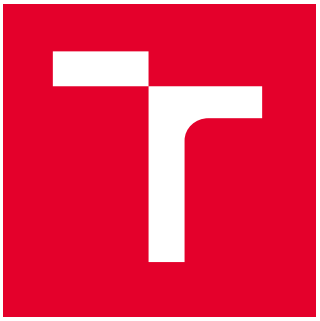


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

AUTOMATICKÉ ODEČET ČÍSEL - MĚŘENÍ VELIČINY Z GRAFICKÝCH DISPLEJŮ RŮZNÝCH PŘÍSTROJŮ V REÁLNÉM ČASE KAMEROU

AUTOMATIC READING OF NUMBERS - REAL-TIME MEASUREMENT OF QUANTITIES FROM GRAPHICAL DISPLAYS OF VARIOUS INSTRUMENTS WITH A CAMERA

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — **Daniel Bartoš**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Martin Appel, Ph.D.**

SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	**Daniel Bartoš**
Studijní program:	Mechatronika
Studijní obor:	bez specializace
Vedoucí práce:	**Ing. Martin Appel, Ph.D.**
Akademický rok:	2023/24

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Automatické odečet čísel – měření veličiny z grafických displejů různých přístrojů v reálném čase kamerou

Stručná charakteristika problematiky úkolu:

Cílem této bakalářské práce je vytvořit systém pro automatický odečet kalibrovaných přístrojů s vizuálním displejem, ale bez komunikačního rozhraní. Kamera bude snímat hodnoty z displeje a následně bude zpracovávána algoritmem pro identifikaci čísel. Klíčová část algoritmu se zaměří na odstranění vizuálního šumu, jako jsou odrazy a stíny. Po získání čistých dat budou tyto hodnoty podrobeny statistickému zpracování pro normalizaci a analýzu chyb. Závěrečná fáze projektu zahrnuje testování efektivity a přesnosti celého systému, což poskytne základ pro jeho širší nasazení.

Cíle bakalářské práce:

– Snímání a separace vybraných segmentů displejů programem v reálném čase kamerou
– Zpracování a úprava jednotlivých snímků před vyhodnocením zobrazovaných údajů
– Rozpoznávání žádaných hodnot a jejich úprava na data
– Odeslání a zpracování získaných dat do databází nebo virtuálních portů

Seznam doporučené literatury:

GREPL, Robert. Kinematika a dynamika mechatronických systémů. Brno: Akademické nakladatelství CERM, 2007. ISBN 978-80-214-3530-8.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2023/24

V Brně, dne

L. S.

prof. Ing. Jindřich Petruška, CSc. ředitel ústavu	doc. Ing. Jiří Hlinka, Ph.D. děkan fakulty

Abstrakt

Tato bakalářská práce se zabývá procesem převedení číselných hodnot zobrazovaných na grafických displejích měřících přístrojů bez komunikačního rozhraní na data, použitelná pro následnou analýzu, využitím kamery a programu popsaného v realizační části.

Teoretická část je věnovaná rešerši a implementaci operací používaných na zpracování obrazu, včetně operací morfologických, v prostředí *MATLAB*.

Praktická část práce je zaměřena na vytvoření programu pro zaznamenávání číselných displejů v reálném čase kamerou, úpravou pořízených snímků operacemi probranými v rešeršní části a jejich čtení pomocí *MATLAB* funkce *OCR*, dále na následné zpracování přečtených dat a jejich ukládání v průběhu měření.

Tato část také obsahuje ukázku měření různých typů displejů, vytvoření uživatelského prostředí a aplikace využitím nástroje *Application Compiler*. Práce je zakončena sekcí sloužící, jako návod k používání aplikace pro uživatele.

Summary

This bachelor thesis deals with the process of converting numerical values displayed on graphical displays of measuring instruments without communication interfaces into data usable for subsequent analysis, using a camera and a program described in the implementation section.

The theoretical part is dedicated to the research and implementation of operations used in image processing, including morphological operations, in the *MATLAB* environment.

The practical part of the thesis focuses on creating a program for real-time recording of numerical displays using a camera, processing the captured images with operations discussed in the research section, and reading them using the *MATLAB OCR* function. It also covers the subsequent processing of the read data and their storage during the measurement.

This part also includes examples of measuring different types of displays, creating a user interface and application using the *Application Compiler* tool. The thesis concludes with a section serving as a user guide for the application.

Klíčová slova

zpracování obrazu, morfologické operace, OCR, MATLAB, měření, displej

Keywords

image processing, morphological operations, OCR, MATLAB, measurement, display

Bibliografická Citace

BARTOŠ, D. *Automatické odečet čísel – měření veličiny z grafických displejů různých přístrojů v reálném čase kamerou.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2024. 49 s., Vedoucí bakalářké práce: Ing. Martin Appel, Ph.D..

Prohlašuji, že tato práce je mým původním dílem, zpracoval jsem ji samostatně pod vedením Ing. Martina Appela, Ph.D. a s použitím informačních zdrojů uvedených v seznamu.

Daniel Bartoš

Brno . . . .

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Martinu Appelovi, Ph.D. za ochotu a cenné rady při vytváření této bakalářské práce.

Daniel Bartoš

Obsah

1 Úvod

Hlavním důvodem každého měření je zajisté získání analyzovaných hodnot, na jejichž základě je zvolen následný postup. V dnešní době se čím dál více spoléháme na moderní technologie, které nám usnadňují mnoho každodenních úkolů. Jedním z těchto úkolů je i snímání a zpracování naměřených dat z kalibračních přístrojů.

V praxi se často setkáváme s měřícími přístroji, které disponují pouze vizuálním rozhraním a nejsou vybaveny zaznamenávacími a komunikačními možnostmi pro jejich analýzu. Tento fakt může způsobit komplikace při získávání a zpracování naměřených dat. V minulosti jsme se často uchylovali k ručnímu zaznamenávání hodnot, což však není ideální přístup zejména v případech, kdy vyžadujeme přesnost, konstantní periodu zaznamenávání v rámci několika sekund a delší dobu měření.

Proto je cílem této bakalářské práce právě vytvořit aplikaci, která umožní automatizovat proces snímání hodnot z digitálních číslicových displejů v reálném čase pomocí kamery, následné zpracování snímků a uložení získaných dat pro další analýzu. Tímto způsobem se usnadní a zrychlí měření.

Bakalářská práce se zaobírá technickými a algoritmickými aspekty vývoje této aplikace. Hlavním zaměřením je vytvoření algoritmu pro snímání, separaci a zpracování jednotlivých segmentů displejů v reálném čase. Dále se práce soustředí na identifikaci a úpravu získaných hodnot a jejich následné uložení do databází nebo virtuálních portů pro další zpracování.

Cílem této práce je tedy poskytnout uživatelům efektivní nástroj pro snadné a v rámci možností spolehlivé zpracování dat z digitálních číslicových displejů, který má potenciál uplatnění v různých odvětvích.

V první části je provedena rešerše zaměřující se na úpravu obrázků v prostředí *MATLAB*, konkrétně na morfologické operace a jejich použití pro tuto tématiku. Poté následuje část, jež se zaobírá samotnou realizací aplikace a řešením problémů s ní spojenou.

2 Rešerše

Pro účely optického rozpoznávání znaků, dále *OCR*, v prostředí *MATLAB* je nezbytné předzpracovat snímky zaznamenané pomocí kamery, jelikož mohou být různého formátu, kvality a obsahovat určitý stupeň šumu. Cílem tohoto předzpracování je zvýšit pravděpodobnost úspěšného rozpoznání znaků pomocí *OCR* funkce v *MATLABu* a tím nám poskytnout použitelná data pro následnou analýzu.

Pro úpravu obrazu jsou v *MATLABu* k dispozici balíček *Image Processing Toolbox*, který nabízí širokou škálu funkcí pro zpracování obrazu. Tyto funkce umožňují transformovat surový snímek z kamery na čitelný binární obraz, což je nezbytné pro efektivní funkci *OCR*.

Při pořizování snímků v různých prostředích může docházet k nepravidelnému osvětlení, které ovlivňuje kvalitu obrazu a následně úspěšnost *OCR* procesu. Nepravidelné osvětlení může být způsobeno faktory jako je změna osvětlení, odlesky a další.[1]

Jedním z přístupů k vyrovnání nepravidelného osvětlení jsou tzv. *tophat* transformace, které se zaměřují na odstranění nežádoucích účinků nepravidelného osvětlení z obrazu. Tyto transformace budou podrobněji prozkoumány v kapitole 2.4.6 rešerše.

Cílem této rešerše je seznámit se s jednotlivými operacemi úpravy obrazu dostupnými v *MATLABu* a jejich použitím pro efektivní aplikaci při optickém rozpoznávání znaků.[2]

2.1 Morfologické operace

Morfologické operace jsou matematické operace používané k modifikaci tvarů a struktur obrazů. Tyto operace pracují na základě geometrických vlastností objektů v obraze a jsou často používány k úpravě, zpracování a analýze obrazů. Základní morfologické operace zahrnují erozi, dilataci, otevření, uzavření a další, kde každý pixel v obrázku je pozměněn na základě hodnoty pixelů v jeho blízkém okolí, kterého velikost nastavíme podle potřeby dané operace.

Jsou často používány v oblastech zpracování obrazu jako je detekce hran, segmentace objektů, odstranění šumu, zpracování textury a další. Jsou to důležité nástroje pro úpravu obrazů a extrakci užitečných informací z obrazových dat.

Eroze a dilatace jsou základními stavebními bloky pro složitější morfologické operace, jako je například otevření, uzavření a tophat transformace. Tyto operace se používají pro černobílé obrázky, což znamená, že je nejprve potřeba převést barevné *RGB* obrázky na černobílé *grayscale*, aby bylo možné aplikovat zmíněné operace. Je také důležité zmínit, že v morfologii se černá bere jako barva pozadí a bílá objektu.[3][4]

2.2 Preprocessing

V této kapitole budou probrány kroky upravení obrazu nutné pro následné použití morfologických operací v kapitole 2.4.

2.2.1 Převod do grayscale

Každý barevný obrázek vyfocený kamerou je v *MATLABu* nahrazen třemi maticemi, jejichž velikost odpovídá počtu pixelů daného obrázku. Jednotlivé matice pak reprezentují složky červené, zelené a modré všech pixelů obrázku a intenzita těchto barev je znázorněna číslem od 0 do 255, kdy druhá hodnota odpovídá maximu dané barvy na daném pixelu.

Pro převod z *RGB* formátu na formát *grayscale* se v *MATLABu* používá funkce `im2gray` nebo `rgb2gray`, která předešlé tři matice nahradí maticí jednou. V této matici je každý pixel obrázku interpretován číslem 0 až 255, kde 0 představuje černou a 255 bílou.

Tedy funkce `rgb2gray` převádí barevný obraz *RGB* na obraz ve stupních šedi, kdy se odstraní informace o odstínu a sytosti a zachovají se pouze hodnoty intenzity jasu. Funkce pro převod do *grayscale* funguje následovně.

Originální obraz má formu *MxNx3* matice, kde *MxN* reprezentuje dimenze obrazu a třetí dimenze, jak už bylo zmíněno, představuje jednotlivé barvy červené, zelené a modré.

Konverze do odstínů šedé funguje na základě váženého průměru zmíněných tří barev, kde každá z nich má různou váhu, která koresponduje s jasovou sensitivitou lidského oka. Často používaná formule pro výpočet intenzity šedi vypadá následovně:

$$I = 0.2989R + 0.5870G + 0.1140B \tag{2.1}$$

kde:

- R, G a B jsou pixelové intenzity červené, zelené a modré
- I je intenzita šedi od 0 do 255

Výstupem této formule, pokud jednotlivé složky barev nabývají hodnot v rozmezí 0 až 255, je pak hodnota intenzity šedi pro každý pixel v rozmezí 0 až 255.[5]

Obrázek 2.1: Barevný obrázek z kamery

Obrázek 2.2: Obrázek převedený do stupní šedi

Obrázek 2.1 vlevo představuje originální barevný obrázek vyfocený kamerou a budou na něj aplikovány některé operace probírané v rešeršní části. Na pravém obrázku 2.3 vidíme obrázek převedený do černobílé formy pomocí následujícího kódu:

```
grayImage = rgb2gray(rgbImage)
```

2.2.2 Zvýraznění obrazu

Po převedení je možné obraz dále vylepšit, a to zvýrazněním intenzit pixelů. V ideálním případě chceme upravený obrázek v binární formě, tedy bílé objekty na černém pozadí, popřípadě opačně, k čemuž nám dále využijeme binarizační funkci. Aby ovšem tato funkce binarizaci provedla co možná nejlépe potřebujeme co nejvýraznější hranice mezi objekty a pozadím a jelikož se pohybujeme v rozmezí 0 až 255, tak to znamená, že by se pozadí a objekty měly na této škále pohybovat co nejdále od sebe.

Například pokud bychom měli černý objekt na šedém pozadí, tak bychom chtěli tuto šedou barvu co nejvíce zbělet a objekty zčernit, k tomu nám pomůže funkce `imadjust`, která pixely s vyšší hodnotou zesvětlí a ty s nižší naopak ztmaví.

