřené hodnoty vypíší do tabulky a porovnají se se zadanými specifikacemi.



Obrázek 14 – Procedura pro měření vlastností spotu

9.3 Detekce objektů v obraze

Aplikace dokáže detekovat objekty v obraze, využívá k tomu metody TwoLoopDetection, která jak název napovídá pracuje ve dvou smyčkách, první smyčka prochází obraz řádek po řádku a druhá sloupec po sloupci, přičemž se hledají pixely které splňují stanovené požadavky (například intenzitu světla dopadajícího na pixel), následně algoritmus projde všechny pixely splňující požadavky a vedle sebe ležící pixely spojí do takzvaných blobů se kterými se pracuje v dalších algoritmech.

Třída TwoLoopBlobDetection implementuje rozhraní IBlobDetection. Tato třída slouží k detekci "blobů" v obraze. Blob je souvislá oblast v obraze, která má stejnou vlastnost nebo charakteristiku (například barvu).

Metoda Execute provede detekci blobů v obraze. Má dva přetížené tvary. První forma metody Execute inicializuje interní pole _board na null a volá druhou formu metody Execute, která přebírá dodatečný parametr _board jako referenci. Druhá forma metody Execute provádí vlastní detekci blobů v obraze.

Během detekce se prochází každý pixel v obraze. Pokud hodnota pixelu překročí zadaný práh (tresh), provede se kontrola sousedních pixelů a případné spojení s existujícím blobem nebo vytvoření nového blobu. Na konci detekce jsou všechny nalezené bloby uloženy do seznamu a vráceny jako výstup metody.

Metoda GetNeighboringBlobLabels slouží k získání seznamu sousedních blobů pro daný pixel v obraze. Prochází okolní pixely a přidává je do seznamu, pokud jsou platné a mají nenulové označení v interním poli _board.

Celkově tato třída poskytuje implementaci algoritmu pro detekci blobů v obraze pomocí dvou smyček (loopů).

9.4 Hledání významých bodů

Pro vyhledávání významných bodů v obraze se používa třída `RowSignificantPointFinder` implementuje rozhraní `ISignificantPointFinder` a slouží k nalezení významných bodů v obrázku. Metoda `GetSignificantPoints` je klíčovou metodou této třídy, která provádí samotné vyhledávání těchto bodů.

Metoda přijímá několik parametrů:

- `inputImage` reprezentuje vstupní obrázek a poskytuje informace o jeho šířce, výšce a hodnotách pixelů.

- `thresh` je hodnota prahu, kterou pixel musí překročit, aby byl považován za významný.

- `area` definuje oblast v obrázku, ve které se mají významné body hledat.

- `token` je token zrušení, který umožňuje asynchronní zrušení operace.

Samotná implementace metody probíhá následovně:

1. Vytvoří se prázdný seznam `ret`, který bude obsahovat výsledné významné body.

2. Vytvoří se buffer `buff` pro ukládání hodnot pixelů obrázku.

3. Prochází se každý řádek obrázku v dané oblasti.

4. V každém řádku se postupně procházejí pixely zleva doprava.

5. Pokud je hodnota pixelu vyšší než zadaný práh a zároveň není nalezen levý okraj významné oblasti, zaznamená se pozice jako levý okraj.

6. Jakmile je nalezen levý okraj a je nalezena hodnota pixelu vyšší než práh, zaznamená se pozice jako pravý okraj.

7. Pokud jsou nalezeny oba okraje, jsou oba zaznamenány jako významné body a přidány do seznamu `ret`.

8. Pokud je vyvolána žádost o zrušení operace pomocí tokenu `token`, vyvolá se výjimka `OperationCanceledException`.

9. Po dokončení procházení všech řádků je seznam `ret` vrácen jako výsledek metody.

Celkově tato metoda provádí jednoduchý průchod obrázkem po řádcích a hledá významné body, kde hodnota pixelu překročí zadaný práh. Výsledné body jsou pak vráceny jako seznam `List<Point>`.

```
    public List<Point> GetSignificantPoints(IImageInfo inputImage, double thresh, Rect area, CancellationToken
token)
    {
      var ret = new List<Point>();
      var buff = new ushort[inputImage.ImageBuffer.Length];
      for (int y = (int)area.Top; y < area.Bottom; y++)
      {
        bool left = false;
```

```
            var leftInd = -1;
            var rightInd = -1;
            token.ThrowIfCancellationRequested();

            for (int x = (int)area.Left; x < area.Right; x++)
            {
                var ind = x + y * (int)inputImage.Width;
                if (inputImage.ImageBuffer[ind] > thresh && !left)
                {
                    left = true;
                    leftInd = x;
                }
                else if (left && inputImage.ImageBuffer[ind] > thresh)
                    rightInd = x;
            }
            if (left && rightInd != -1)
            {
                ret.Add(new Point(leftInd, y));
                ret.Add(new Point(rightInd, y));
            }
        }
        return ret;
    }
```

Kód 1 – Hledání významných bodů

Paralelní smyčka Parallel.For prochází indexy 0 a 1, což odpovídá směrům: 0 pro řádky a 1 pro sloupce. V každé iteraci smyčky se vytváří nová instance buď třídy RowSignificantPointFinder nebo ColumnSignificantPointFinder podle hodnoty indexu j. Poté jsou volány metody GetSignificantPoints těchto instancí s odpovídajícími parametry.

Výsledky hledání významných bodů pro každý směr jsou ukládány do ConcurrentDictionary. Po dokončení paralelní smyčky jsou všechny nalezené body pro řádky přidány do seznamu ret.

Nakonec jsou projity všechny nalezené body pro sloupce a přidány do seznamu ret, pokud se ještě nevyskytly v seznamu. Tím se zajistí, že výsledný seznam obsahuje unikátní body.

Celkově tato implementace umožňuje paralelní zpracování hledání významných bodů v obrázku podle řádků a sloupců, což může výrazně zvýšit výkon při zpracování velkých obrázků.

```
    public List<Point> GetSignificantPoints(IImageInfo inputImage, double thresh, Rect area, CancellationToken
token)
    {
        var ret = new List<Point>();
        var cD = new ConcurrentDictionary<int, List<Point>>();
        Parallel.For(0, 2, j =>
```

```
        {
            cD[j] = (j == 0) ? new RowSignificantPointFinder().GetSignificantPoints(inputImage, thresh, area, token) :
new ColumnSignificantPointFinder().GetSignificantPoints(inputImage, thresh, area, token);
        });


        ret.AddRange(cD[0]);
        foreach (var item in cD[1])
            if (!ret.Any(x => x.X == item.X && x.Y == item.Y))
                ret.Add(item);
        return ret;
    }
```

Kód 2 – Hledání významných bodů

9.5 Fitování kružnicí

Pro fitování kružnicí se využívá metoda Compute v třídě HalirFlusserElipticFit, která je zodpovědná za výpočet eliptické křivky na základě zadaných bodů xyCoordinates. Nejprve je vytvořena matice points, ve které jsou uloženy souřadnice jednotlivých bodů. Poté jsou vytvořeny další dvě matice, D1 a D2, které obsahují základní výrazy pro výpočet eliptické křivky. Následně jsou vypočteny tři matice S1, S2 a S3 na základě výrazů z předchozích matic. Tyto matice jsou důležité pro následující výpočty. Poté je vypočtena matice T a následně matice N, která je kombinací předchozích dvou matic.

