ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

# BAKALÁŘSKÁ PRÁCE

Optická čtečka

**Praha, 2012** **Autor: Tomáš Reichl**

## Prohlášení

Prohlašuji, že jsem zadanou bakalářskou práci vypracoval samostatně pod vedením prof. Ing. Pavla Zahradníka, CSc. a použil jsem pouze literaturu a zdrojové kódy v práci uvedené.

V Kladně dne 29.3.2012

.....

podpis

## Poděkování

Děkuji především vedoucímu bakalářské práce prof. Ing. Pavlu Zahradníkovi, CSc. za cenné připomínky a rady při řešení bakalářské práce. Dále bych chtěl poděkovat své rodině za podporu během dosavadního studia.

# Abstrakt

Tato práce se zabývá návrhem algoritmů a následně vytvořením softwaru v programovacím jazyce C, který by byl schopen v daném hardwaru a za pomoci kamery, rozpoznat sedmisegmentové znaky na měřicích přístrojích. V této práci jsou dále popsány základní pojmy a metody pro počítačové vidění a strojové rozpoznávání. Pro návrh a ladění algoritmu je použito vývojové prostředí Matlab. V poslední části je probrána implementace algoritmu do signálového procesoru TMS320DM6437 firmy Texas Instruments a dosažené výsledky při rozpoznávání.

# Abstract

This work deals with design algorithms and create software in programming language C, which would be capable of camera-based recognizing the seven-segment characters on measuring instruments. This paper also describes the basic concepts and methods for computer vision and machine recognition. For the design and debugging algorithm is used the Matlab environment. The last section discussed the implementation of the algorithm to the signal processor Texas Instruments TMS320DM6437 and the results achieved in the recognition.

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Tomáš Reichl**

Studijní program: Kybernetika a robotika
Obor: Systémy a řízení

Název tématu: **Optická čtečka**

Pokyny pro vypracování:

1. Proveďte rozbor řešení.
2. Navrhněte a realizujte funkční vzorek zařízení pro čtení číselných údajů z mechanického počítadla.
3. Pro snímání obrazu použijte polovodičový obrazový snímač popř. kameru s polovodičovým obrazovým snímačem.
4. Obrazová data vyhodnocujte vhodným mikrokontrolérem, mikroprocesorem popř. číslicovým signálovým procesorem. Uvažujte výstup číselných dat v sériové podobě pomocí sériového rozhraní.

Seznam odborné literatury:

[1] www.parallax.com
[2] www.ti.com

Vedoucí: Prof.Ing. Pavel Zahradník, CSc.