Složení barev v obrázku si můžeme zobrazit pomocí takzvaného histogramu, který na ose X vyobrazuje intenzitu od 0 do 255 a počet pixelů na ose Y, tím nám tedy histogram říká, kolik pixelů se nachází v konkrétní intenzitě. Pokud bychom tedy měli náš světlý objekt na tmavém šedém pozadí, v histogramu bychom měli dvě špičky, například v okolí 200 a 100. Pak funkce `imadjust` provede roztažení těchto špiček od sebe a tím pádem zvětší rozdíl kontrastu mezi objektem a pozadím, což vylepší následnou binarizaci obrázku.[6]

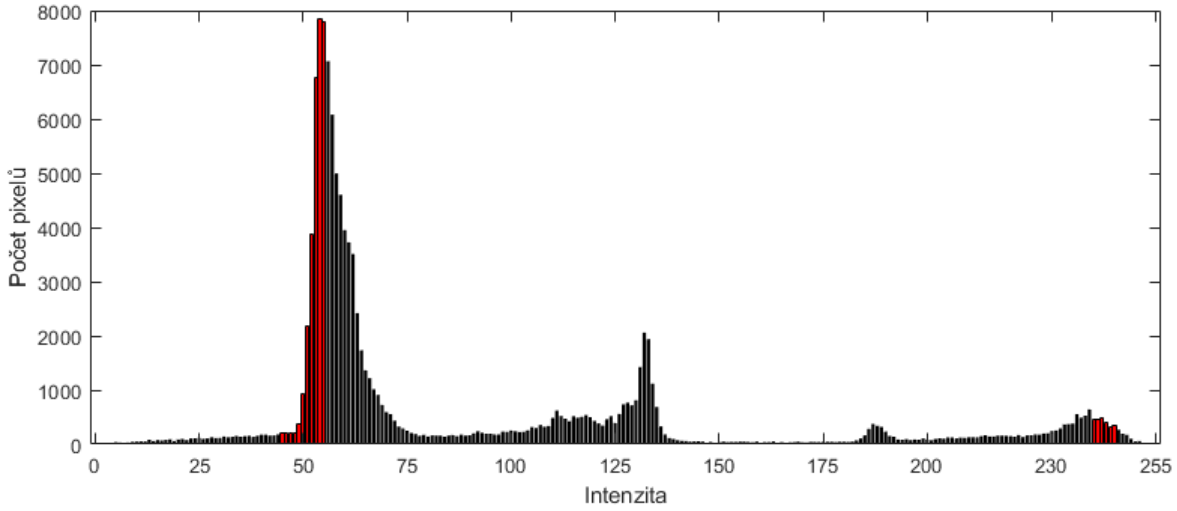
V *MATLABu* se tato operace provede následovně:

```
adjustedImage = imadjust(grayImage)
```

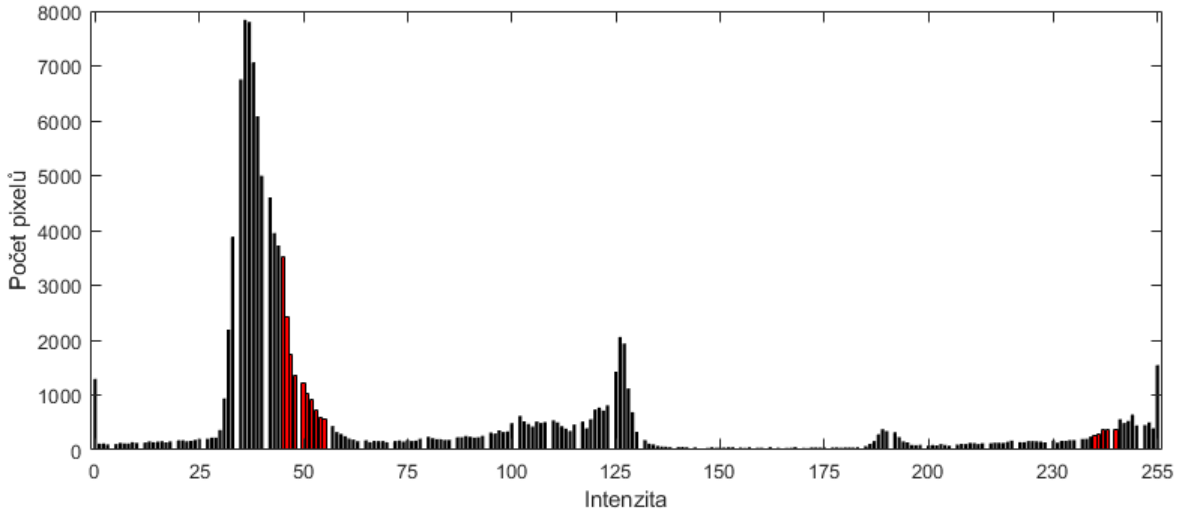
Obrázek 2.3: Obrázek ve stupních šedi

Obrázek 2.4: Obrázek po zvýraznění

Při porovnání obrázků můžeme vidět, že po aplikaci zvýraznění na levý obrázek 2.3 se na obrázku 2.9 zintenzivnila černá pozadí a bílá jednotlivých čísel, což bude lépe vidět na níže zobrazených histogramech obou obrázků.



Obrázek 2.5: Histogram obrázku ve stupních šedi



Obrázek 2.6: Histogram zvýrazněného obrázku

Jak můžeme vidět na histogramech, většina pixelů pozadí u původní intenzity 2.5 se pohybuje v intervalu 50-75, zatímco po zvýraznění intenzit 2.6 se nám většina pixelů pozadí ztmavila do intervalu 25-50, to samé můžeme vidět u pixelů čísel které se více přesunuly k hodnotě 255.

2.2.3 Filtrace obrazu

Při focení obrázků se často jejich kvalita může snížit kvůli zvýšené citlivosti kamery při slabém osvětlení a dostaneme zašuměný obrázek. Tento šum nám pak komplikuje převod do binárního stavu, jelikož některé oblasti pixelů mají díky šumu mnohem odlišnější hodnotu než jejich okolí, proto by se nám hodilo použít filtr, který průměruje hodnotu každého pixelu na základě svého okolí a tím snižuje vliv šumu a celkově vyhlazuje obrázek, pro vytvoření takového filtru použijeme funkci `fspecial`.

```
filtr = fspecial('average',19);
```

Výše uvedený příkaz vytvoří průměrovací filtr o rozměrech 19x19 pixelů, v *MATLABu* vznikne 19x19 matice, ve které budou všechny prvky rovny $1/19^2$ a jejich součet bude vždy 1.

Pro aplikaci vytvořeného filtru použijeme funkci `imfilter`, jejíž aplikaci v *MATLABu* můžeme vidět na níže uvedeném kódu, ta tímto vytvořeným oknem projde celý obrázek, kdy se posouvá po jednom pixelu a průměruje hodnoty pixelů právě se nacházejících v okně.

```
filteredImage = imfilter(grayscaleImage, H,'replicate');
```

Režim `replicate` vy výše uvedeném kódu zajišťuje, že hodnoty pixelů mimo hranice obrazu jsou považovány za rovny nejbližším hodnotám na hranici, čímž minimalizuje artefakty na okrajích obrazu.[7]

Obrázek 2.7: Obrázek ve stupních šedi

Obrázek 2.8: Obrázek po použití filtrace

Při porovnání obrázků vidíme, že dolní 2.8 vypadá více rozmazaně než horní 2.7, avšak odstranili jsme z něj velké množství šumu a celkově jej vyhladili, tím pádem bude následné uzavření znaků a binarizace kvalitnější.

2.2.4 Binarizace

Binarizace obrazu je proces, který provede přeměnu graysacale černobílého obrazu na obraz binární, kde každý pixel reprezentuje bílou nebo černou. Tato separace pixelů do dvou skupin, pozadí a objektu, se provádí na základě globálně vypočteného prahu, kdy

hodnoty nad touto hranicí se nastaví do 1, tedy bílého objektu a zbylé do 0, tedy černého pozadí.

Hodnota prahu se dá určit buď manuálně, kdy si zvolíme určitou hodnotu nebo automaticky pomocí nějaké z metod na základě charakteristiky obrázku. Pro automatické určení prahu se často používá Otsuova metoda.[8]

Otsuova metoda hledá optimální intenzitu prahu, tak že volí různé intenzity, podle nichž obraz rozdělí na pozadí a objekt a následně vypočte mezitřídní variace podle níže uvedené rovnice 2.2. Dále algoritmus vybere prah, který maximalizuje tuto variaci, tedy hledá, pro který prah je mezitřídní variace největší a ten zvolí jako optimální.[9]

$$\sigma_B^2 = W_b \times W_f \times (\mu_b - \mu_f)^2 \tag{2.2}$$

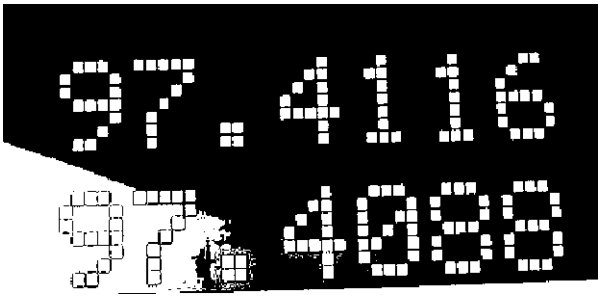
kde:

- W_b je počet pixelů pozadí podělený celkovým počtem pixelů
- W_f je počet pixelů objektu podělený celkovým počtem pixelů
- μ_b průměrná intenzita pozadí
- μ_f průměrná intenzita objektu
- σ_B meziskupinová variace

V MATLABu binarizaci pomocí prahu určeného podle Utsuho metody provedeme následovně:

```
threshold = graythresh(grayscaleImage)
binarizedImage = imbinarize(grayscaleImage,threshold)
```

Obrázek 2.9: Obrázek ve stupních šedi



Obrázek 2.10: Binarizovaný obrázek

Můžeme vidět, že pokud bychom chtěli binarizovat obrázek 2.9 vlevo, tak by binarizace kvůli odlesku neproběhla velmi úspěšně, jak vidíme na obrázku 2.10 vpravo. Proto je nutné nejdříve, než budeme obraz binarizovat odstranit nekonstantní pozadí, což bude probráno dále v kapitole 2.4.6.

2.3 Maska

Pro všechny funkce, upravující obrázek musíme vytvořit masku pomocí níž bude proces prováděn. Vytvoření vhodné masky, její velikosti a popřípadě tvaru, je jedna z nejdůležitějších částí pro úspěšnou úpravu obrázku. Maska je malý obrazový prvek o velikosti v pixelech, který určuje tvar a velikost operace.

Maska, také nazývaná jádro nebo strukturální prvek, je malý obrazový vzorek, který se používá v morfologickém zpracování obrazu. Tento prvek určuje tvar a velikost operací jako eroze, dilatace, openning a dalších.

Maska je obvykle binární matice, které obsahuje hodnoty 0 a 1, přičemž 1 označuje oblast, která je součástí proužku nebo objektu, a 0 označuje oblast, která součástí není. Tvar a velikost strukturálního prvku závisí na konkrétní aplikaci a účelu operace. Může to být obdélníková, čtvercová, kruhová nebo jiná geometrická forma. Velikost strukturálního prvku ovlivňuje rozsah, ve kterém se operace provádí, a malé změny v tomto parametru mohou mít významný vliv na výsledky zpracování obrazu.[10]

Maska může být vytvořena ručně pomocí základních funkcí *MATLABu*, jako jsou `ones`, `zeros` nebo `strel`. Strel objekt představuje 2D prostor, čtvercovou binární matici, kde prvky s hodnotou 1, jsou použity pro morfologickou operaci, zatímco nulové prvky pouze vyplňují zbytek matice. Pixel přímo ve středu masky identifikuje pixel obrázku, který je právě zpracováván.

Syntaxe pro použití `strel` může vypadat například, jako na níže uvedeném kódu, kde první v závorce je určen tvar prvku, zde `disk`, ale můžeme použít i jiné například `diamond`, `oktagon`, `line` a druhý následuje poloměr, který značí počet sloupců matice od středu, ten se může u různých prvků lišit.[11][2]

```
SE = strel('disk',5)
```

	3	2	1	0	1	2	3
3	0	0	0	1	0	0	0
2	0	0	1	1	1	0	0
1	0	1	1	1	1	1	0
0	1	1	1	1	1	1	1
1	0	1	1	1	1	1	0
2	0	0	1	1	1	0	0
3	0	0	0	1	0	0	0

Obrázek 2.11: Maska tvaru diamant

	4	3	2	1	2	3	4
4	0	0	1	1	1	0	0
3	0	1	1	1	1	1	0
2	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1
3	0	1	1	1	1	1	0
4	0	0	1	1	1	0	0

Obrázek 2.12: Maska tvaru disk

Na obrázku 2.11 vlevo můžeme vidět masku vytvořenou příkazem `strel('diamond',3)`, zatímco vpravo 2.12 stejně velký prvek tvaru disku vytvoříme příkazem `strel('disk',4)`, je tedy vhodné zkontrolovat reálnou velikost masky pro jednotlivé tvary.

2.4 Jednotlivé morfologické operace

V této kapitole budou podrobněji probrány morfologické operace, eroze, dilatace, *opening, closing* a *tophat a bottomhat* transformace, a jejich použití.

2.4.1 Eroze

Operace eroze, je proces zmenšování objektů, který postupně odstraňuje jednotlivé prvky objektu, které se dotýkájí pozadí.

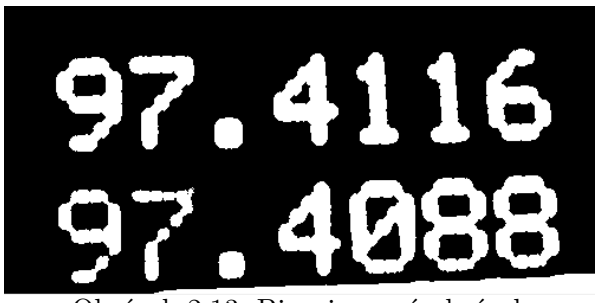
Během eroze je vytvořená maska projížděna přes celý obraz. Tam, kde se maska shoduje s objektem, je tento objekt zachován, naopak, tam, kde se strukturální prvek nepřekrývá s objektem, je tento objekt odstraněn.

Hodnota pixelu v centru masky, kterým je snímek projížděn je změněna na nejnižší hodnotu ze všech pixelů, které se právě nachází v rámci strukturálního prvku. Tím se dosáhne zmenšení objektů a zvýraznění jejich hraničních oblastí.

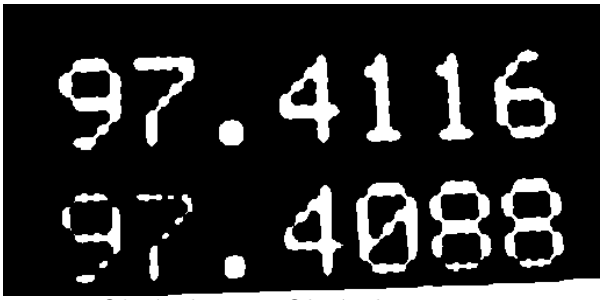
Okrajové pixely objektu jsou změněny na pixely pozadí což má za následek zvýraznění pozadí a zachování pouze výrazných objektů, které jsou zároveň zmenšeny.[3]

V *MATLABu* je operace realizována pomocí funkce `imerode` následovně:

```
se = strel('disk',3);
erodedImage = imerode(grayscaleImage,se);
```



Obrázek 2.13: Binarizovaný obrázek



Obrázek 2.14: Obrázek po erozi

Na pravém obrázku 2.14 vidíme erozi obrázku 2.13 pomocí disku o poloměru 3, můžeme si všimnout, že jednotlivé znaky se výrazně zmenšili na úkor pozadí.

2.4.2 Dilatace

Dilatace je operace v morfologickém zpracování obrazu, která rozšiřuje každý prvek pozadí, který se dotýká objektu. Tímto procesem dochází k zvětšení a zvýraznění objektů v obraze a ke spojení blízkých bodů objektů dohromady.

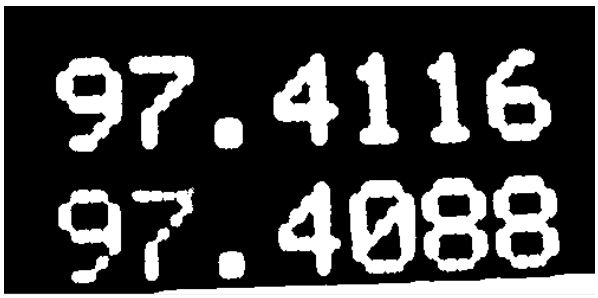
Hodnota pixelu v centru masky, kterým je snímek projížděn je změněn na nejvyšší hodnotu ze všech pixelů, které se právě nachází v rámci strukturálního prvku.

Během dilatace je maska projížděna přes celý obraz. Hodnota pixelu v centru masky, kterou je snímek projížděn, je změněna na nejvyšší hodnotu ze všech pixelů, které se nacházejí v rámci prvků masky a tím se objekty v obraze rozšiřují.