Poté následuje výpočet vlastního rozkladu matice M, která je použita k určení parametrů eliptické křivky. Mezi tyto parametry patří poloměry, úhel náklonu a střed křivky. Tyto parametry jsou pak použity k vytvoření instance třídy Ellipse, která reprezentuje eliptickou křivku.

Nakonec je tato instance vrácena jako výsledek metody Compute. Celkově tato metoda provádí sofistikované výpočty na základě poskytnutých bodů s cílem nalézt nejlepší eliptickou křivku pro danou sadu bodů.

9.6 Výpočet velikosti nalezeného spotu

Pro výpočet se používá třída BeamSizeCalculator, která obsahuje metody pro výpočet velikosti světelného záření (či jiného druhu signálu) na základě poskytnutého obrazového profilu.

Třída BeamSizeCalculator poskytuje sadu metod pro analýzu obrazových profilů a výpočet velikosti světelného záření na základě poskytnutých dat. Tato třída je užitečná pro aplikace, které vyžadují detailní analýzu intenzitních profilů v různých směrech.

Metoda FindMaximumInProfile slouží k nalezení maximální hodnoty v daném obrazovém profilu. Průchodem profilu v zadaném směru a zpětným průchodem od středu profilu se zjišťuje nejvyšší intenzita signálu.

Metoda FindSizeInProfile analyzuje daný obrazový profil a hledá velikost signálu na základě zadaného prahu intenzity. Tato metoda identifikuje oblasti signálu, které překračují zadaný práh, a určuje jejich velikost.

Metoda SpotSize je hlavním výpočtovým nástrojem třídy. Provádí analýzu obrazových profilů ve směrech x, y a pod úhly 45° a -45°. Pro každý směr se zjišťuje maximální intenzita signálu a na základě ní se určuje velikost signálu při daném poměru k maximu. Následně jsou tyto velikosti průměrovány a vrací se jako výsledek.

Třída BeamSizeCalculator poskytuje flexibilní a robustní nástroje pro analýzu obrazových dat a měření velikosti světelných záření v různých směrech. Je vhodná pro širokou škálu aplikací od zpracování obrazu po vědecký výzkum.

```
public static class BeamSizeCalculator
{
   private static ushort FindMaximumInProfile(IImageInfo info, int centreX, int centreY, (int, int) dir)
   {
      var (dirX, dirY) = dir;
      var posX = centreX + dirX;
      var posY = centreY + dirY;
      var maximum = info.ImageBuffer[centreX + (int)info.Width * centreY];

      while (true)
      {
         if (posX < 0 || posX >= info.Width || posY < 0 || posY >= info.Height)
            break;

         var intensity = info.ImageBuffer[posX + (int)info.Width * posY];
         if (intensity > maximum)
            maximum = intensity;

         posX += dirX;
         posY += dirY;
      }

      var negX = centreX - dirX;
      var negY = centreY - dirY;
      while (true)
      {
```

```
        if (negX < 0 || negX >= info.Width || negY < 0 || negY >= info.Height)
            break;

        var intensity = info.ImageBuffer[negX + (int)info.Width * negY];
        if (intensity > maximum)
            maximum = intensity;

        negX -= dirX;
        negY -= dirY;
    }

    return maximum;
}

private static double? FindSizeInProfile(IImageInfo info, int centreX, int centreY,
    (int, int) dir, ushort edgeIntensity)
{
    var (dirX, dirY) = dir;
    var posX = centreX + dirX;
    var posY = centreY + dirY;

    var i = 0;
    while (true)
    {
        if (posX < 0 || posX >= info.Width || posY < 0 || posY >= info.Height)
            return null;

        if (info.ImageBuffer[posX + (int)info.Width * posY] < edgeIntensity)
            break;

        i++;
        posX += dirX;
        posY += dirY;
    }

    var negX = centreX - dirX;
    var negY = centreY - dirY;
    while (true)
    {
        if (negX < 0 || negX >= info.Width || negY < 0 || negY >= info.Height)
            return null;

        if (info.ImageBuffer[negX + (int)info.Width * negY] < edgeIntensity)
            break;

        i++;
        negX -= dirX;
        negY -= dirY;
    }

    return i + 1;
}

public static double? SpotSize(IImageInfo info, int centreX, int centreY,
    double edgeToMaximumRatio)
{
    var maximum = FindMaximumInProfile(info, centreX, centreY, (1, 0));
    var xProfile = FindSizeInProfile(info, centreX, centreY, (1, 0), (ushort)(edgeToMaximumRatio * maximum));
    if (!xProfile.HasValue)
        return null;
```

```
            maximum = FindMaximumInProfile(info, centreX, centreY, (0, 1));
            var yProfile = FindSizeInProfile(info, centreX, centreY, (0, 1), (ushort)(edgeToMaximumRatio * maximum));
            if (!yProfile.HasValue)
                return null;

            maximum = FindMaximumInProfile(info, centreX, centreY, (1, 1));
            var pos45Profile =
                FindSizeInProfile(info, centreX, centreY, (1, 1), (ushort)(edgeToMaximumRatio * maximum));
            if (!pos45Profile.HasValue)
                return null;

            maximum = FindMaximumInProfile(info, centreX, centreY, (1, -1));
            var neg45Profile =
                FindSizeInProfile(info, centreX, centreY, (1, -1), (ushort)(edgeToMaximumRatio * maximum));
            if (!neg45Profile.HasValue)
                return null;

            return (neg45Profile + pos45Profile + xProfile + yProfile) / 4.0;
        }
    }
```

Kód 3 – Výpočet velikosti nalezeného spotu

9.7 Výpočet vzdálenosti dvou spotů

Pro výpočet vzdálenosti dvou spotů se využívá třída SpotDistanceCalculator, která slouží k výpočtu vzdálenosti mezi dvěma nalezenými body v obraze. Při inicializaci třídy je vytvořena instance třídy TwoLoopBlobDetection, která se používá pro hledání spotů. Metoda GetSpotDistance přijímá informace o obraze (image) a CancellationToken, který umožňuje zrušení operace v případě potřeby.

Nejprve se zkontroluje, zda nebyl token pro zrušení operace žádán, a poté se provádí analýza blobů na zadaném obraze. Pokud je detekováno méně než dva bloby, funkce vrátí nulovou vzdálenost, protože není možné vypočítat vzdálenost mezi méně než dvěma body.

Pokud jsou nalezeny alespoň dva bloby, jsou seřazeny sestupně podle jejich počtu pixelů. Poté jsou určeny středy těchto dvou největších blobů (A a B) a vypočítána vzdálenost mezi nimi pomocí euklidovské vzdálenosti. Tato vzdálenost je následně vrácena jako výstup metody.

```
    public class SpotDistanceCalculator : ISpotDistanceCalculator
    {
        private readonly TwoLoopBlobDetection blobAnalysis = new TwoLoopBlobDetection();

        public double GetSpotDistance(IImageInfo image, CancellationToken token)
```

```
        {
            token.ThrowIfCancellationRequested();

            var blobImg = image;
            ushort[,] map = null;
            var localThresh = 40;
            var blobs = this.blobAnalysis.Execute(blobImg, (int)localThresh, ref map);
            if (blobs.Count < 2)
                return 0;

            blobs.OrderByDescending(x => x.PixelCount);

            var A = blobs[0].Center;
            var B = blobs[1].Center;

            return Math.Sqrt(Math.Pow(A.X - B.X, 2) + Math.Pow(A.Y - B.Y, 2));
        }
    }
```

Kód 4 – Výpočet vzdálenosti dvou spotů

9.8 Výpočet těžiště spotu

Třída BaseCoGCalculator slouží k výpočtu středu těžiště (centroidu) a velikosti oblasti zájmu na obraze na základě zadaného prahu (threshold). Tato třída není určena k přímému použití, ale spíše jako základní implementace pro konkrétní výpočty těžiště a velikosti oblasti.