Platnost zadání: do konce zimního semestru 2012/2013

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 13. 12. 2011

# Obsah

**Seznam obrázků** **ix**

**Seznam tabulek** **x**

**1 Úvod** **1**

1.1 Počítačové vidění . . . . . 1

1.2 Cíle práce . . . . . 1

**2 Základní pojmy počítačového vidění** **3**

2.1 Reprezentace obrazu . . . . . 3

2.2 Základní postup při rozpoznávání . . . . . 6

**3 Návrh algoritmu** **8**

3.1 Snímání obrazu . . . . . 8

3.2 Předzpracování . . . . . 9

3.3 Segmentace obrazu . . . . . 11

3.3.1 Prahování . . . . . 11

3.3.2 Adaptivní prahování . . . . . 13

3.3.3 Segmentace založená na detekci hran . . . . . 13

3.3.4 Filtrace binárního šumu . . . . . 15

3.3.5 Identifikace objektů . . . . . 15

3.4 Lokalizace sedmisegmentových znaků . . . . . 17

3.4.1 Odstranění nevhodných objektů . . . . . 17

3.4.2 Sjednocení objektů . . . . . 18

3.4.3 Nalezení znaků . . . . . 20

3.4.4 Detekce desetinné tečky . . . . . 21

3.5 Klasifikace objektů . . . . . 22

3.6 Výsledky návrhu v Matlabu . . . . . 23

**4 Implementace** **26**

4.1 Seznámení s procesorem . . . . . 26

4.2 Přepis kódu . . . . . 27

4.3 Časová optimalizace . . . . . 29

**5 Experimentální výsledky** **32**

**6 Závěr** **35**

**Literatura** **36**

**A Seznam použitého software** **I**

# Seznam obrázků


1.1 Vzor sedmisegmentového znaku . . . . . 2

2.1 Čtvercová (vlevo) a hexagonální (vpravo) mřížka . . . . . 4
2.2 Histogram . . . . . 4
2.3 4-sousedství (vlevo) a 8-sousedství (vpravo) . . . . . 5
2.4 Příklad oblasti v 4-sousedství (vlevo) a 8-sousedství (vpravo) . . . . . 5
2.5 Transformace jasové stupnice - transformace pro zvýšení kontrastu (červeně), prahování (černě) a negativ (modře) . . . . . 6

3.1 Příklad sejmutého obrazu . . . . . 9
3.2 Výchozí obraz s nízkým kontrastem (vlevo) a obraz s ekvalizovaným histogramem (vpravo) . . . . . 10
3.3 Histogram obrazu s nízkým kontrastem (vlevo) a ekvalizovaný histogram (vpravo) . . . . . 11
3.4 Výsledek prahování s nízkým prahem (vlevo), se správně nastaveným prahem (uprostřed) a s vysokým prahem (vpravo) . . . . . 12
3.5 Určení hodnoty prahu z histogramu . . . . . 12
3.6 Výsledek prahování Niblackovou metodou pro lokální okolí 11×11 (vlevo) a 21×21 (vpravo) . . . . . 14
3.7 Cannyho hranový detektor . . . . . 15
3.8 Filtrační maska (vlevo), příklad stavu splňující (uprostřed) a nesplňující (vpravo) podmínku 3.5 . . . . . 16
3.9 Výsledek filtrování šumu - vlevo originál, vpravo výstupní obraz . . . . . 16
3.10 Maska pro barvení ve 4-sousednosti (vlevo) a 8-sousednosti (vpravo) . . . . 17
3.11 Algoritmus barvení pro 4-sousedství - výchozí obraz (vlevo), obraz po prvním průchodu (uprostřed), obraz po druhém průchodu (vpravo) . . . . . 17
3.12 Odstranění nevhodných objektů - vlevo originál, vpravo výstupní obraz . . 19

3.13 Procházení hranice objektů (červeně) . . . . . 19
3.14 Sjednocení objektů - vlevo originál, vpravo výstupní obraz . . . . . 20
3.15 Nalezení pozice znaků (červeně) . . . . . 21
3.16 Předpokládané oblasti desetinné tečky (zeleně) . . . . . 21
3.17 Klasifikační třídy . . . . . 22
3.18 Detekce segmentů ve znaku . . . . . 22
3.19 Příklad správně rozpoznaného snímku - rozpoznaná hodnota 155.7 . . . . . 24
3.20 Příklad správně rozpoznaného snímku - rozpoznaná hodnota 608.0 . . . . . 24
3.21 Příklad špatně rozpoznaného snímku - nenalezen druhý znak vlivem odrazu předmětu v displeji, rozpoznaná hodnota 11.4 . . . . . 25
3.22 Příklad špatně rozpoznaného snímku - nesprávně detekovaná desetinná tečka, rozpoznaná hodnota 1.401 . . . . . 25
3.23 Příklad špatně rozpoznaného snímku - chybná klasifikace třetího znaku vlivem nízkého kontrastu, rozpoznaná hodnota 17X.0 . . . . . 25

4.1 Vývojový modul TMS320DM6437 EVM . . . . . 26
4.2 Blokové schéma vývojového modulu TMS320DM6437 EVM (převzato z [11]) 27

5.1 Výsledek rozpoznávání zobrazený na výchozím obrazu (vlevo) a na obrazu po segmentaci (vpravo) . . . . . 34
5.2 Výsledek rozpoznávání zobrazený na výchozím obrazu (vlevo) a na obrazu po segmentaci (vpravo) . . . . . 34