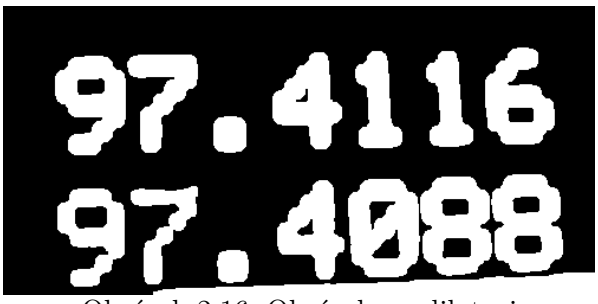
Na místech, kde se prvky masky shodují s objektem, je tento objekt zachován a rozšířen. Okrajové pixely pozadí jsou během dilatace změněny na pixely objektu, což má za následek spojení a zvětšení objektů v obraze.[3]

V *MATLABu* je operace realizována pomocí funkce `imdilate` následovně:

```
se = strel('disk',3);
dilatedImage = imdilate(grayscaleImage,se);
```



Obrázek 2.15: Binarizovaný obrázek



Obrázek 2.16: Obrázek po dilataci

Na pravém obrázku 2.16 vidíme dilatovaný levý obrázek 2.13 pomocí masky tvaru disku o poloměru 3, můžeme si všimnout, že jednotlivé znaky se v tomto případě naopak výrazně zvětšili na úkor pozadí.

2.4.3 Closing a Opening

Operace *closing* a *opening* jsou důležité techniky morfologického zpracování obrazu, které kombinují erozi a dilataci za účelem úpravy tvarů objektů v obraze.

Tyto operace jsou užitečné pro předzpracování obrazu před další analýzou nebo extrakcí objektů. Uzavření a otevření jsou často používány pro odstranění šumu, vyhlazení hran a zvýraznění objektů v obraze.

Je důležité poznamenat, že obě operace používají stejný strukturální prvek pro dilataci i erozi.

2.4.4 Opening

Operace *opening* je užitečná pro odstranění šumu, zjemnění hran a oddělení přilehlých objektů v obraze. Používá se například při předzpracování obrazu před segmentací nebo při odstraňování malých artefaktů z obrazových dat.

Tato operace je definovaná jako eroze následovaná dilatací stejným strukturálním prvkem, tedy nejprve je na vstupní obraz aplikována eroze, čímž se zmenší a oddělí objekty od pozadí, poté je aplikována dilatace, jež zvětší velikost původních výraznějších objektů, které byly zachovány po erozi.

Tento postup pomáhá odstranit malé a oddělit blízké objekty zároveň také vyhlazuje hrany objektů.[3]

V *MATLABu* je operace realizována pomocí funkce `imopen` následovně:

```
se = strel('disk',4);
openedImage = imopen(grayscaleImage,se);
```

Vpravo na obrázku 2.18 vidíme jak se v porovnání s 2.17, při použití funkce `imopen` s maskou o poloměru 4, tenké části v obou číslech změnily na pozadí a také se vyhladily jejich hrany.

Obrázek 2.17: Binarizovaný obrázek

Obrázek 2.18: Obrázek po operaci opening

2.4.5 Closing

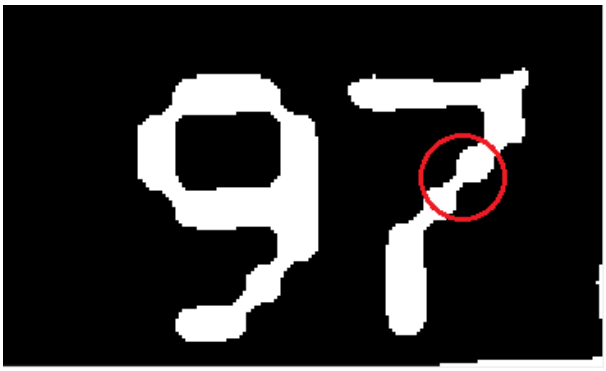
Operace *closing* je užitečná pro uzavření malých děr v objektech, spojení blízkých objektů, vyhlazení hran a odstranění malých objektů a šumu z obrazu. Odstraní malé a tenké prvky pozadí.

Tato operace je definovaná jako dilatace následovaná erozí stejným strukturálním prvkem, tedy nejprve se na vstupní obraz aplikuje dilatace, čímž zvětší objekty a spojí blízké body objektů dohromady, následně je aplikována eroze, jež zmenší objekty v obraze a oddělí je od okolního pozadí, tímto krokem se vyhlazují hrany objektů a odstraňují se malé objekty a šum.[3]

V *MATLABu* je operace realizována pomocí funkce `imclose` následovně:

```
se = strel('disk',6);
closedImage = imclose(grayscaleImage,se);
```

Obrázek 2.19: Binarizovaný obrázek



Obrázek 2.20: Obrázek po operaci closing

Vpravo na obrázku 2.20 vidíme jak se, při použití funkce `imclose` s maskou o poloměru 6, menší části pozadí v okolí obou čísel změnily na objekt a tím spojily mezery ve znacích na obrázku 2.19 a stejně jako v předchozím případě také vyhladily jejich hrany.

2.4.6 Tophat transformace

Tophat transformace je operace, která se používá k zvýraznění větších struktur a hran v obraze. Jedná se o kombinaci původního obrazu a jeho *openingu*, který odstraní jemné detaily a malé objekty. Tímto způsobem *tophat* transformace vytváří efekt, který zdůraz-

ňuje pouze větší struktury a hrany v obraze.

Tato transformace je využívána v různých oblastech zpracování obrazu, včetně detekce hran, hledání špiček intenzit v obrazech, segmentace objektů a odstranění šumu v obrazech s nejednotným osvětlením, což zlepšuje kvalitu obrazu pro další zpracování a analýzu.

V *MATLABu* je pro provedení *tophat* transformace k dispozici funkce `imtophat`, která provádí *opening* obrázku pomocí specifikovaného strukturního prvku a následně tento upravený obrázek odečte od původního obrázku. Tímto postupem se docílí zvýraznění kontrastu v obraze a zároveň se odstraní nekonstantní osvětlení.[12]

Toho se docílí, tak že během první fáze *tophat* transformace, kdy je proveden *opening*, jsou odstraněny objekty, přičemž zůstává pouze nekonstantní pozadí. K dosažení tohoto efektu je zapotřebí dostatečně velký strukturální prvek, který zajistí, že objekty budou odstraněny a zachováno bude pouze pozadí s nekonstantním osvětlením. Ve druhé fázi funkce `imtophat` provede odečtení otevřeného obrazu od originálního, což má za následek eliminaci nekonstantního osvětlení.[3]

V *MATLABu* je použití *tophat* transformace provedeno následovně:

```
se = strel('disk',21);
top = imtophat(grayscaleImage, se);
```

Obrázek 2.21: Binarizovaný obrázek

Obrázek 2.22: Obrázek po tophat transformaci

Na pravém obrázku 2.21 můžeme vidět, jak jsme pomocí *tophat* transformace o prvku s poloměrem 21, ten je v porovnání s předchozími prvky mnohem větší, jelikož chceme odstranit jen větší nekonzistence, úspěšně odstranili nekonstantní pozadí z levého obrázku, ve vyznačené oblasti obrázku 2.22 vidíme, že znaky, které se nacházejí v transformací ztmaveném pozadí se také ztmavily.

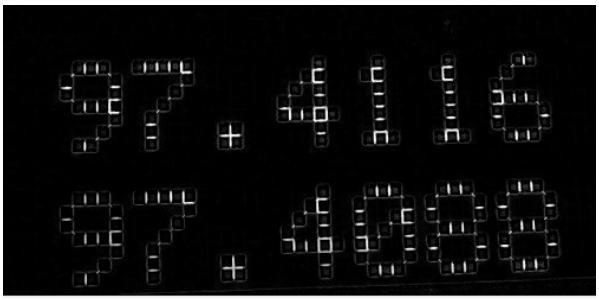
2.4.7 Bottomhat transformace

Bottomhat transformace je operace, která kombinuje *closing* s původním obrazem, což vede k odstranění pozadí a zvýraznění jemných detailů a malých objektů v obraze. Tato transformace je definována jako rozdíl mezi původním obrazem a obrazem, který byl uzavřen pomocí specifikované masky. Výsledkem *bottomhat* transformace je obraz, který ukazuje, co bylo v původním obraze ztraceno při operaci *closing*, což často zahrnuje jemné detaily a malé objekty. *Bottomhat* transformace je užitečná pro zvýraznění objektů v obraze, které jsou menší nebo méně kontrastní ve srovnání s okolními objekty nebo pozadím. Typické aplikace zahrnují detekci objektů v obrazech s nejednotným osvětlením, odstranění šumu nebo zvýraznění textury. Tato transformace izoluje pixely, které jsou tmavší než pixely v jejich okolí ve specifikovaném prostředí uvnitř masky, a je proto používána

pro detekci nízkých intenzit v obraze. V *MATLABu* je pro provedení *bottomhat* transformace k dispozici funkce `imbothat`, která bere jako vstup původní obraz a strukturální prvek pro operaci *closing* a její provedení můžeme vidět na níže uvedeném kódu.[3][13]

```
se = strel('disk',4);
bot = imbothat(greyscaleImage, se);
```

Obrázek 2.23: Binarizovaný obrázek


Obrázek 2.24: Obrázek po operaci bottomhat

Na pravém obrázku 2.24 můžeme vidět, jak po použití *bottomhat* transformace z levého obrázku 2.23 zbyly pouze malé detaily mezi znaky, které se odečtením od původního obrazu změnily na objekty. Takto bychom následně mohli oba obrázky spojit a vyplnit tak mezery mezi jednotlivými znaky.

3 Cíle řešení

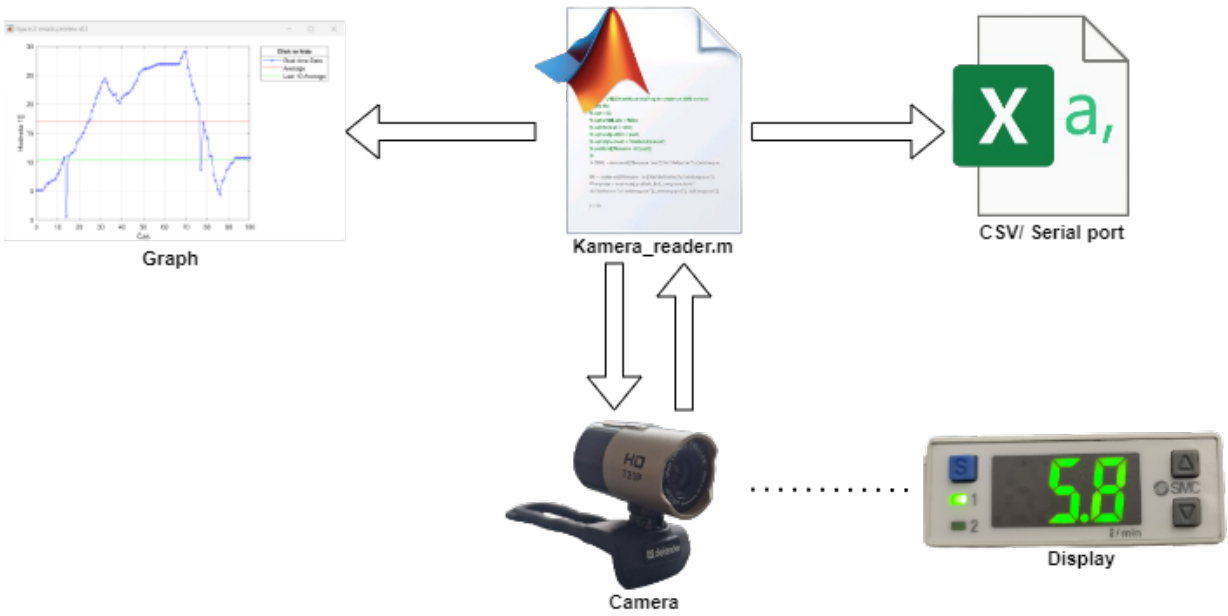
3.1 Vytvoření aplikace pro odečet čísel

Výsledkem práce budet uživatelské prostředí pomocí kterého si uživatel bude moct vybrat oblast displeje, kterou bude chtít snímat a po nastavení parametrů a spuštění se budou zaznamenávat přečtená data z daného displeje spolu s jejich časovou informací do *CSV* souboru, popř. na sériový port.

Po zapnutí aplikace se zobrazí dvě okna, v jednom bude zobrazován náhled kamerou, v němž si uživatel vybere snímanou oblast a na horní liště se bude nacházet nastavení parametrů. V druhém okně se v průběhu měření budou zobrazovat přečtené hodnoty v grafu. Parametry, které si uživatel bude moct nastavit budou výběr kamery, její rozlišení, periodu snímání displeje, typ snímaného displeje a parametry jednotlivých funkcí pro zpracování obrazu. Také bude možné si zvolit testovací režim, kde se v náhledu zobrazí zpracovaný obraz pomocí, kterého je možné ladit parametry funkcí.

V první řadě budeme potřebovat kameru pro snímání displejů, nainstalovat potřebné balíčky pro *MATLAB*, uvedené v kapitole 3.1 a dále vytvořit funkce pro jednotlivé kroky procesu. Program budeme programovat v objektech a následně jednotlivé funkce zakomponujeme to uživatelského prostředí. Z *MATLAB* kódu následně vytvoříme exe aplikaci, která bude se bude dát stáhnout a spustit na PC.

Jednoduché schéma automatického odečtu čísel můžeme vidět na obrázku 3.1.



Obrázek 3.1: Schéma automatického odečtu čísel

4 Realizace

V této kapitole probereme vypracování jednotlivých bodů zadání a jejich následnou aplikaci.

Abychom mohli používat určité, pro tuto práci stěžejní, funkce potřebujeme mít stažené rozšiřující balíčky pro *MATLAB*, a to konkrétně následující:

- **Image Processing Toolbox**
  Poskytuje set funkcí a nástrojů vytvořených konkrétně pro úpravu obrázků, jako například funkce pro zvýraznění, segmentaci, extrakci objektů, morfologické operace a další.

- **Image Acquisition Toolbox**
  Balíček se zaměřuje na snímání obrázků a videa z kamer a dalších snímacích zařízení. Poskytuje funkce k propojení, komunikaci a pořizování snímků v reálném čase s těmito zařízeními.

- **Computer Vision Toolbox**
  Sada funkcí navržená pro Computer Vision, nachází se v něm funkce pro rozeznání objektů a jejich následné zpracování, také je zde funkce pro rozeznání znaků v obrazu „OCR“, kterou budeme používat.