Metoda GetCog (centroid) provádí výpočet středu těžiště. Prochází každý pixel v obraze a pokud jeho hodnota překračuje zadaný práh, přidává hodnoty váženého x-ového a y-ového souřadnice do celkových součtů sumX a sumY odpovídajících vážených hodnot sumMi. Po projití všech pixelů je vypočtena průměrná hodnota těžiště na základě součtů sumX a sumY dělených součtem sumMi.

Metoda GetSize (velikost) určuje velikost oblasti zájmu kolem daného bodu. Nejprve se určí maximální hodnota pixelu v řádku a sloupci, a na základě tohoto maxima se určí nový práh pro segmentaci oblasti zájmu. Následně jsou nalezeny hranice oblasti zájmu ve směrech x a y pro zadaný střed a práh. Tyto hranice jsou použity k vytvoření obdélníkové oblasti zájmu.

Obě metody pracují s parametry vstupního obraze, středu bodu a prahu. Výstupem metody GetCog je instance třídy Point, zatímco metoda GetSize vrací oblast zájmu ve formě instance třídy Rect.

```
public class BaseCoGCalculator
   {
       protected Point GetCog(IImageInfo image, double treshold)
       {
           double sumMi = 0;
           double sumX = 0;
           double sumY = 0;
           for (int i = 0; i < image.ImageBuffer.Length; i++)
           {
               if (image.ImageBuffer[i] >= treshold)
               {
                   int x = i % (int)image.Width;
                   int y = i / (int)image.Width;
                   sumMi += image.ImageBuffer[x + (int)image.Width * y];
                   sumX += image.ImageBuffer[x + (int)image.Width * y] * x;
                   sumY += image.ImageBuffer[x + (int)image.Width * y] * y;
               }
           }
           if (sumMi > 0) return new Point((sumX / sumMi), (sumY / sumMi));
           else throw new Exception("Spot not found");
       }
       public Rect GetSize(IImageInfo image, Point center, double treshold)
       {
           int max = 0;
           for (int i = 0; i < image.Width; i++)
               max = max < image.ImageBuffer[i + (int)image.Width * (int)center.Y] ? im
age.ImageBuffer[i + (int)image.Width * (int)center.Y] : max;
           var tresh = ((max - treshold) / 10) + treshold;
           int x = 0;
           int y = (int)center.Y;
           while (x < image.Width && image.ImageBuffer[x + (int)image.Width * y] < tres
h) x++;
           int Start = x;
           x = (int)image.Width;
           y = (int)center.Y;
           while (x > 0 && image.ImageBuffer[x + (int)image.Width * y] < tresh) x--;
           int End = x;
           max = 0;
           for (int i = 0; i < image.Height; i++)
               max = max < image.ImageBuffer[(int)center.X + (int)image.Width * i] ? im
age.ImageBuffer[(int)center.X + (int)image.Width * i] : max;
           tresh = ((max - treshold) / 10) + treshold;
           x = (int)center.X;
           y = (int)0;
           while (y < image.Height && image.ImageBuffer[x + (int)image.Width * y] < tre
sh) y++;
           int StartY = y;
           x = (int)center.X;
           y = (int)image.Height - 1;
           while (y > 0 && image.ImageBuffer[x + (int)image.Width * y] < tresh) y--;
           int EndY = y;
           return new Rect(new Point(Start, StartY), new Point(End, EndY));
       }
   }
```

Kód 5 – Výpočet těžiště spotu

9.9 Komunikace s laserem

Třída HeraController je odvozena od třídy BaseSerialController, která slouží k ovládání zařízení přes sériovou komunikaci (RS-232).

Zde je popis jednotlivých metod a vlastností této třídy:

Konstruktor inicializuje výchozí hodnoty portu (ComPort) a baud rate.

Metoda SetActionGroup() pošle příkaz pro nastavení akční skupiny.

Metody TurnOnPower() a TurnOffPower() zapínají a vypínají napájení.

Metoda GetPowerState() získává stav napájení a vrací ho jako stav výčtu (PowerState).

Metoda TurnOffAllChannels() vypíná všechny kanály.

Metody TurnOnEmmission() a TurnOffEmmission() zapínají a vypínají emisi.

Metody SetChannelRatedPulsePower(), SetChannelMaxPulsePower() a SetChannelMinPulsePower() nastavují hodnoty pulzního výkonu pro různé kanály.

Privátní metoda Set() používá sériovou komunikaci k odeslání příkazu s možností předání až dvou parametrů.

Privátní metoda Get() používá sériovou komunikaci k získání odpovědi na zadaný příkaz s možností předání až dvou parametrů.

Metoda SendPeriodicalQuery() je označena jako protected a vyvolává výjimku NotImplementedException, což naznačuje, že je třeba implementovat periodické dotazy, ale zatím není implementována.

Tato třída poskytuje rozhraní pro ovládání nějakého zařízení, které komunikuje pomocí sériového portu. Metody umožňují různé operace s tímto zařízením, jako je zapínání/vypínání, nastavování hodnot a zjišťování stavů.

```
public class HeraController : BaseSerialController
  {
      public HeraController()
      {
          ComPort = "COM12";
          BaudRate = 57600;
      }


      public void SetActionGroup()
```

```
        {
            Set("AB");
        }


        public void SetChannelRatedPulsePower(ChannelType channel, int numberOfParameter
Set, int value)
        {
            switch (channel)
            {
                case ChannelType.Channel1:
                    Set("FA", numberOfParameterSet.ToString(), value.ToString());
                    break;

. . .
                case ChannelType.Channel8:
                    Set("FV", numberOfParameterSet.ToString(), value.ToString());
                    break;
                default:
                    break;
            }
        }


        public void SetChannelMaxPulsePower(ChannelType channel, int numberOfParameterSe
t, int value)
        {
            switch (channel)
            {
                case ChannelType.Channel1:
                    Set("FB", numberOfParameterSet.ToString(), value.ToString());
                    break;

                  . . .

                case ChannelType.Channel8:
                    Set("FW", numberOfParameterSet.ToString(), value.ToString());
                    break;
                default:
                    break;
            }
        }
        private void Set(string command, string param1 = "", string param2 = "")
        {
            if (serialCommunication.CommunicationPort.IsOpen)
            {
                if (!serialCommunication.CommunicationPort.SendPacket($"?{command}{param
1}{param2}\r\n"))
                    throw new MeoException("Device failed to send response");
            }
            else
            {
                throw new MeoException("Serial port is not open.");
            }
        }
        private string[] Get(string command, string param1 = "", string param2 = "")
        {
            if (serialCommunication.CommunicationPort.IsOpen)
```

```
            {
                if (!serialCommunication.CommunicationPort.SendPacketAck($"?{command}{pa
ram1.PadLeft(2,'0')}{param2.PadLeft(9, '0')}\r\n", "\r\n", out string resp))
                    throw new MeoException("Device failed to send response");

                resp = resp.Replace(":", "").RemoveLineEndings();

                string[] splitStrings = new string[2];
                splitStrings[0] = resp.Substring(0, 2);
                splitStrings[1] = resp.Substring(2, 8);
                return splitStrings;
            }
            else
            {
                throw new MeoException("Serial port is not open.");
            }
        }
    }
```

Kód 6 – Komunikace s laserem

9.10 Komunikace s power meterem

Třída A200D60SController je implementována jako odvozená třída jak z třídy BaseSerialController, tak z rozhraní IA200D60SSlotController. Stejně jako HeraController i tato třída slouží k ovládání zařízení přes sériovou komunikaci (RS-232).

9.11 Komunikace s kamerami Basler

Komunikace s kamerami od výrobce Basler je realizována pomocí SDK pylon, které nabízí obecné, jednotné programovací rozhraní, které skrývá většinu rozdílů mezi technologiemi rozhraní. Rozhraní API umožňuje snadnou migraci z jednoho rozhraní kamery na jiné bez větších změn kódu.

Protože většina objektů pylonu alokuje nespravované systémové prostředky, je důležité vědět, jak zacházet s objekty, které je třeba zlikvidovat. Nepředpokládá se, že by objekty pylonů byly automaticky uklízeny garbage collectorem. Proto objekty pylonů implementují rozhraní System.IDisposable. Instanci objektu implementujícího IDisposable je třeba řádně zlikvidovat voláním Dispose() nebo pomocí příkazu "using". Případně můžete použít příkaz "try-finally". V tomto případě musí být volání Dispose() umístěno v bloku "finally".

Rozhraní IDisposable je implementováno také u výsledků grabování, aby bylo možné ovládat jejich odpovídající buffer. pylon používá pro grabování obrázků pool bufferů s pevným počtem bufferů (výchozí: 10). To je nutné, protože ovladač musí provést určité přípravy, např. uzamknout paměťové stránky bufferu, aby mohl grabovat obrázky do

bufferu. Použití fondu vyrovnávacích pamětí je tedy mnohem efektivnější než alokace nové vyrovnávací paměti pro každý grabovaný obrázek. Voláním funkce Dispose() signalizujete, že buffer může být znovu použit pro grabování jiného obrázku. Toto nemůže být implicitně řešeno garbage collector, protože nelze předvídat, kdy bude probíhat garbage collection.

```
camera.StreamGrabber.Start();
for (int i = 0; i< 10; ++i)
{
    IGrabResult grabResult = camera.StreamGrabber.RetrieveResult(5000,
TimeoutHandling.ThrowException);
    using (grabResult)
    {
        if (grabResult.GrabSucceeded)
        {
            Console.WriteLine( "SizeX: {0}", grabResult.Width );
            Console.WriteLine( "SizeY: {0}", grabResult.Height );
            byte[] buffer = grabResult.PixelData as byte[];
    Console.WriteLine( "Gray value of first pixel: {0}", buffer[0] );
            Console.WriteLine( "" );
        }
        else
{
    Console.WriteLine("Error: {0} {1}", grabResult.ErrorCode,
grabResult.ErrorDescription);
}
    }
}
camera.StreamGrabber.Stop();
```

Kód 7 – Sběr snímků

9.11.1 Třída BaseCamera

Třída `BaseCamera` představuje abstraktní základ pro zdroje obrazu kamer. Obsahuje různé konstanty pro identifikaci parametrů kamery a implementuje několik metod pro manipulaci s obrazovými daty a konfigurací kamery.

Konstruktor třídy inicializuje některé proměnné a načítá konfigurační XML soubor pro seznam kamer. Metoda `LoadParamsFromCameraList` načítá parametry kamery z konfiguračního souboru.

Třída poskytuje možnosti nastavení expozice (`ExposureTime`) a zesílení (`Gain`) kamery. Pokud není dostupný alias pro některý z těchto parametrů, třída použije výchozí hodnoty a zaznamená chybu.

Metoda `ReadCameraElement` vyhledá konfiguraci pro konkrétní kameru v XML souboru. Metoda `CheckConfigAvailable` ověřuje dostupnost konfigurace pro kameru a v případě nedostupnosti zaznamená chybu.

Další metody se zabývají manipulací s obrazovými daty, jako je změna velikosti obrazového rámečku (`ResetRectangle`) nebo nastavení formátu pixelů (`SetPixelFormat`). Tyto metody jsou implementovány v odvozených třídách.

Nakonec, třída obsahuje několik abstraktních metod, které musí být implementovány v odvozených třídách pro specifické typy kamer. Tyto metody zahrnují ukládání a načítání uživatelské konfigurace kamery a manipulaci s formátem pixelů.

Celkově tato třída poskytuje základní funkcionalitu pro práci s kamerami a manipulaci s jejich obrazovými daty.

9.11.2 Třída BaslerCamera

Třída BaslerCamera je součástí prostředí MeoComponents a slouží k integraci a řízení kamery od společnosti Basler. Tato třída dědí od třídy BaseCamera a implementuje rozhraní IDisposable, což umožňuje správu jejích prostředků.

Mezi hlavní vlastnosti a metody této třídy patří:

Inicializace a konstrukce: Konstruktor třídy přijímá informace o kameře jako parametr. Během inicializace se provádí otevření kamery, získání informací o ní a nastavení požadovaných parametrů.

Nastavení a získávání parametrů: Metody pro nastavení a získání různých parametrů kamery, jako je čas expozice, zisk, hloubka bitů, rozlišení a další.

Ovládání snímání a streamování obrazu: Metody pro zapnutí a vypnutí snímání obrazu, zachycení obrazu ze streamu a zpracování získaných dat.

Manipulace s formátem obrazu: Funkce pro nastavení a získání pixelového formátu, včetně podpory různých barevných hloubek.

Správa prostředků: Metody pro správu zdrojů, jako je otevření a uzavření kamery a uvolnění používaných prostředků.

Celkově tato třída poskytuje komplexní rozhraní pro práci s kamerou od společnosti Basler v rámci aplikace, včetně možností nastavení parametrů, snímání obrazu a správy zdrojů.

9.11.3 Třída ImageSourceCore

Třída ImageSourceCore představuje jádro systému zpracování obrazu a zajišťuje správu dostupných zdrojů obrazu včetně kamer a jiných zařízení. Zde je souhrn jejích hlavních vlastností a funkcí:

Singleton Design Pattern: Třída implementuje návrhový vzor Singleton, což znamená, že existuje pouze jedna instance této třídy během běhu aplikace. To je dosaženo pomocí statického vlastníka Instance a privátního konstruktoru.

Správa dostupných kamer: Třída udržuje kolekci dostupných kamer v ObservableCollection AvalaibleCameras, která se automaticky aktualizuje při změnách dostupných zařízení.

Podpora různých typů kamer: Kóduje podporu pro různé typy kamer, včetně Basler a Pleora kamer, pomocí konstantních řetězců pro identifikaci typů a verzí SDK.