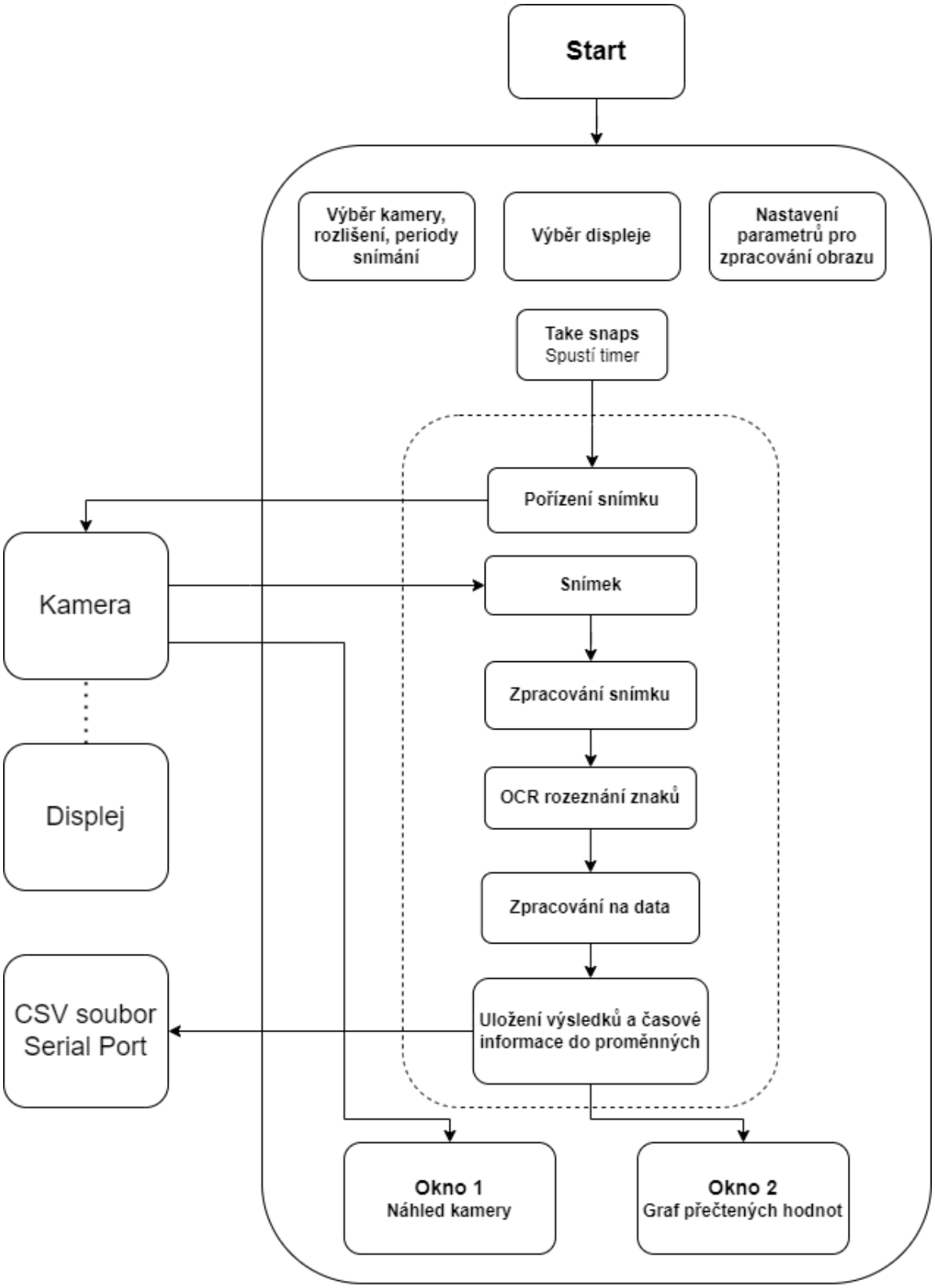
- ***MATLAB* Support Package for USB Webcams**
  Balíček pro USB webkamery sloužící k pořízení realtime obrazů z jakékoliv podporované USB videokamery do*MATLABu*.

- **Image Acquisition Toolbox Support Package for OS Generic Video Interface**
  Podpůrný balíček pro *Image Acquisition Toolbox*, který umožňuje získávat obrázky a video z zařízení pro záznam videa pomocí rozhraní *DirectShow* (pro Windows) přímo do *MATLABu*.

V realizační části jsme se zaměřili na zpracování obrazu a jeho čtení pomocí *MATLAB* funkce. Nejdříve, než jsme vytvářeli uživatelské prostředí, vytvořili jsme si program, ve kterém jsme upravovali a snažili se rozeznávat již pořízené obrázky, uložené ve složce, z které jsme je imporovali do *MATLABu*.

Důvod proč tento proces úpravy obrázků vůbec provádíme je funkce *OCR*, do které budeme upravený obrázek vkládat pro rozeznání znaků.

Na následující stránce můžeme vidět schéma celého procesu automatického odečtu čísel.



Obrázek 4.1: Schéma automatického odečtu čísel

4.1 Snímání displeje

K tomu abychom mohli snímat displeje měřících zařízení budeme využívat různých USB kamer připojených k počítači, proto potřebujeme uživateli umožnit výběr a nastavení kamery v *GUI*.

Pro zobrazení všech dostupných webkamer použijeme funkci `webcamlist` a funkce `listdlg` nám umožní výběr té požadované, jelikož různé kamery jsou schopné pořizovat obraz v různém rozlišení, musíme také nastavit rozlišení pomocí přípony `AvailableResolutions` a znovu `listdlg` pro výběr.

```
selectedCamera = listdlg('PromptString', 'Select a camera:',...
'ListString', webcamList, 'SelectionMode', 'single');


selectedRes = listdlg('PromptString', 'Select a resolution:',...
'ListString',res, 'SelectionMode', 'single');
```

Abychom mohli s kamerou v programu pracovat musíme v našem kódu vytvořit objekt pomocí něhož ke kameře budeme přistupovat, k tomu musíme mít vybranou kameru, rozlišení a s použitím funkce `videoinput` vytvoříme objekt, který bude danou kameru reprezentovat.

```
this.hCameraObj = videoinput('winvideo', this.hCam,...
"MJPG_"+this.hResolution);
```

Pro snímání displeje a určení oblasti pro zpracování na data musíme nejprve vidět, to stejné co vidí zařízení, které bude snímky pořizovat, proto vytvoříme náhled kamery. K tomu, aby bylo možné v reálném čase zobrazovat kameru v *GUI* použijeme funkci `preview(this.hCameraObj,im)` , kde `im` je matice o velikosti rozlišení, ve kterém kamera pracuje. V tomto náhledu bude umožněno označit oblast, ve které se nachází měřené hodnoty a následně pozorovat průběh měření.

Samotné označení bude umožněno vytvořením obdélníku přetažením kurzoru přes náhledovou oblast a v případě oblastí dvou jednoduše vytvoříme obdélníky dva.

```
ROI1 = drawrectangle();
this.hROI = ROI1.Position;
```

4.1.1 Pořízení snímku pro zpracování

Dále vytvoříme funkci pro pořizování snímků s požadovanou periodou v reálném čase. Samotné pořízení snímku kamerou obstaráme funkcí `getsnapshot`, jejímž vstupem je námi vytvořený objekt reprezentující kameru. Následně snímek ořízneme na předem zvolenou oblast (*ROI*) využitím příkazu `imcrop`, pokud jsme zvolili možnost pro oblasti dvě budeme mít ostřižené snímky dva. Tento snímek, popřípadě snímky poté uložíme do proměnné a následně budeme číst funkcí pro zpracování obrazu.

```
snapshot = getsnapshot(this.hCameraObj);
croppedImage1 = imcrop(snapshot, this.hROI(1, :));
```

4.2 Úprava snímků

V této kapitole vytvoříme proces úpravy snímků, který bude aplikovatelný při snímání v realném čase. Cílem sekvence funkcí je upravit obrázek do binární čitelné formy pro *OCR*.

Funkce *OCR* dosahuje nejlepších výsledků v případě, že jsou znaky umístěny na jednotném pozadí a mají formát černého textu nabílém pozadí.

V první časti procesu jsme se pouze zaměřili na úpravu obrázků displejů, které nám byly poskytnuty a na nichž jsme otestovali, zda vůbec bude možné data z těchto displejů číst, což se nám podařilo.

Nejjednodušším krokem úpravy obrázků je vymezení oblasti ve které se hledané hodnoty na displeji vyskytují, nazývané *ROI* (region of interest), ten jsme učinili již předchozí kapitole.

Pro další úpravy budeme potřebovat funkce pro úpravu obrazu probrané v rešeršní části 2.

Poté, co dostaneme obrázek v *RGB* formátu do *MATLABu*, se snímek skládá z tří matic interpretujících jednotlivé barvy *RGB*. V prvním kroku převedeme *RGB* na *grayscale* formát pomocí níže uvedeného kódu, ten budeme následně moci upravovat mimojiné pomocí morfologických operací.

```
grayImage = rgb2gray(rgbImage)
```

Důležité pro úpravu obrázků je podotknout, že záleží, zda máme světlé znaky na tmavém pozadí, což je většina našich případů nebo tmavé znaky na světlém pozadí. Z rešerše morfologických operací viz kapitola 2.1, víme, že tyto operace správně fungují pro světlé objekty na tmavém pozadí, případě tmavých objektů stačí provést inverzi celého obrázku, kdy oděčteme hodnotu každého pixelu od 255 nebo pomocí funkce `imcomplement` viz níže uvedený kód.

```
inverseGrayImage = uint8(255) - grayImage
inverseGrayImage = imcomplement(grayImage)
```

Každopádně, jak jsme výše zmínili funkce *OCR* nejlépe funguje pro černé objekty na bílém pozadí, proto na konci úpravy při binarizaci převrátíme hodnoty pixelů.

Pokud máme obrázek ve správném formátu, můžeme přejít k aplikaci následujících operací.

Jako první chceme, co nejvíce zvýraznit objekty do světlé a pozadí do tmavé šedi, čehož se docílí pomocí funkce `imadjust` jejímž vstupem je pouze *grayscale* obrázek.

```
adjustedImage = imadjust(grayImage)
```

Jak již bylo zmíněno je potřeba docílit jednotného pozadí a jelikož se v obrázku mohou vyskytovat různé odlesky a šum měli bychom se je pokusit odstranit. K tomu prvně použijeme funkci `imfilter` s průměrovacím filtrem.

```
averageFilter = fspecial('average',10);
filteredImage = imfilter(adjustedImage, averageFilter,'replicate');
```

Velikost filtru volíme podle toho, jak velké okolí chceme filtrovat, čím větší tím více rozmazaný bude obrázek působit.

Dalším krokem k jednotnému pozadí je *tophat* transformace, viz kapitola 2.4.6, pomocí níž, na rozdíl od filtrace, která průměruje pouze blízké okolí, dosáhneme jednotného pozadí napříč celým obrázkem.

Pro funkci `imtophat` budeme muset vytvořit masku pomocí které se bude transformace provádět, v našem případě použijeme funkci `strel`, tvar strukturálního prvku `disk` a velikost se bude lišit pro různé displeje, avšak v porovnání s ostatními strukturálními prvky bude značně větší, v rozmezí mezi 15 až 25.

```
transformedImage = imtophat(filteredImage, strel('disk',21));
```

V této fázi bychom měli mít *grayscale* obrázek bez šumu, s jednotným pozadím a značně rozeznatelné znaky od pozadí.

Dalším problémem, s nímž si budeme muset poradit je fakt, že většina displejů zobrazuje znaky v segmentech a znaky tedy nejsou spojené ve všech jejich místech. Je tedy potřeba tyto mezery odstranit a uzavřít tak znaky. Na tuto operaci použijeme morfologickou operaci *closing*, viz kapitola 2.4.5, jejímž vstupem je transformovaný obrázek se strukturálním prvkem určujícím, jak velké mezery ve znaku se uzavřou, ve většině našich případů v rozmezí 2 až 7, podle typu displeje.

```
closedImage = imclose(transformedImage, strel('disk',3));
```

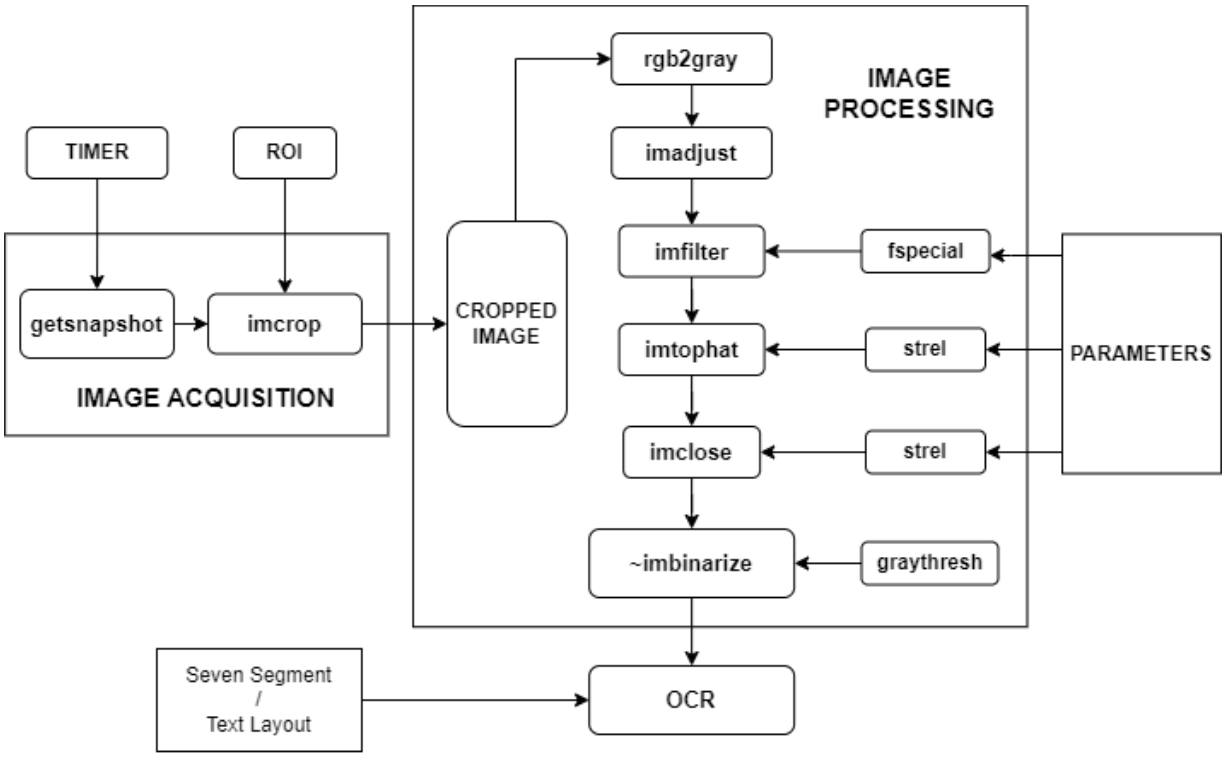
Posledním krokem v úpravě obrázku pro *OCR* je převedení do binární formy, kapitola 2.2.4, a inverze znaků z bílé na černou, kdy v ideálním případě dostaneme čitelné černé znaky na bílém pozadí. Pro binarizaci použijeme funkci `imbinarize`, do níž jako vstupy použijeme uzavřený obrázek a hodnotu prahu pro binarizaci určené pomocí funkce `graythresh`, která na základě intenzit pixelů v obrázku určí hodnotu prahu viz 2.2.4.

```
threshold = graythresh(close);
binarizedImage = ~imbinarize(closedImage,threshold);
```

Znak ~před funkcí imbinarize slouží právě k převrácení hodnoty všech pixelů z bílé na černou a naopak, čímž dostaneme černé znaky na bílém pozadí.