Automatické načítání kamer: Třída automaticky zjišťuje přítomnost a typy dostupných kamer pomocí metody GetConnectedCameras(), která vrací seznam dostupných kamer spolu s jejich stavem.

Podpora rozšíření kamer: Umožňuje přidání dalších typů kamer pomocí metody AddExtensionCamera, která umožňuje dynamicky rozšiřovat funkcionalitu systému.

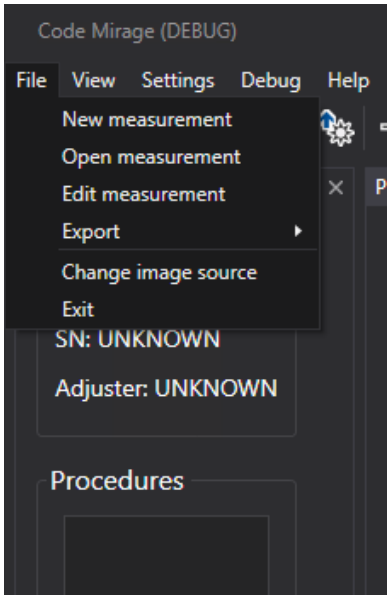
Asynchronní aktualizace kamer: Metoda ReloadCameras() provádí asynchronní aktualizaci dostupných kamer a umožňuje dynamické přidávání a odebírání kamer za běhu aplikace.

Správa životního cyklu objektů: Implementuje rozhraní IDisposable pro řízení uvolňování prostředků a řeší správu zdrojů v metodě Dispose(), což je důležité pro zajištění efektivního uvolňování paměti a zamezení úniků paměti.

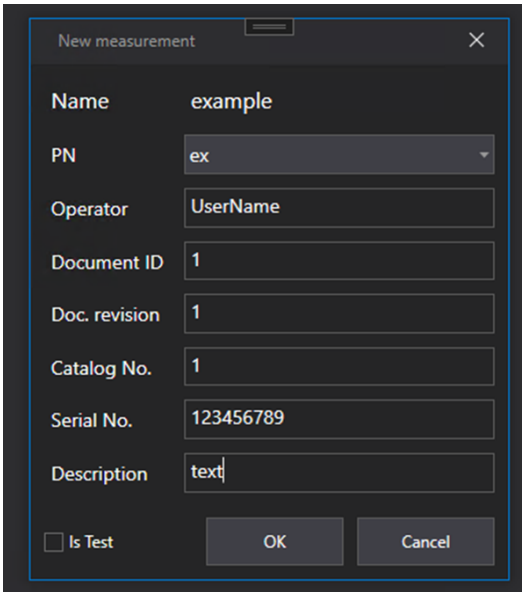
Tímto způsobem třída ImageSourceCore poskytuje robustní a flexibilní základ pro správu a manipulaci s různými typy zdrojů obrazu v aplikaci.

10 DEMONSTRACE POUŽITÍ APLIKACE

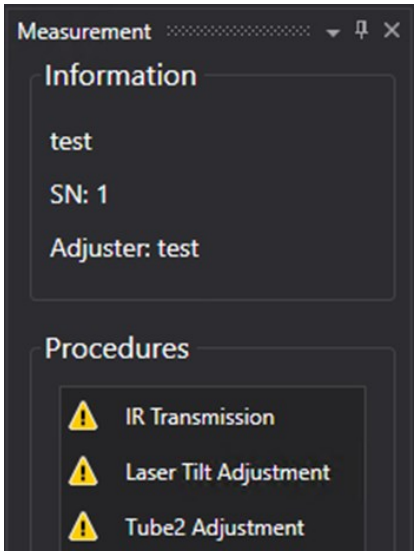
Aplikace vytvořená v rámci této diplomové práce, byla vytvořena za účelem demonstrace možnosti využití frameworku .NET pro zpracování obrazu a možnosti komunikace s hardwarovými zařízeními. V rámci tvorby této diplomové práce byla také vytvořena krátká videa, která jsou součástí příloh této práce, demonstrující funkcionalitu níže popsaných procedur. Jednotlivé procedury aplikace jsou v praxi použitelné pro měření kvality optických systémů. Proces měření kvality optických systémů začíná vytvořením nového



Obrázek 15 – Nabídka menu



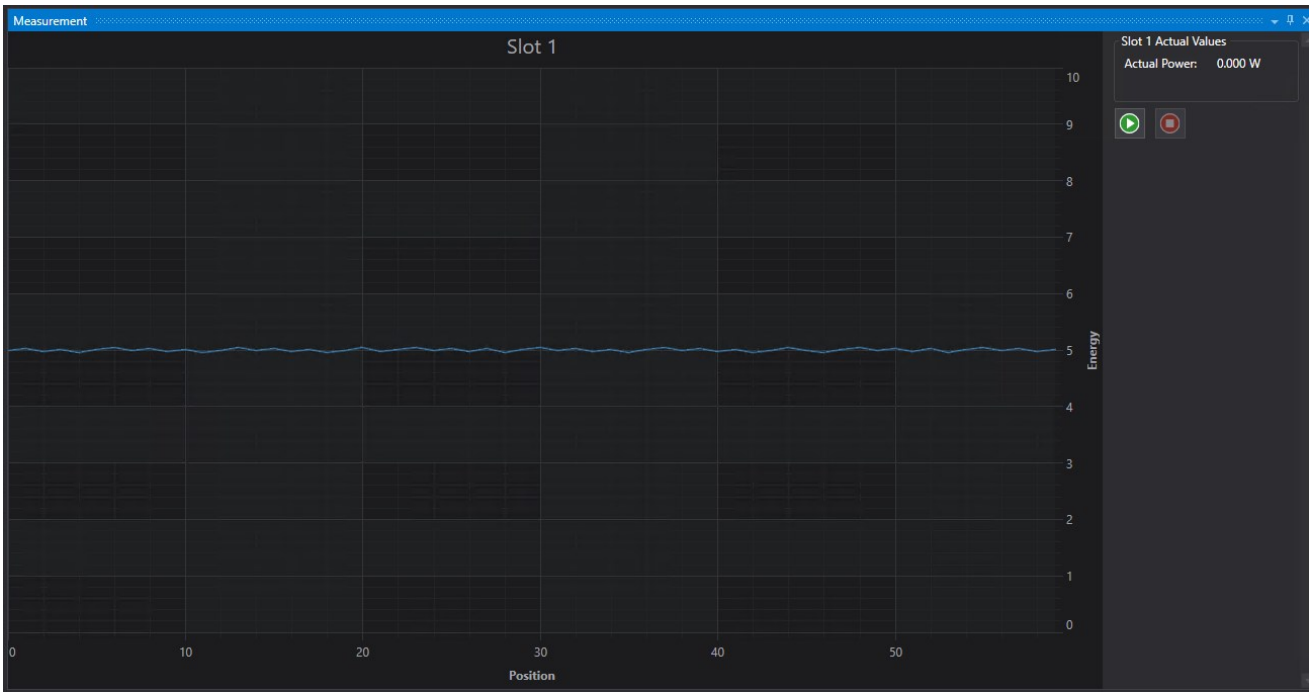
Obrázek 16 – Okno pro vytvoření nového měření



Obrázek 17 – Informace o zvoleném měření a nabídka procedur

měření v aplikaci. Tím uživatel získá přístup k procedurám, které jsou k dispozici pro dané měření.

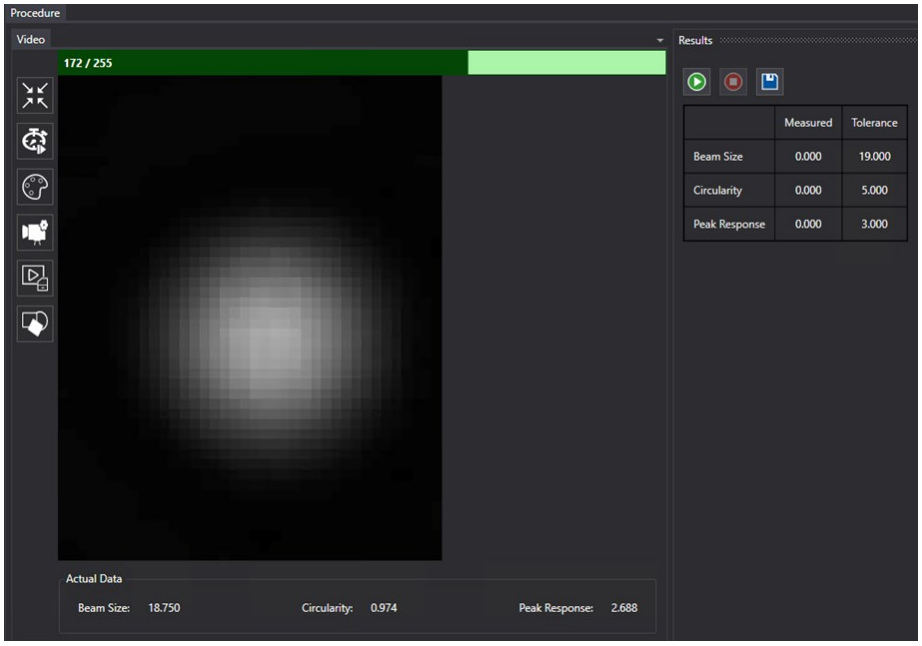
Jako první by měl uživatel spustit proceduru pro měření stability výkonu laseru (IRTMeasurement), která je zaměřena na monitorování stability výkonu laseru v čase. Po spuštění této procedury je v průběhu měření sledován výkon laseru pomocí power meteru. Výsledkem je průměrná hodnota výkonu laseru a grafické zobrazení vývoje výkonu v čase. Z časového průběhu a výsledné hodnoty, uživatel rozhodne zda laser splňuje požadavky na stabilitu a je vhodný pro použití v dalších procedurách.



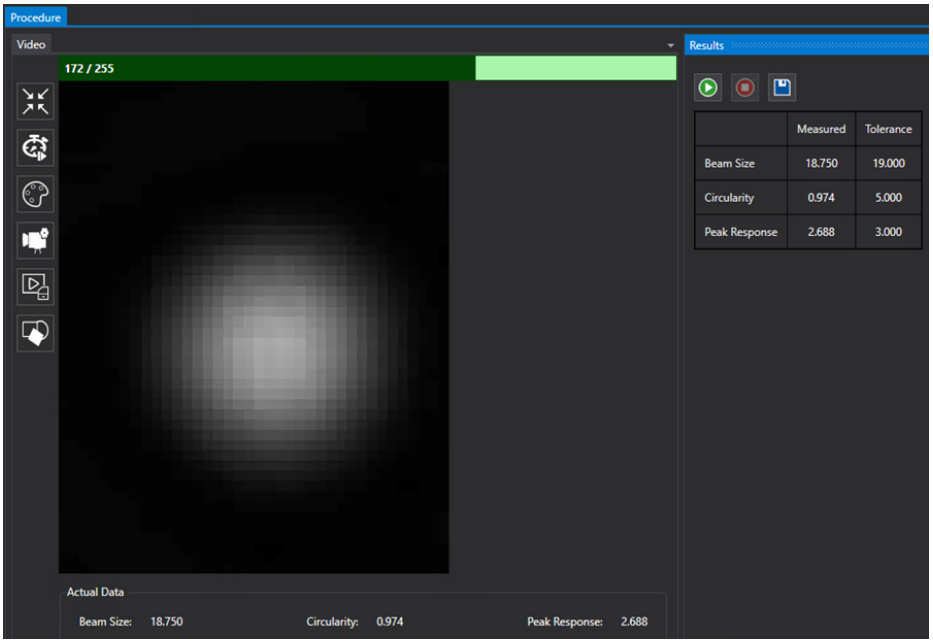
Obrázek 18 – Procedura pro měření stability laseru

Další procedurou je Procedura pro měření vlastností největšího spotu, která slouží k nalezení největšího spotu v obraze a analýze jeho vlastností. Zahrnuje měření velikosti a kruhovitosti tohoto spotu, přičemž výsledné hodnoty jsou porovnány s požadovanými hodnotami. Pokud při měření optických systémů zjistíme, že některý z parametrů nesplňuje specifikaci, může to naznačovat několik možných problémů s optickým systémem. Jednou z možností je deformace optických čoček. Pokud jsou čočky deformované nebo poškozené, může to vést k nesprávné tvorbě obrazu a nízké kruhovitosti spotu. Dalším možným problémem může být neoptimální zaostření optické soustavy. Nesprávné zaostření může mít za následek nekruhový tvar spotu, a to buď přílišnou nebo nedostatečnou zaostřeností. Problémy s kolimací jsou také možnou příčinou nízké kruhovitosti spotu. Nesprávná kolimace optického

systému může vést k nekruhovému tvaru spotu a rozmazání obrazu. Dále by mohly být problémy s nekvalitními optickými komponenty. Použití nekvalitních nebo poškozených komponentů může negativně ovlivnit kvalitu obrazu a vést k nízké kruhovitosti spotu. Neposlední řešení by mohlo být spočívat ve zlepšení osvětlení. Nedostatečné nebo nerovnoměrné osvětlení může mít vliv na tvar a kvalitu spotu, což může vést k nekruhovému obrazu. Pokud se objeví problémy s kruhovitostí spotu, je důležité provést detailní diagnostiku a identifikovat konkrétní příčinu. To může zahrnovat kontrolu a údržbu optického systému, jakož i možnou výměnu poškozených komponent.