Tím bychom měli mít upravený obrázek ve formě binární matice, připravený pro vložení do funkce *OCR* pro rozeznání znaků. Pomocí tohoto procesu budeme nastavovat parametry pro různé druhy displejů a následně bude tento proces proveden v reálném čase pokaždé když program pořídí kamerou snímek.

Schéma procesu pořízení a úpravy snímků na znaky můžeme vidět na obrázku 4.2.



Obrázek 4.2: Struktura kódu pořízení a úpravy obrázku na znaky

4.3 Rozpoznávání hodnot a úprava na data

4.3.1 Optical Character Recognition

Funkce, která je naprosto stěžejní pro řešení problému, kterým se bakalářská práce zabývá je již mnohokrát zmíněná funkce *OCR* pro rozeznávání znaků v obrázku.

Správné nastavení této funkce nám umožní číst data z displejů v reálném čase, ale jak již bylo zmíněno, pro dobrou úspěšnost rozeznání znaků musíme nejprve upravit původní obrázek do formy černých znaků na bílém pozadí s co možná nejvyšší čitelností. Do funkce *OCR* dáváme jako vstup upravený snímek, ale také je nutné další nastavení.[14][13]

Nastavení *OCR* spočívá v první řadě rozlišení typu displeje, jestli se například jedná o sedmi segmentový displej nebo displej, kde je segmentů více a znaky pak mohou mít odlišný font. Dále také nastavení rozložení, ve kterém se znaky v obrázku nachází pomocí `'TextLayout'`. Jelikož se na displejích měřících přístrojů vyskytují různé druhy fontů, tak budeme v našich případech používat, jak zmíněný sedmi segmentový mód, tak i mód textový.

V případě sedmi segmentového displeje musíme v *MATLABu* nastavit model jako `seven-segment`, ukázáno níže.

```
results = ocr(binarizedImage,Model='seven-segment');
```

Pro textový font, nastavíme rozložení znaků do bloku a můžeme také nastavit, set

znaků, které bude funkce v obrázku hledat pomocí `CharacterSet`, ukázáno níže.

```
results = ocr(binarizedImage,'TextLayout','Block');
results = ocr(binarizedImage,'TextLayout','Block',...
'CharacterSet','0123456789.+-');
```

Funkce nám kromě přečteného textu vrátí také přesnost přečtení jednotlivých znaků a jejich umístění v obrázku.

4.3.2 Zpracování výsledků

Z funkce *OCR* dostaneme přečtený set znaků v různých formátech v objektu *ocrText* uložené v naší proměnné, např.`results`. Pro nás budou užitečné `results.Text`, kde jsou ve formátu *char* uloženy všechny přečtené znaky na jednom řádku, ovšem pokud máme znaky ve více řádcích, např. dvě čísla pod sebou, funkce nám do `results.TextLines` uloží znaky na jednotlivých řádcích.

```
res = results.TextLines
stringValue = res{1};
```

Nyní potřebujeme převést přečtené znaky z textového do číselného formátu. Jelikož v mnoha případech *OCR* funkce přečte mezeru mezi znaky, protože má displej segmenty dále od sebe, musíme se těchto mezer zbavit. To uděláme jednoduše, tak že pokud jsou v textu mezery, přečtený text rozložíme na více částí, kde jednotlivé části oddělují přečtené mezery a následně části zbavené mezer spojíme zpět. Následně už pouze převedeme číslo z formátu *string* do formátu *double*.

```
parts = strsplit(stringValue, ' ');
if numel(parts) == 1
   numberValue = str2double(parts{1});
else
   numberValue = str2double(strcat(parts{:}));
end
```

4.3.3 Záznam časové informace

Jelikož měření provádíme v reálném čase, je vhodné ke každé přečtené hodnotě přidat i časovou informaci za které byla pořízena a ukládat je zároveň. Pro zaznamenání časové informace můžeme použít funkci `datetime`, ve které nastavíme, jaký čas a formát zapsání chceme.

Jelikož chceme znát přený čas ve kterém se informace z kamery zpracovává použijeme nastavení `now` a bude vhodné rozdělit čas a datum do dvou proměnných, kde časová proměnná bude ve formě „hodina:minuta:sekunda“ a datum „den-měsíc-rok“.

```
this.hTime(this.hi, 1) = datetime('now','Format', 'HH:mm:ss');
this.hDate(this.hi, 1) = datetime('now','Format','d-MMM-y');
```

4.4 Odeslání a zpracování získaných dat

Data budeme ukládat do souboru formátu *CSV*, který vytvoříme vždy před začátkem měření. Uvnitř budou data rozděleny podle volby snímání, do jednoho nebo dvou sloupců s naměřenými hodnotami a dvou sloupců s časovou hodnotou, kde v jednom bude čas a v druhém datum. Soubor bude pojmenován ve formě `Data_2023-12-12_19-04-44.csv` podle datumu a času měření.

```
filename = ['Data_', datestr(datetime('now'),'yyyy-mm-dd_HH-MM-SS'),'.csv'];
```

Data se budou ukládat při každé periodě zpracování snímku tak aby nedošlo k jejich ztrátě a přepsání v existujícím souboru. Soubor při každé periodě otevřeme zapíšeme hodnoty a zavřeme, možnost přepsání zamezíme příponou 'a' při otevírání souboru.

```
file = fopen(this.hFileName, 'a');
fprintf(file,'%g,%s,%s\n',this.hResult(this.hi,1),...
char(this.hTime(this.hi,1)), char(this.hDate(this.hi,1)));
fclose(file);
```

Data také budeme zapisovat na serialport z kterého je možné poslaná data číst například v jiném programu. Pomocí funkce serialportlist uložíme port do proměnné a tu následně budeme používat pro zapisování.

```
serialPort = serialportlist;
s = serial(serialPort(1));
fopen(s);
fprintf(s, '%g,%s,%s\n', this.hResult(this.hi,1),...
char(this.hTime(this.hi,1)), char(this.hDate(this.hi,1)));
fclose(s);
```

4.5 Měření konkrétních displejů

Jelikož v realitě existuje spousta druhů displejů, s různým typem zobrazení a zápisu měřených, hodnot, tak se v této kapitole zaměříme na aplikaci předchozích kapitol 4.2 a 4.3 na několik konkrétních displejů. Bude probrán postup řešení a problémy na které jsme narazili. U jednotlivých displejů nebude zmíněna základní úprava obrazu jako převod do *grayscale*, zvýraznění před použitím morfologických operací a následná binarizace, jelikož jsou u všech stejné a byly popsány v předchozí kapitole 4.2.

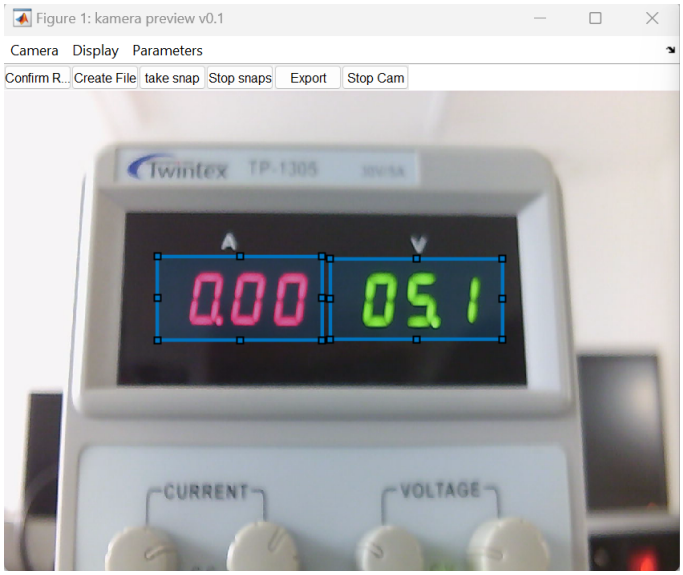
4.5.1 Měření zdroje Twintex

Obrázek 4.3: Zdroj Twintex

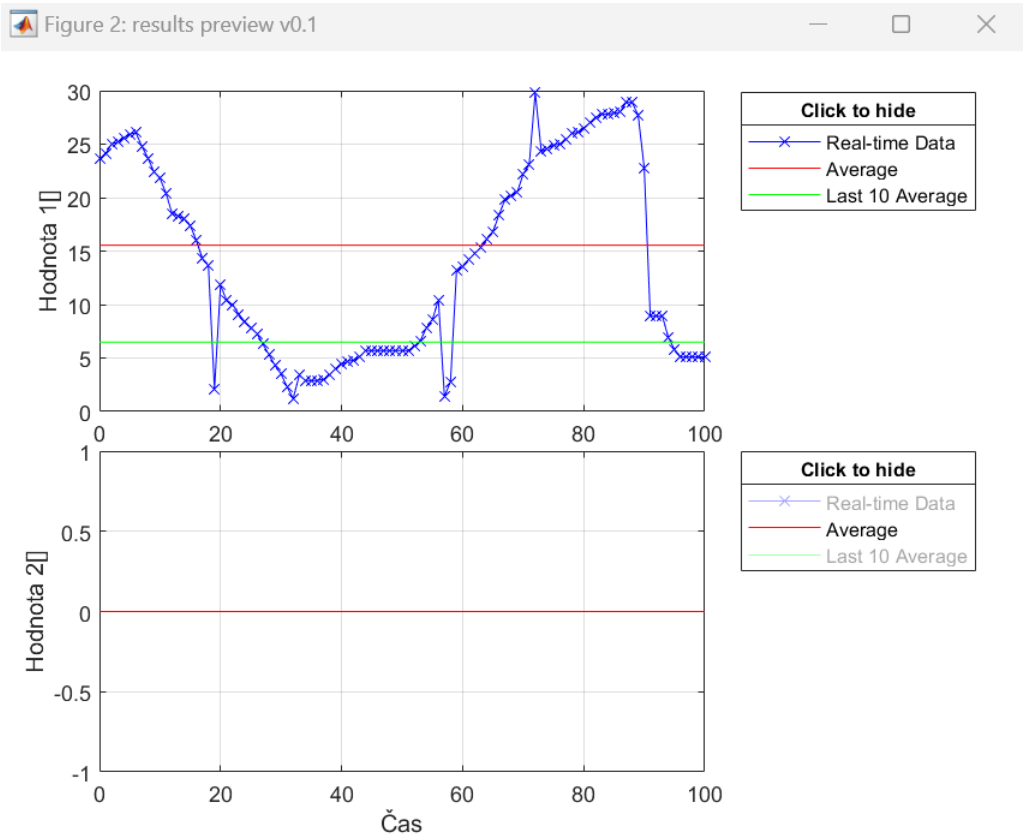
Jedná se o sedmi segmentový displej, který zobrazuje světlé hodnoty na černém pozadí viz obrázek 4.3. Proto jsme použili celý postup úpravy obrazu popsaný v kapitole 4.2 a pro *OCR* jsme použili sedmi segmentový mód viz 4.3. V tomto případě nastavujeme tři parametry, z měření nám hodnoty parametrů nejvíce vyhovující pro tento displej vycházejí 7 pro filtraci, 21 pro *tophat* transformaci a 3 pro *closing*. Ukázku provedení úpravy snímku 4.4 můžeme vidět na obrázku 4.5, kde vidíme, že desetinná tečka není ideálně odsazena od čísla, avšak mód seven segment s touto problematikou na rozdíl od normálního problém nemá a tečku přečte. Na obrázku 4.6 můžeme vidět náhled průběhu měření a na 4.7 graf přečtených hodnot. Na grafu můžeme vidět nespojitá místa, kdy *OCR* udělá chybu, v tomto konkrétním případě má občas problém přečíst jedničku na prvním místě, což se ale vylepšilo měřením z lehce větší vzdálenosti.

Obrázek 4.4: Barevný snímek po oříznutí

Obrázek 4.5: Upravený snímek



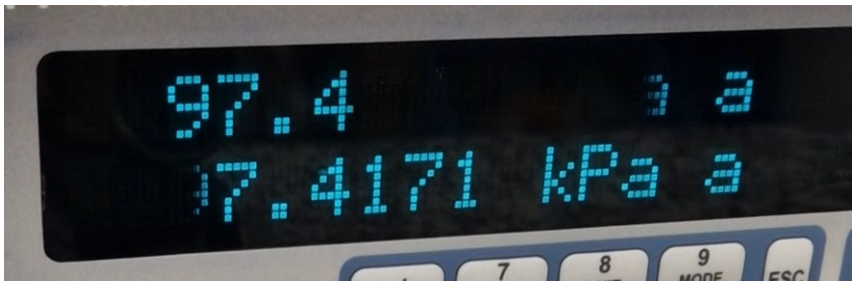
Obrázek 4.6: Náhled měření přístroje Twintex v GUI



Obrázek 4.7: Průběh měření přístroje Twintex v grafu

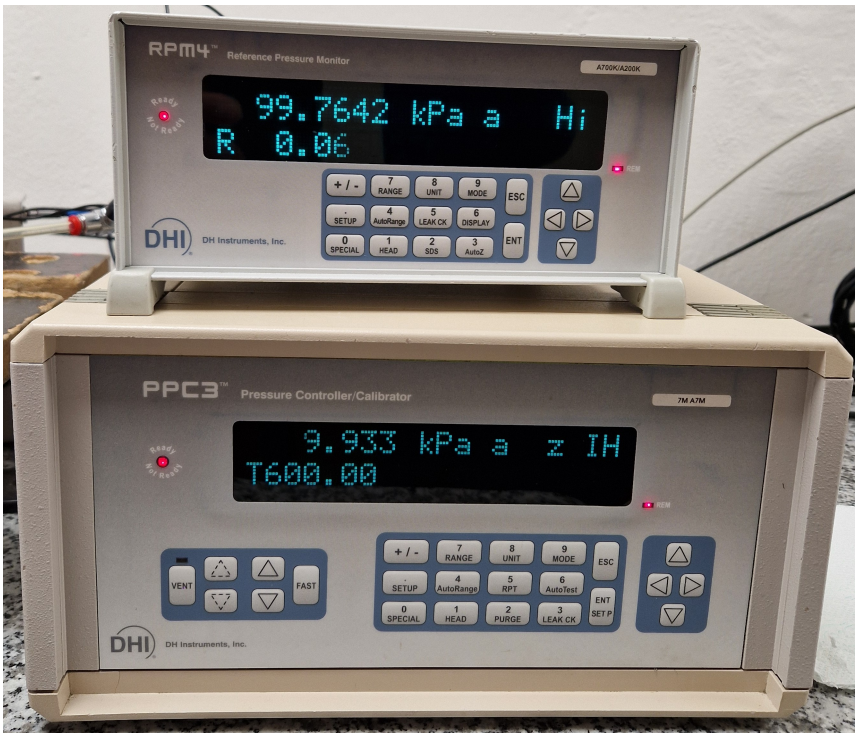
4.5.2 LCD displeje

Z důvodu LED osvětlení v místnosti dochází k blikání segmentů displejů viz 4.8, což je velký problém, pokud chceme pořizovat snímky těchto displejů. Tento problém vzniká u LCD displejů kvůli nesynchronní frekvenci osvícení od LED a obnovovací frekvenci displeje. Tento problém jsme zkusili vyřešit přisvícením displeje jiným světlem, avšak neúspěšně. Došli jsme k závěru, že nejjednodušším řešením je zastínit displej stínítkem nebo v průběhu měření co nejvíce omezit LED osvícení.



Obrázek 4.8: Ukázka vlivu LED osvětlení na LCD displej

4.5.3 Měření přistrojů RPM4 a PPC3



Obrázek 4.9: Měřící přistroje RPM4 a PPC3

Přístroje na obrázku 4.9 mají LCD displej, u kterého se vyskytuje problém zmíněný v předchozí kapitole 4.5.2 s blikáním způsobený LED osvětlením. Je tedy nutné stínění nebo snížení osvícení v místnosti.

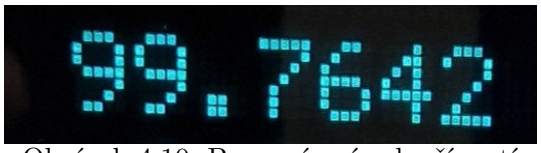
Displej je více segmentový se světlými znaky a černým pozadím, tedy pro *OCR* pou-

žijeme normální textový mód. V tomto případě jsme použili funkce filtrace, *tophat* transformace, která je zde stěžejní kvůli lesklému povrchu displeje na němž se často vyskytují odlesky, címž se pak na snímku vyskytuje nekonstantní pozadí. Také jsme použili *closing* na uzavření mezer segmentů. Z testování nám hodnoty parametrů nejvíce vyhovující pro tento displej vycházejí 11 pro filtraci, 21 pro *tophat* transformaci a 3 pro *closing*.

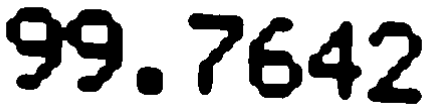
U tohoto displeje jsme narazili na problém s občasným přečtením čísla 0 jako 8 kvůli tomu, že displej zobrazuje 0 se šikmým přeškrtnutím, jak můžeme vidět na obrázku 4.13. Tento problém se po menší úpravě parametrů podařilo lehce minimalizovat, avšak ne úplně a je nutné to brát v potaz při měření a nastavování parametrů.

Nastavování parametrů také lehce ovlivňuje osvětlení v místnosti, jak bylo zmíněno u tohoto typu displeje je vhodné měření při omezení LED osvětlení, v tom případě je, ale nutné zmenšit parametr pro filtraci na 8, jelikož se displej v přítmí lehce „rozmaže“ sám. Po snížení parametru měření dále probíhalo bez problémů.

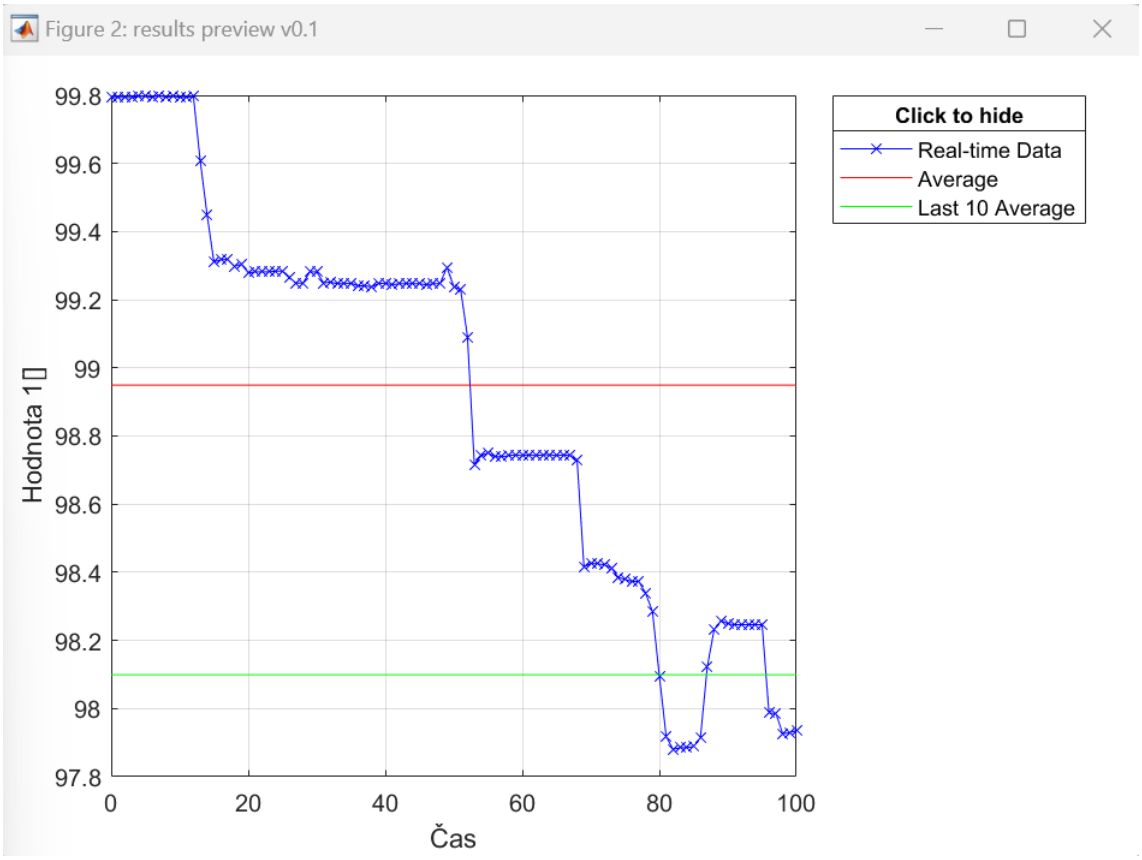
Na obrázku 4.11 můžeme vidět úpravu snímku 4.10, na obrázku 4.12 graf průběhu měření a na 4.13 náhled na měřený displej.



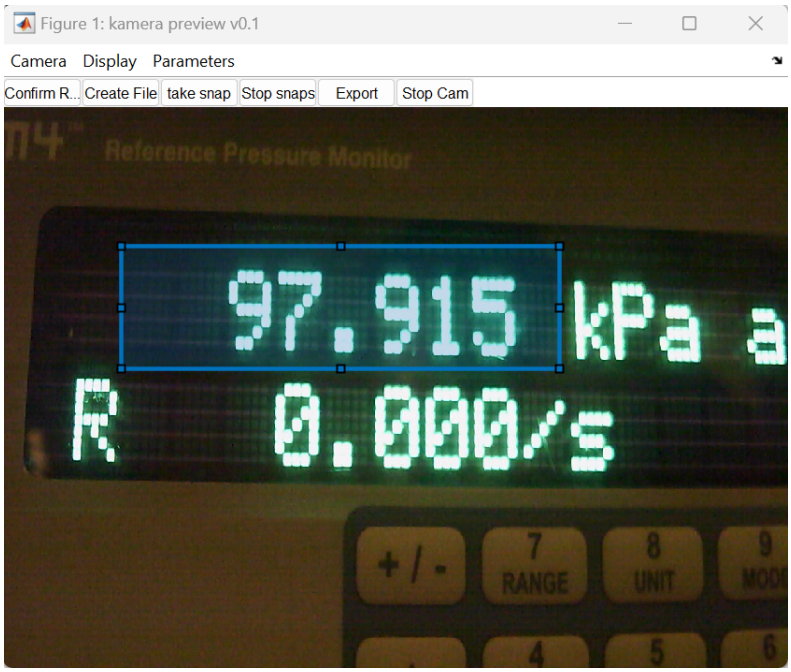
Obrázek 4.10: Barevný snímek oříznutí



Obrázek 4.11: Upravený snímek



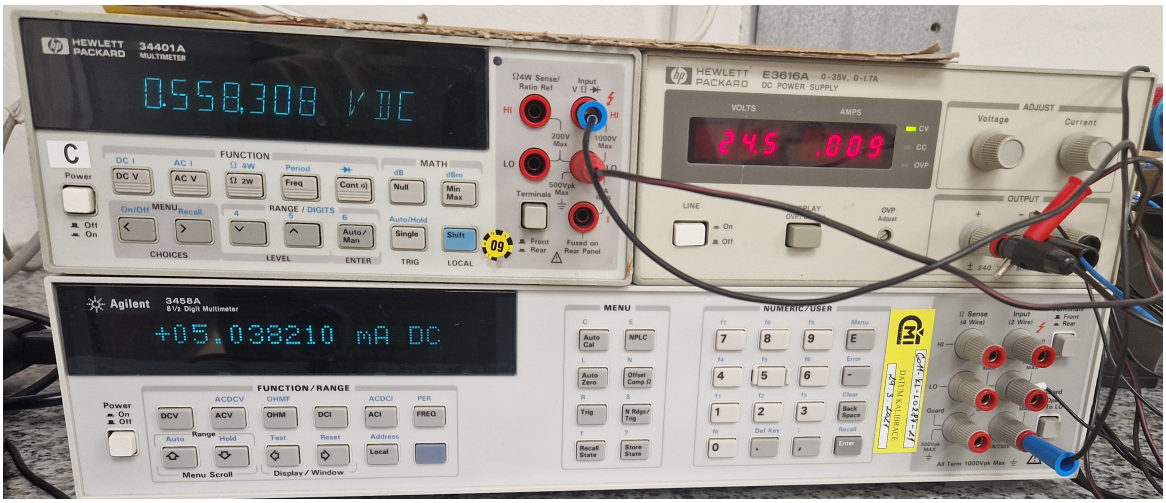
Obrázek 4.12: Průběh měření přístroje RPM4 v grafu



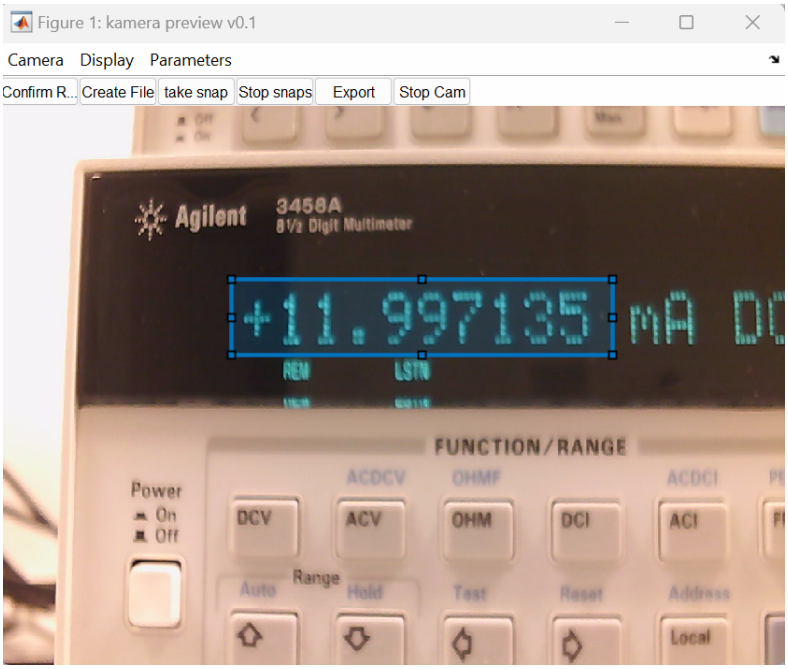
Obrázek 4.13: Náhled měření přístroje RPM4 v GUI

4.5.4 Měření přístroje Agilent

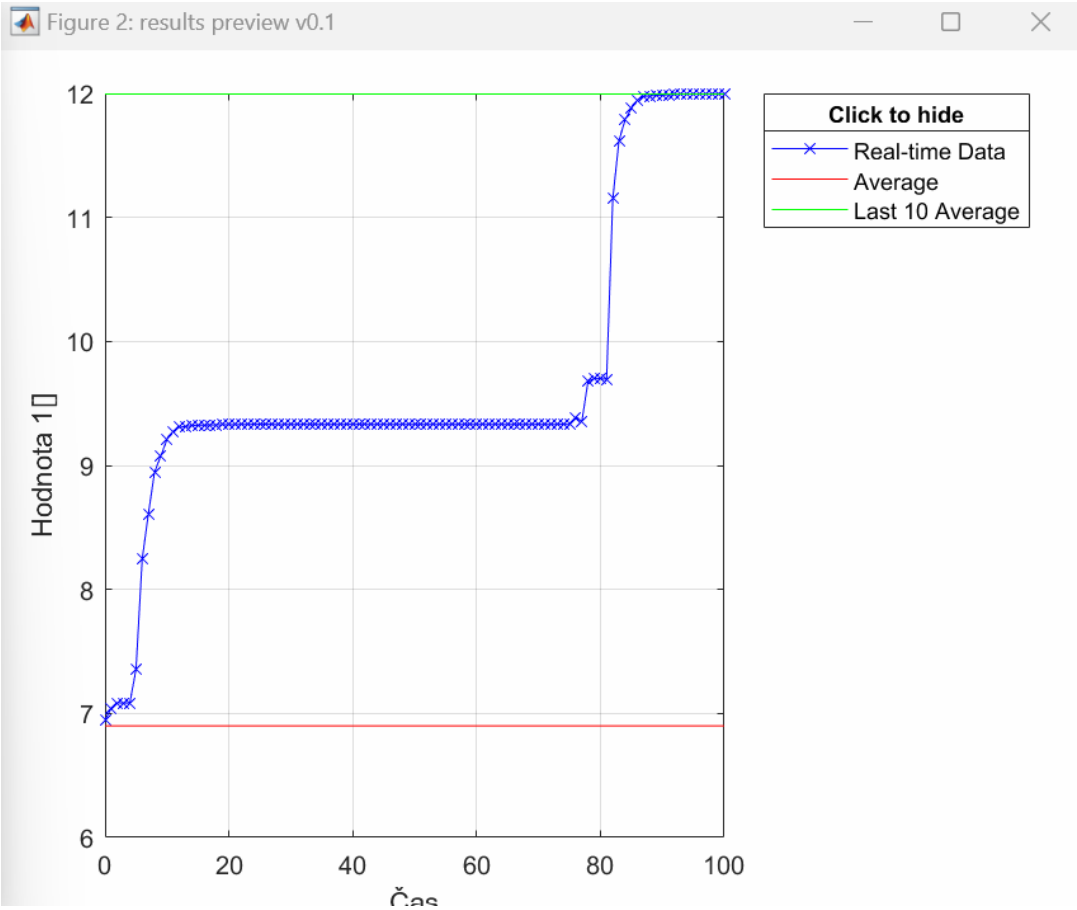
Přístroj Agilent můžeme vidět v dolní části obrázku 4.14. Z obrázku vidíme, že se jedná o více segmentový displej zobrazující světlé hodnoty na černém pozadí, takže pro *OCR* jsme použili klasický mód. Pro úpravu snímku byla použit postup popsaný v kapitole 4.2, který zde neměl žádné problémy. Z měření nám hodnoty parametrů nejvíce vyhovující pro tento displej vycházejí 9 pro filtraci, 21 pro *tophat* transformaci a 3 pro *closing*. Úpravu snímku 4.17 můžeme vidět na obrázku 4.18. Průběh a náhled měření je možné vidět na obrázcích 4.15 a 4.16.