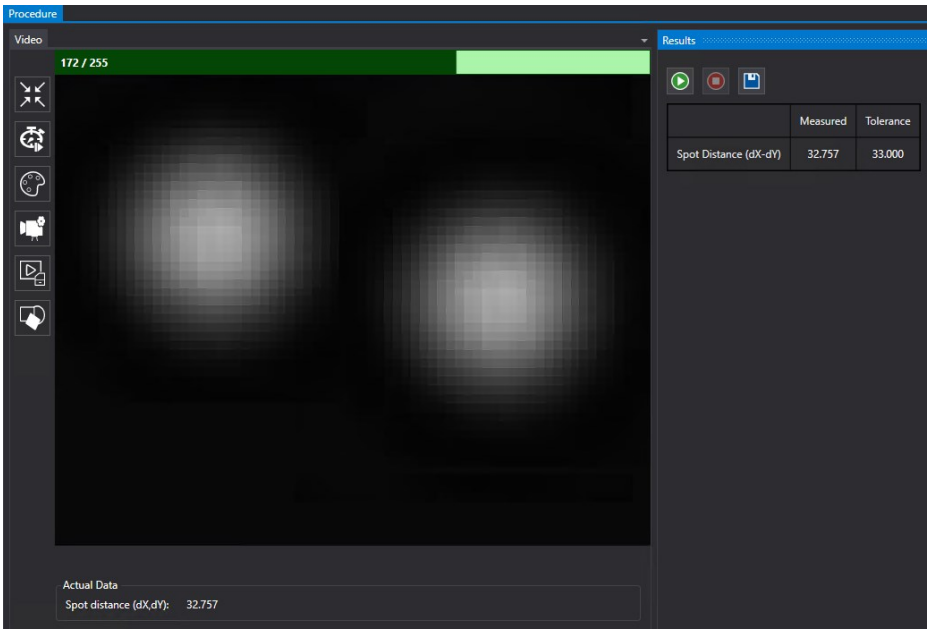


Obrázek 19 – Procedura pro měření vlastností spotu (před měřením)



Obrázek 20 – Procedura pro měření vlastností spotu (po měření)

Třetí procedurou je Procedura pro měření vzdálenosti spotů, která slouží k měření vzdálenosti mezi spoty v obraze. Tato procedura zahrnuje nalezení dvou spotů v obraze a výpočet jejich vzájemné vzdálenosti. Výsledkem je naměřená vzdálenost spotů která je porovnána s hodnotou specifikace. Pokud je naměřená vzdálenost mezi spoty větší než specifikovaná hodnota, výsledek je označen červeně a uživatel je informován o překročení specifikace. Pokud při měření zjistíme, že vzdálenost spotů nesplňuje specifikaci, může to naznačovat několik možných problémů s optickým systémem. Jednou z možností je nesprávné zaostření optického systému. Pokud je systém špatně zaostřený, může to vést k rozmazaným obrazům a zdánlivému oddělení bodů v obraze. Důsledkem může být to, že body se budou jevit dále od sebe, než ve skutečnosti jsou. Další možnou příčinou může být deformace optických čoček. Pokud jsou čočky deformované nebo poškozené, může to vést k nepřesnému zobrazení obrazu a zdánlivému oddálení bodů v obraze. Optická aberace může také hrát roli. Optická aberace může deformovat obraz a způsobit, že body v obraze budou vypadat dále od sebe, než ve skutečnosti jsou. To může být způsobeno vadnými optickými komponenty nebo nedostatečnou kvalitou optického systému. Nakonec, problémy s nastavením nebo kalibrací mohou vést k nesprávným měřením vzdálenosti mezi spoty. Pokud jsou dva spoty příliš daleko od sebe, je důležité provést detailní diagnostiku a identifikovat konkrétní příčinu problému. To může zahrnovat kontrolu zaostření optických komponentů, kontrolu integrity optických čoček a dalších komponentů, a dále optimalizaci nastavení optického systému. Oprava a kalibrace optického systému jsou nezbytné k zajištění přesných a spolehlivých výsledků měření.



Obrázek 21 – Procedura pro měření vzdálenosti dvou spotů

Takovýto pracovní postup umožňuje systematické a efektivní měření a vyhodnocování kvality optických systémů. Každá z těchto procedur přispívá k získání důležitých informací o výkonu a vlastnostech optických systémů, což umožňuje identifikaci případných problémů a jejich řešení. Získaná data lze dále využít k optimalizaci nastavení systému, monitorování jeho stability a kontrolu kvality výroby. Takovýto systematický přístup přispívá k zajištění spolehlivosti a výkonnosti optických systémů v různých aplikacích.

Můžeme tedy konstatovat, že využití frameworku .NET pro zpracování obrazu a implementaci komunikace s hardwarovými zařízeními prokázalo svou efektivitu a vhodnost v rámci této diplomové práce. Díky široké škále nástrojů a knihoven poskytovaných tímto frameworkem bylo možné efektivně vytvořit aplikaci, která splňuje požadavky na měření a vyhodnocování kvality optických systémů.

Dosavadní výsledky ukazují, že framework .NET poskytuje robustní prostředí pro vývoj sofistikovaných aplikací s využitím moderních technologií pro zpracování obrazu a komunikaci s hardwarem. Možnost využití jazyků jako C# či VB.NET a přístup k pokročilým knihovnám jako je například OpenCV umožňuje efektivní implementaci různých funkcionalit potřebných pro měření optických systémů.

V budoucnu je možné rozšířit funkcionalitu aplikace o další procedury pro měření a vyhodnocování optických systémů či implementovat další algoritmy pro zpracování obrazu. Dále lze zkoumat možnosti optimalizace výkonu aplikace a integraci s dalšími systémy a technologiemi pro ještě širší využití v praxi.

Celkově lze tedy konstatovat, že framework .NET představuje silný nástroj pro vývoj aplikací pro zpracování obrazu a komunikaci s hardwarem, který poskytuje dostatečnou flexibilitu a výkon pro potřeby aplikací v oblasti měření optických systémů a dalších aplikacích vyžadujících podobné funkcionality.

11 BEZPEČNOST

Bezpečnost aplikace je zajištěna tím, že je připojena pouze k interní síti společnosti, která je opatřena řadou bezpečnostních opatření. Jedním z klíčových opatření je použití firewalu, který monitoruje a řídí tok dat mezi interní a externí sítí. Autentizace a autorizace jsou také důležité, protože vyžadují silná hesla a dvoufaktorovou autentizaci, aby se zabránilo neoprávněnému přístupu k aplikaci. Šifrování dat zajišťuje, že veškerá komunikace v interní síti je chráněna před odposlechem. Tato síť je pečlivě monitorována a chráněna před neoprávněným přístupem. Stejně tak je zabezpečení databáze řešeno striktním omezením přístupu pouze na interní síť a použitím connection stringů chráněných hesly. Databáze samotná nenese žádné citlivé informace, což dále zvyšuje bezpečnost celého systému.

ZÁVĚR

Cílem této diplomové práce bylo vytvořit aplikace demonstrující využití frameworku .NET pro zpracování obrazu a integraci hardwaru v desktopových aplikacích. Práce byla zaměřena na vytvoření komplexní aplikace, která efektivně komunikuje s kamerami, výkonným laserem a power meterem, a přináší poznatky o implementaci komunikačních protokolů a práci s daty z těchto zařízení.

Důkladná analýza a implementace algoritmů pro zpracování obrazu prostřednictvím frameworku .NET ukázaly schopnost tohoto prostředí řešit složité úlohy detekce, analýzy a vizualizace obrazových dat. Integrování hardwaru do aplikace bylo úspěšné a zdůraznilo stabilitu a spolehlivost implementovaných komunikačních protokolů.

Výsledná aplikace slouží jako důkaz o širokých možnostech frameworku .NET a poskytuje užitečný nástroj pro další výzkum a vývoj v oblasti zpracování obrazu a integrace hardwaru. Její implementace a výsledky představují významný přínos k pochopení a využití těchto technologií v praxi.