Obrázek 4.14: Měřící přístroje Agilent a HP

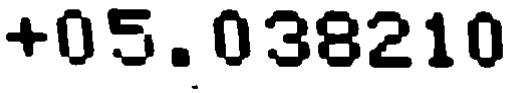


Obrázek 4.15: Náhled měření přístroje Agilent v GUI



Obrázek 4.16: Průběh měření přístroje Agilent v grafu

Obrázek 4.17: Barevný snímek oříznutí



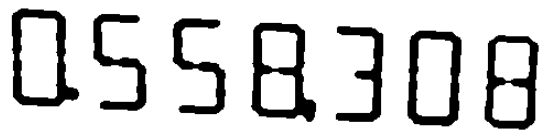
Obrázek 4.18: Upravený snímek

4.5.5 Měření přístroje HP

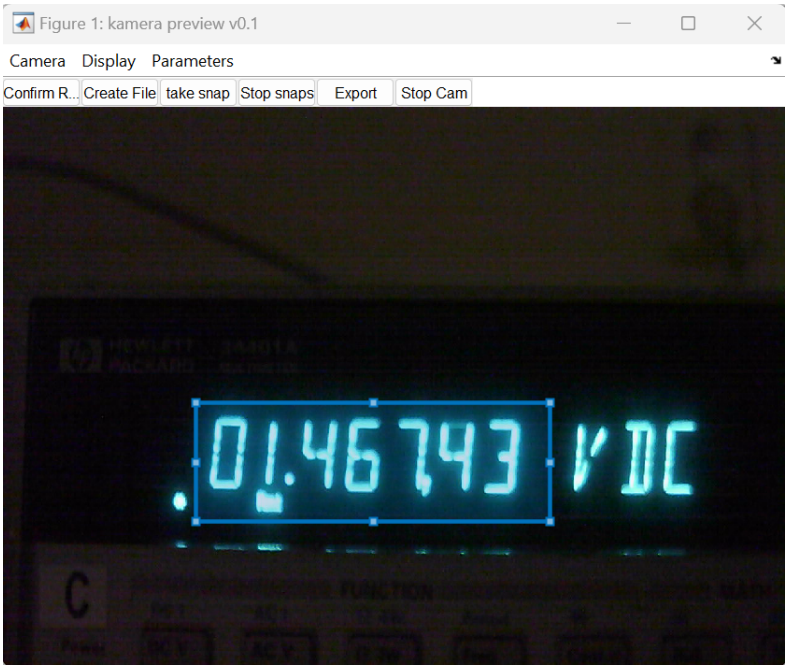
Přístroj HP můžeme vidět v levé horní části obrázku 4.14. Z obrázku vidíme, že se jedná o sedmi segmentový displej zobrazující světlé hodnoty na černém pozadí, pro úpravu snímků jsme použili postup viz kapitola 4.2 a pro *OCR* sedmi segmentový mód. Hodnoty parametrů, použitých operací po testování tohoto displeje, vycházejí 7 pro filtraci, 15 pro *tophat* transformaci a 5 pro closing.

U tohoto přístroje můžeme na obrázku 4.19 vidět, že má dvě tečky což je značný problém, jelikož funkce nerozliší, že se jedná o oddělení tisíců, ale přečte číslo s dvěmi desetinými tečkami, tím pádem jsme museli tisícinou čárku z přečteného textu odstranit a poté měření probíhalo správně. Úpravu snímku 4.19 můžeme vidět na obrázku 4.20, průběh měření v grafu a náhled na obrázcích 4.22 a 4.21.

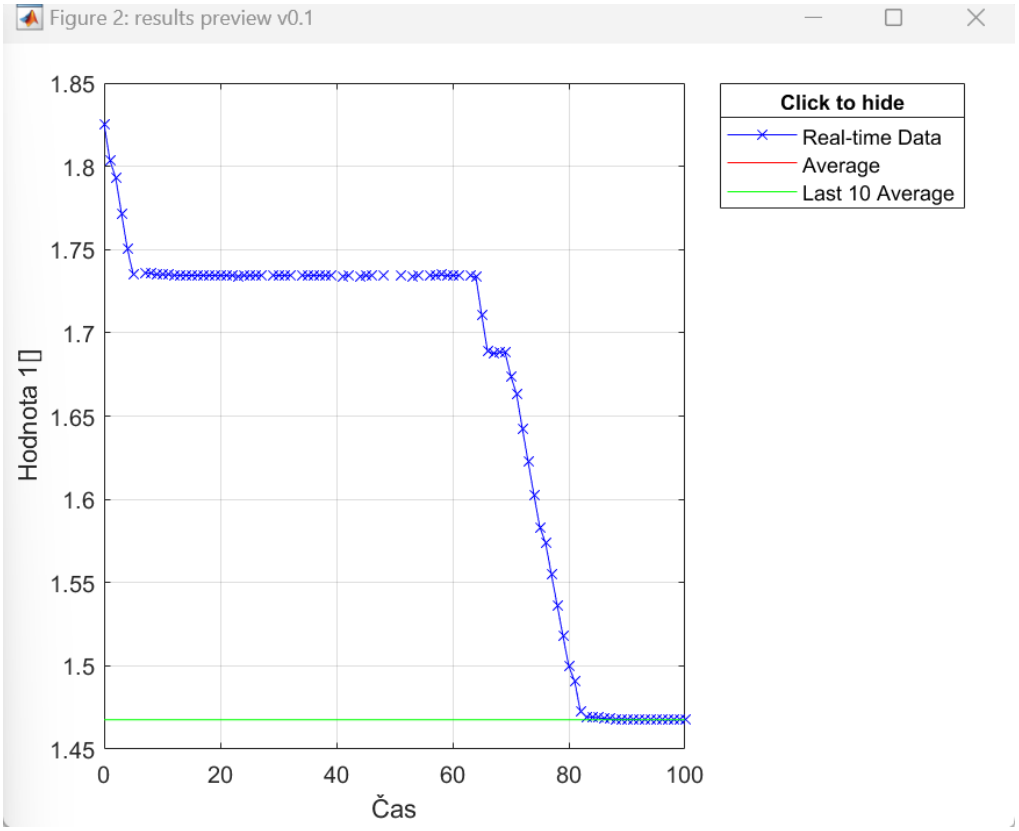
Obrázek 4.19: Barevný snímek po oříznutí



Obrázek 4.20: Upravený snímek



Obrázek 4.21: Náhled měření přístroje HP v GUI



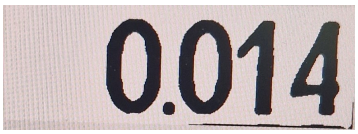
Obrázek 4.22: Průběh měření přístroje HP v grafu

4.5.6 Měření přístroje Bronkhorst

Přístroj můžeme vidět na obrázku 4.23. V tomto případě jsou naměřené hodnoty zobrazovány černě na výrazně světlém pozadí a vidíme, že se jedná o spojitý textový font. Z tohoto důvodu jsme nepotřebovali žádné funkce pro spojení znaků, ale pouze funkci `imfilter` na odstranění šumu, pro *OCR* jsme použili textový mód.

Jak vidíme na obrázku úprava obrázku 4.24 proběhla úspěšně, avšak při použití *OCR* v normálním módu nebyla rozeznána desetinná tečka ani při různých nastaveních, proto jsme pro tento displej v případě, že se v přečteném textu desetinná tečka nevyskytuje, přidali podělení tisícem, což by v tomto případě mělo dostačovat, jelikož se počet desetinných míst nemění. Následně už měření probíhalo správně.

Obrázek 4.23: Přístroj Bronkhorst



Obrázek 4.24: Upravený snímek

4.6 Tvorba uživatelského prostředí

Jelikož práci vytváříme pro jednoduché používání širší oblastí uživatelů, zaobalili jsme program do intuitivního uživatelského prostředí, pomocí kterého bude proces měření prováděn a kde bude možné nastavovat různé parametry pro jednotlivé požadavky.

V uživatelském prostředí jsme vytvořili tlačítka pro výběr kamery a jejího rozlišení z kamer právě připojených k počítači. Uživateli je také umožněno nastavení periody snímání kamerou, výběr počtu snímaných hodnot a možnost zvolení testovacího režimu, ve kterém se upravené snímky zobrazují v náhledu.

Tento režim je vhodný při nastavování parametrů úpravy snímků jednotlivých displejů, což je další prvek, který se v GUI nachází.

V programu si uživatel může vybrat z 6 možných typů displejů, pro které je následně umožněno již zmíněné nastavování parametrů. Jednotlivé displeje jsou popsány v kapitole 4.5 a také pak v následující kapitole 4.7, určené pro používání aplikace.

Program se skládá ze dvou oken, v hlavním okně je náhled kamery a tlačítka pro ovládání a nastavování programu, v druhém pak graf, v kterém se v reálném čase zobrazují přečtené hodnoty. V grafu kromě real time hodnot vidíme také celkový průměr a průměr posledních deseti hodnot, kdy všechny tři ze zmíněných grafů se dají skrýt pomocí kliknutí na jejich legendu v pravém horním rohu.

V hlavním okně se také nachází 5 tlačítek, pro ovládání průběhu měření, a to tlačítko pro potvrzení oblasti snímání, ta se pomocí myši označí po vybrání kamery v jejím náhledu v hlavním okně. Tlačítko pro vytvoření CSV souboru, do kterého se budou hodnoty ukládat. Následně jsou zde tlačítka pro započetí a pozastavení měření a tlačítko pro ukončení aplikace viz 4.35.

Aby nebylo nutné vlastnit *MATLAB* vytvořili jsme z GUI exe aplikaci stažitelnou na počítač bez *MATLABovské* licence. K vytvoření exe aplikace z *MATLABu* jsme použili balíček *Application Compiler*, v kterém jsme nastavili potřebné informace a vybrali *m.file* z kterého se aplikace vytvoří. K tomu abychom mohli pracovat s aplikací je nutné si stáhnou *MATLAB Runtime* pro verzi *2023b*, ve které byl program vytvořen.

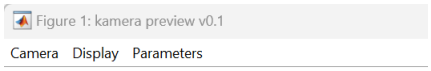
4.7 Postup měření

Tato kapitola slouží, jako návod k používání vytvořeného programu pro čtení dat z displejů. Bude popsán postup, jak s programem pracovat, včetně výběru kamery, rozlišení, typu displeje, také nastavení oblasti pro snímání a parametrů pro úpravu snímků.

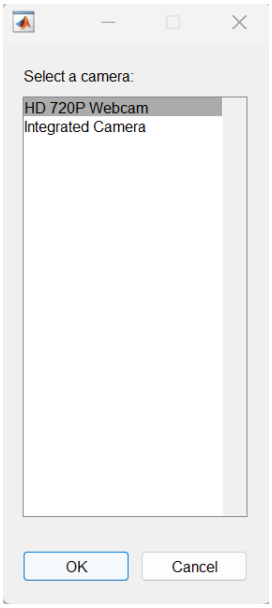
Po spuštění programu je nejprve nutné provést výběr kamery, kliknutím na **Camera** a **Select Camera** 4.25. V prvním okně se zobrazí všechny dostupné kamery 4.26, vybranou odklikneme tlačítkem OK. Dále vybereme rozlišení kamery 4.27, po kterém následuje výběr periody, kterou budeme displej snímat 4.28.

V dalším okně 4.29 je možnost výběru počtu snímaných hodnot, kdy možnost 1 je pro všechny displeje a 2 jen u některých, konkrétní displeje, pro které je možné snímat dvě hodnoty budou uvedeny dále.

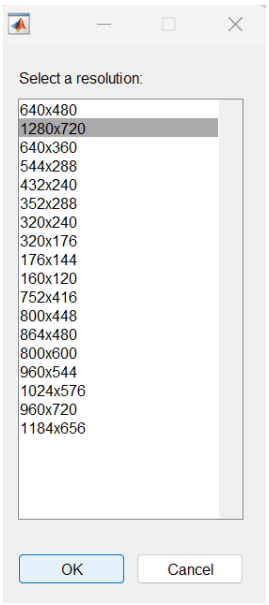
Jako poslední v tomto kroku je výběr módu 4.30, kde je možné vybrat mód pro testování vybráním 1, místo náhledu kamery se po spuštění měření budou zobrazovat upravené snímky pro *OCR*, pro normální měření vybereme 0.



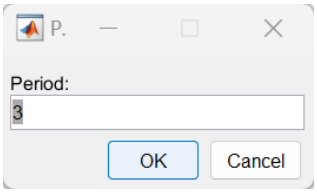
Obrázek 4.25: Náhled po zapnutí



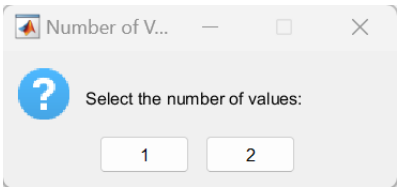
Obrázek 4.26: Výběr kamery



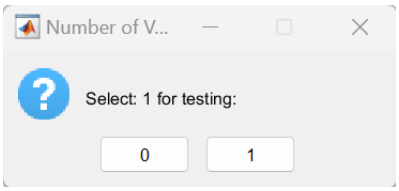
Obrázek 4.27: Výběr rozlišení



Obrázek 4.28: Výběr periody



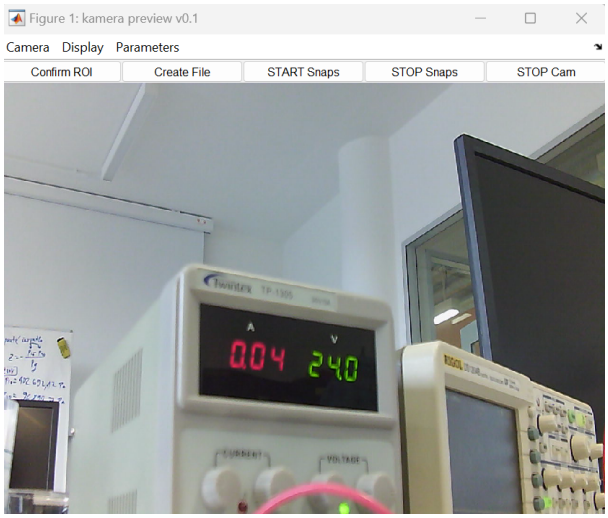
Obrázek 4.29: Výběr hodnot



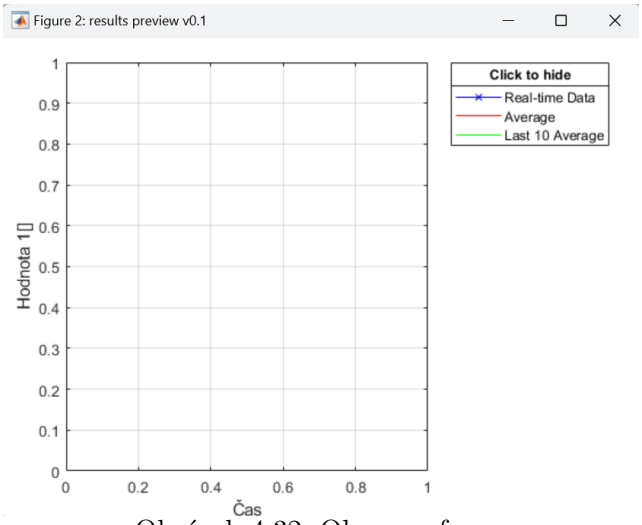
Obrázek 4.30: Výběr módu

Po dokončení prvního kroku se zobrazí náhled kamery v hlavním okně 4.31 a graf ve kterém se budou zobrazovat hodnoty v okně druhém 4.32, jednotlivé průběhy se dají vypnout kliknutím na legendu.

Ve druhém kroku musíme vybrat typ displeje, který budeme snímat. Ten vybereme kliknutím na **Display**, **Select Display** a kliknutím na zvolený typ 4.33. Jednotlivé typy displejů budou popsány dále.



Obrázek 4.31: Hlavní okno



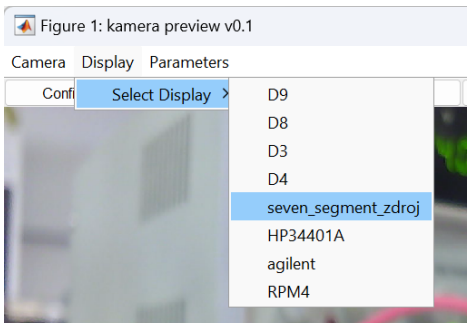
Obrázek 4.32: Okno grafu

Ve třetím kroku máme možnost nastavit parametry konkrétního displeje vybráním **Parameters** a **Set Parameters**, nejprve však vždy musí být zvolen displej, jelikož podle něj se zvolí původní parametry a jejich případná změna.

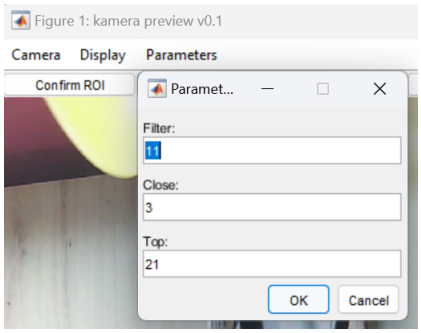
Parametry, které je možné nastavovat jsou parametry pro filtraci, *closing* a *tophat* transformaci, parametr pro *tophat* transformaci ve většině případů není nutné měnit, protože slouží pouze k odstranění odlesků a jiných nekonstantních elementů. Ve většině případů stačí změnit filtraci o jednotky nahoru pokud se segmenty čísel snímku nepodaří spojit nebo dolů pokud naopak bude snímek moc rozmazaný. *Closing* slouží k jemnému uzavírání znaků tedy podobně jako u filtrace, ale hodnoty *closingu* jsou ve většině případů nižší.

Ideální je spustit měření bez vytvoření složky a z grafu zjistit, zda jsou parametry optimální nebo ne. V případě, že nejsou, tak je vhodné měření zastavit a měnit parametry filtrace po jedné dolů, popř. nahoru, to stejné pro *closing*, ale ten by nemělo být nutné měnit více než o 2 a poté program spustit. Takto bychom se během chvíle měli dostat k optimálním parametrů, následně můžeme vytvořit složku a začít měřit.

Pokud by se ovšem nepodařilo dostat optimální parametry, další změny jsou vhodné v testovacím režimu, kde po spuštění uvidíme vliv změny na snímek. Nejčastějším důvodem změny parametrů může být změna osvětlení v místnosti, kdy je dobré snížit parametry pro filtraci a *closing* (uzavření segmentů), nebo změna vzdálenosti měření.



Obrázek 4.33: Výběr displeje



Obrázek 4.34: Nastavení parametrů

Vytvořili jsme několik setů programů pro různé druhy displejů, níže budou popsány, tak aby uživatel mohl správně zvolit typ pro dané měření.

- **RPM4**
  Jedná se o více segmentový displej popsaný v kapitole 4.5.3, v úpravě přečtených hodnot se nevyskytují žádné abnormálnosti, takže je možno tento typ použít na více podobných přístrojů se stejným zobrazením hodnot. Obsahuje všechny tři parametry pro úpravu snímku popsané výše a je zde umožněno měřit i dvě hodnoty zároveň.

- **seven segment zdroj**
  Jedná se o klasický sedmi segmentový displej používaný na laboratorních zdrojích popsaný v kapitole 4.5.1. Tento typ by měl být použitelný pro sedmi segmentové displeje se světlými hodnotami na tmavém pozadí. Obsahuje všechny tři parametry pro úpravu snímku popsané výše a je zde umožněno měřit i dvě hodnoty zároveň.

- **Agilent**
  Více segmentový displej přístroje Agilent zobrazuje světlé hodnoty na tmavém pozadí popsaný v sekci 4.5.4. Obsahuje všechny tři parametry pro úpravu snímku popsané výše.

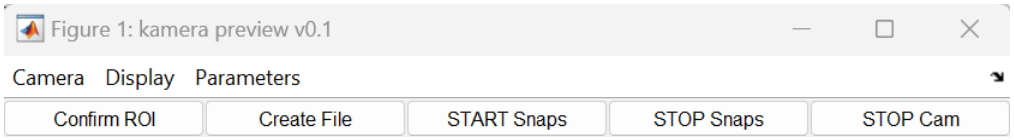
- **HP34401A**
  Jedná se o sedmi segmentový displej přístroje HP, zobrazující světlé hodnoty na černém pozadí, popsaný v sekci 4.5.5. Obsahuje všechny tři parametry pro úpravu snímku popsané výše.

- **D4**
  Displej přístroje Bronkhorst s vysokým rozlišením zobrazuje černé hodnoty na bílém pozadí, popsaný v sekci 4.5.6. Obsahuje pouze parametr pro filtraci.

- **D8**
  Jedná se o sedmi segmentový displej zobrazující tmavé hodnoty na světlém pozadí. Obsahuje parametry pro filtraci a *closing*.

- **D9**
  Displej typu Agilent, jedná se o dřívější verzi, která také fungovala, ale je náchylnější na chyby, každopádně jsme se jí rozhodli zachovat pro případ nefunkčnosti novější verze. Obsahuje všechny tři parametry pro úpravu snímku popsané výše.

V horní části uživatelského prostředí se nachází tlačítka pro ovládání měření viz 4.35.



Obrázek 4.35: Přehled tlačítek

Po nastavení displeje je dalším krokem výběr oblasti snímání, ten provedeme v náhledu kamery pomocí kurzoru myši, oblast můžeme následně upravit a potvrdit tlačítkem **Confirm ROI**.

Jako poslední před spuštěním měření musíme vytvořit CSV soubor do kterého se budou měřená data ukládat, to provedeme tlačítkem **Create File** čímž vytvoříme soubor s názvem ve tvaru `Data_2024-03-25_11-27-14`, název obsahuje datum a čas vytvoření (CSV soubor se nevytvoří hned po stisknutí tlačítka Create File, ale až po spuštění měření).

Měření se spustí tlačítkem **START Snaps**, tlačítko **STOP Snaps** pozastaví snímání, které můžeme předchozím tlačítkem **START Snaps** zase spustit, to platí také v případě zastavení měření způsobené chybou čtení nebo špatnými rozměry proměnné.

Ukončení programu se provádí tlačítkem **STOP Cam**, po kterém se při zapnutí aplikace musí všechna nastavení provést znovu.

5 Závěr

Cílem práce bylo usnadnit práci se staršími digitálními měřícími přístroji, které nemají komunikační rozhraní pro přenos naměřených dat do počítače, vytvořením programu pro zaznamenávání číselného displeje kamerou, následnou úpravou vyfocených snímků a uložením naměřených dat. Všechny tyto body zadání se podařilo splnit viz 4.5.

Úplně na začátku bylo nutné zvolit program, pomocí kterého se bude aplikace programovat, kdy jsme se rozhodovali mezi prostředími Python a *MATLAB*. Kvůli velkému množství online zdrojů na toto téma, z důvodu naší dosavadní zkušenosti s tímto prostředím a po konzultaci s vedoucím práce jsme se rozhodli pro *MATLAB*.

V rešeršní části (2) jsme se detailně zaměřili na funkce používané k zpracování obrázků pro jejich čtení pomocí *MATLAB*ovské *OCR* funkce včetně morfologických operací.

Kdybychom měli vyjmout některé z nich, tak byly pro úspěšné čtení dat z upravených snímků stěžejní funkce `rgb2gray`, tato funkce převádí barevné fotky na černobílé, `imadjust` pro zvýšení kontrastu tmavých a světlých pixelů, `imfilter` pro odstranění šumu, `imclose` na spojení segmentů znaků v obrázku, `imtophat` kvůli odstranění nekonstantního osvětlení a `imbinarize` pro převod z *grayscale* do binárního formátu.

V realizační části jsme se věnovali vytvoření programu pro snímání displeje USB kamerou, zpracování snímku, tedy praktickému využití rešeršní části. Dále čtení upravených snímků pomocí funkce *OCR*, následnému zpracování přečtených hodnot a jejich uložení pro následnou analýzu. Kapitoly 4.1 až 4.4 se zaměřují na vypracování jednotlivých bodů zadání, které jsou následně použity pro měření konktrétních displejů, což je popsáno v kapitole 4.5.

Při vytváření programu jsme zpočátku narazili na problémy s univerzalitou programu, kdy při malé odlišnosti displejů bylo nutné provádět někdy i větší změny v programu, konkrétně ve velikosti strukturálních prvků a použití různých funkcí pro zpracování obrazu, které jsou probírány v první části a hlavně ve zpracování přečtených znaků na data. Proto jsme se rozhodli jít spíše cestou lepší funkcionality a spolehlivosti pro menší počet zařízení, než se za každou cenu snažit o úplnou univerzálnost programu, která by sice byla optimální, ale ne úplně realistická, což by ve výsledku mohlo vést k nepoužitelnosti aplikace. Avšak právě proces univerzálnosti se nám podařilo vylepšit, alespoň co se úpravy obrázku týče, kdy pro většinu displejů používáme postup popsaný v kapitole 4.2. Ovšem univerzálnost programu je něco, co by bylo možné dále vylepšovat.

Dalším problémem byl různý font čísel různých displejů. To jak vypadají čísla zajisté hraje velkou roli, pokud se snažíme rozeznávat tyto čísla pouze na základě jejich vzhledu. Funkce *OCR*, která byla pro toto rozpoznávání použita, má několik parametrů, pomocí nichž se do jisté míry dá nastavit, jak čísla vypadají a podle toho pak tato funkce námi poskytnuté obrázky vyhodnocuje.

Pro nějaké fonty má však větší chybovost, než pro jiné, a i když se obrázek upraví do kvalitní binární podoby, v jistých případech jej nepřečte správně, což je jistý nedostatek používané *MATLAB* funkce. Řešením tohoto problému by do budoucna mohlo být

natrénování této funkce pomocí data setů pro každý displej zvlášť.

Jelikož měření neprobíhá pouze na jednom přístroji a jedné laboratoři, museli jsme vytvořit programy pro různé zobrazovací jednotky a možnost jejich výběru před počátkem měření, což se týká také kamer, kterými jsou snímky pořizovány, kdy je v aplikaci umožněn výběr z kamer právě připojených k počítači a pro ně použitelných rozlišení.

Co se jednotlivých zobrazovacích jednotek týče, je na výběr z několika konkrétních displejů a možnost úpravy parametrů každého z nich, což je popsáno v kapitole 4.5 a 4.7, kdy ve druhé ze zmíněných kapitol je popsáno, čím aplikace disponuje a postup pro používání GUI.

Zaznamenávání přečtených hodnot do CSV souboru a vypisování na sériový port nijak problémové nebylo. Zaznamenávání probíhá v reálném čase, naměřené hodnoty se spolu s časovou stopou zapisují a ukládají po každém snímku vždy na nový řádek bez možnosti úpravy předchozích, aby nebylo možné již naměřená data znehodnotit, pokud by nastala chyba.

Seznam použitých zkratek

OCR Optical Character Recognition

RGB Red Green Blue

ROI Region Of Interest

GUI Graphic User Interface

CSV Comma-Separated Values

Seznam zdrojů a použité literatury

[1] SMITH, Ray. An Overview of the Tesseract OCR Engine. In: [online]. [B.r.], s. 1–5 [cit. 2024-05-21]. Dostupné z DOI: 10.1109/ICDAR.2007.4376991.

[2] SZELISKI, Richard. *Computer Vision: Algorithms and Applications*. 2nd ed. Springer, 2022. ISBN 3030343715.

[3] *Types of Morphological Operations* [online]. 2024. [cit. 2024-04-08]. Dostupné z: https://www.mathworks.com/help/images/morphological-dilation-and-erosion.html.

[4] GRIFFIN, Anselm. *Matlab Morphology erode dilate open close strel tutorial* [online]. 2009. [cit. 2024-04-23]. Dostupné z: https://www.youtube.com/watch?v=ZTbGlriKFtU%5C&t=3s.

[5] *Rgb2gray* [online]. 2024. [cit. 2024-04-08]. Dostupné z: https://www.mathworks.com/help/matlab/ref/rgb2gray.html.

[6] *Imadjust* [online]. 2024. [cit. 2024-04-08]. Dostupné z: https://www.mathworks.com/help/images/ref/imadjust.html.

[7] *Imfilter* [online]. 2024. [cit. 2024-04-08]. Dostupné z: https://www.mathworks.com/help/images/ref/imfilter.html.

[8] *Imbinarize* [online]. 2024. [cit. 2024-04-08]. Dostupné z: https://www.mathworks.com/help/images/ref/imbinarize.html.

[9] TAY, Jian Wei. *Otsu's Method* [online]. 2020. [cit. 2024-04-10]. Dostupné z: https://www.youtube.com/watch?v=jUUkMaNuHP8%5C&t=2s.

[10] *Structuring Elements* [online]. 2024. [cit. 2024-04-08]. Dostupné z: https://www.mathworks.com/help/images/structuring-elements.html.

[11] *Strel* [online]. 2024. [cit. 2024-04-08]. Dostupné z: https://www.mathworks.com/help/images/ref/strel.html.

[12] GRIFFIN, Anselm. *Remove non constant illumination using iImtophat and imbothat in Matlab* [online]. 2012. [cit. 2024-04-23]. Dostupné z: `https://www.youtube.com/watch?v=s7FeO4c50K8%5C&t=1s`.

[13] *Morphological Operations* [online]. 2024. [cit. 2024-04-08]. Dostupné z: `https://www.mathworks.com/help/images/morphological-filtering.html`.

[14] GRIFFIN, Anselm. *OCR (Optical Character Recognition) in Matlab using imreconstruct, imtophat & imbinarize* [online]. 2017. [cit. 2024-04-23]. Dostupné z: `https://www.youtube.com/watch?v=BL9eP8qniwg`.

Přílohy

Kamera_reader Složka s Matlab kódem vytvořené aplikace