Vhodnost použití frameworku .NET pro tuto aplikaci se ukázala jako velmi pozitivní. .NET framework poskytuje robustní a komplexní prostředí pro vývoj aplikací s využitím jazyků jako C# nebo VB.NET, které jsou dobře známé a široce používané v průmyslu.

Díky dobře navržené architektuře a silné podpoře ze strany Microsoftu je .NET framework stabilní a spolehlivý, což je klíčové pro úspěšnou integraci s hardwarovými zařízeními a implementaci komunikačních protokolů. Další výhodou je také široká podpora komunitních zdrojů a dokumentace, což usnadňuje řešení problémů a rychlý vývoj aplikace.

Celkově lze tedy konstatovat, že .NET framework poskytuje vhodné prostředí pro vývoj aplikací, které vyžadují zpracování obrazu a integraci s hardwarovými zařízeními. Jeho robustnost, široká podpora a efektivnost přispívají k úspěchu projektu a umožňují dosažení požadovaných výsledků.

Nicméně, je důležité zdůraznit, že tento projekt není konečným cílem, ale spíše začátkem nových možností a výzkumných směrů. Možnosti rozšíření aplikace nebo uplatnění získaných poznatků v jiných kontextech jsou obrovské. Dále se nabízí prostor pro optimalizaci implementace a zkoumání dalších technik zpracování obrazu či komunikačních protokolů.

SEZNAM POUŽITÉ LITERATURY

[1] DOBEŠ, Michal. Zpracování obrazu a algoritmy v C#. Praha: BEN - technická literatura, 2008. ISBN 9788073002336.

[2] GONZALEZ, Rafael C. a WOODS, Richard E. Digital image processing. Fourth edition. New York: Pearson, 2018. ISBN 978-1-292-22304-9.

[3] Medical Imaging and Diagnostics. [cit. 2024-04-08]. Dostupné z: https://courses.lumenlearning.com/suny-physics/chapter/32-1-medical-imaging-and-diagnostics/

[4] X-Rays. Online. Medlineplus. 2016 [cit. 2024-04-08]. Dostupné z: https://medlineplus.gov/xrays.html

[5] ULTRAVIOLET IMAGING. Online. MIDOPT. [cit. 2024-04-08]. Dostupné z: https://midopt.com/solutions/monochrome-imaging/ultraviolet-imaging/

[6] AXELSON, Jan. Serial port complete. Madison: Lakeview Research, 2000. ISBN 0-9650819-2-3.

[7] BAI, Ying. The windows serial port programming handbook. Auerbach Publications, 2004.

[8] EDWARDS, James; BRAMANTE, Richard. Networking self-teaching guide: OSI, TCP/IP, LANs, MANs, WANs, implementation, management, and maintenance. John Wiley & Sons, 2015.

[9] Microsoft technical documentation. *Microsoft technical documentation* [online]. Microsoft, 2022 [cit. 2024-02-02]. Dostupné z: https://docs.microsoft.com/en-us/

[10] ALANI, Mohammed M. Guide to OSI and TCP/IP models. 2014.

[11] BUONANNO, Enrico. Functional Programming in C#: How to write better C# code. Manning, 2017. ISBN 978-1617293955.

[12] WATSON, Ben. Writing High-Performance .NET Code. 2nd. 2018. ISBN 978-0990583431.

[13] YOSIFOVICH, Pavel. Windows Presentation Foundation 4.5 Cookbook. Packt Publishing, 2012. ISBN 9781849686228.

[14] XU, Jack. Practical WPF Charts and Graphics. 2010th edition. Apress, 2011. ISBN 9781430223160.

[15] Swift Documentation. *Swift* [online]. Apple, 2022 [cit. 2024-02-08]. Dostupné z: https://www.swift.org/

[16] Best frameworks for desktop application development. Solace Infotech Pvt. Ltd. [online]. Solace Infotech Pvt., 2021 [cit. 2024-02-02]. Dostupné z: https://solaceinfotech.com/blog/best-frameworks-for-desktop-application-development/

[17] SwiftUI. *Apple developer* [online]. Apple, 2022 [cit. 2024-02-02]. Dostupné z: https://developer.apple.com/xcode/swiftui/

[18] Qt [online]. The Qt Company, 2022 [cit. 2024-02-02]. Dostupné z: https://www.qt.io

[19] Electron Documentation. *Electron* [online]. OpenJS Foundation, 2022 [cit. 2022-03-30]. Dostupné z: https://www.electronjs.org/docs/latest

[20] Advancement of Swing. *Section* [online]. Section, 2021 [cit. 2024-02-02]. Dostupné z: https://www.section.io/engineering-education/advancement-of-swing-frameworks-in-java/

[21] Documentation archive. *Documentation Archive* [online]. Apple, 2013 [cit. 2022-03-30]. Dostupné z: https://developer.apple.com/library/archive/documentation/Cocoa/

[22] 2021 Developer survey. Stack owerflow [online]. Stack Exchange, 2021 [cit. 2024-02-08]. Dostupné z: https://insights.stackoverflow.com/survey/2021

[23] GAMMA, Erich; HELM, Richard; JOHNSON, Ralph a VLISSIDES, John. Design patterns: elements of reusable object-oriented software. Addison-Wesley professional computing series. Boston: Addison-Wesley, 1995. ISBN 0201633612.

[24] FOWLER, Martin. Patterns of enterprise application architecture. The Addison-Wesley signature series. Boston: Addison-Wesley, 2003. ISBN 0321127420.

[25] SKIENA, Steven. The Algorithm Design Manual. Second edition. Springer, 2008. ISBN 978-1849967204.

[26] OWASP Top Ten. OWASP [online]. OWASP Foundation, 2021 [cit. 2024-02-02]. Dostupné z: https://owasp.org/www-project-top-ten/

[27] OWASP Cheat Sheet Series. OWASP Cheat Sheet Series [online]. Attribution 3.0 Unported, 2021 [cit. 2024-02-02]. Dostupné z: https://cheatsheetseries.owasp.org/

[28] What is server-side request forgery. Acunetix [online]. Acunetix 2022, by Invicti, 2022 [cit. 2024-02-02]. Dostupné z: https://www.acunetix.com/blog/articles/server-side-request-forgery-vulnerability/

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

CAN	Controller Area Network
CSS	Cascading Style Sheet
DDoS	Distributed denial of Service
DIP	Dependency Inversion Principle
DoS	Denial of Service
FIFO	Fist In First Out
GPS	Global Positioning System
HTML	Hypertext Markup Language
ISP	Interface Segregation Principle
LAN	Local Area Network
LSP	Liskov Substitution Principle
MVVM	Model-View-ViewModel
MVC	Model-View-Controller
MVP	Model-View-Presenter
NMEA	National Marine Electronics Association
OBD	On Board Diagnostics
OCP	Open Closed Principle
PN	Part Number
SDK	Software Development Kit
SRP	Single Responsibility Principle
SQi	SQL Injection
SQL	Structured Query Language
UART	Universal asynchronous receiver transmitter
USB	Universal Serial Bus
UWP	Universal Windows Platform

VB	Visual Basic
WinUI	Windows UI Library
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language
XML	Extensible Markup Language
XSS	Cross Site Scripting

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM KÓDŮ

SEZNAM PŘÍLOH

Příloha P1: CD obsahující zdrojový kód, diplomovou práci a videa demonstující použití aplikace