# Seznam tabulek

3.1 Nastavení parametrů objektů . . . . . 18
3.2 Výsledky rozpoznávání . . . . . 23

4.1 Výpočetní doba algoritmu před optimalizací . . . . . 30
4.2 Výpočetní doba algoritmu po optimalizaci . . . . . 30
4.3 Srovnání výpočetní doby algoritmu před a po optimalizaci . . . . . 30

5.1 Výsledky rozpoznávání pro měřicí přístroj Voltcraft VC220 . . . . . 33
5.2 Výsledky rozpoznávání pro měřicí přístroj Solid RE830D . . . . . 33
5.3 Shrnutí výsledků rozpoznávání . . . . . 33

# Kapitola 1

# Úvod

## 1.1 Počítačové vidění

S rozvojem výpočetní techniky se zrodil nový perspektivní obor - počítačové vidění, který si klade za cíl vytvářet a zpracovávat informace ze zachyceného digitalizovaného obrazu. Metody počítačového vidění mají uplatnění v mnoha oborech jako robotika, doprava nebo medicína. Jednou z úloh počítačového vidění je nalezení a určení alfanumerických znaků z naskenovaného nebo nasnímaného obrazu nazývané optické rozpoznávání znaků (Optical Character Recognition).

Optické rozpoznávání znaků je založené na detekci a klasifikaci objektů, které svým charakterem nejvíce odpovídají hledaným znakům. Je třeba podotknout, že samotný problém zpracování obrazu je velice složitý, protože každý obraz je zatížen šumem a chybou vzniklou digitalizací obrazu. Proces optického rozpoznávání znaků není nikdy naprosto bezchybný, ale je jen více či méně přesný.

## 1.2 Cíle práce

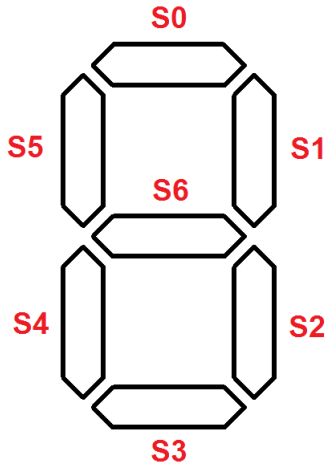
Cílem této práce je navrhnout algoritmy a realizovat zařízení pro rozpoznávání sedmisegmentových číselných údajů na měřících přístrojích. Zaměříme se na měřicí přístroje s LCD displejem, na kterém jsou údaje prezentovány tmavou barvou na světlém pozadí. Ke snímání obrazu použijeme polovodičový obrazový snímač. Návrh, ladění a testování algoritmu provedeme v prostředí Matlab, které poskytuje potřebnou podporu obrazového zpracování. Následně převedeme zdrojový kód do jazyka C a implementujeme do vhodného mikroprocesoru. Výsledné zařízení se bude skládat z obrazového snímače, mikroprocesoru

a výstupu pro připojení k dalšímu zařízení. Toto zařízení by mohlo najít uplatnění např. pro zapisování měřené veličiny z přístroje, který nedisponuje digitálním výstupem nebo pro kalibraci měřících přístrojů.

Zadaná úloha je podobná úloze optického rozpoznávání znaků, ale je třeba upozornit na některé odlišnosti.

1. Hledané sedmisegmentové znaky nejsou spojitě spojeny, ale skládají se z jednotlivých segmentů, které mohou být po segmentaci obrazu rozděleny do více objektů. Vzor sedmisegmentového znaku je vidět na obrázku 1.1. Při hledání objektů, které odpovídají sedmisegmentovým znakům je proto třeba s tímto omezením počítat. Tento problém je popsán v kapitole 3.4.

2. Při snímání měřících přístrojů dochází velmi často ke snížení kontrastu číselných údajů, vlivem nedostatečného osvětlení LCD displeje. Je proto nutné navrhnout robustní segmentační metody, případně použít vhodnou jasovou transformaci. Tímto problémem se zabývá kapitola 3.3.




Obrázek 1.1: Vzor sedmisegmentového znaku

# Kapitola 2

# Základní pojmy počítačového vidění

V této kapitole vysvětlíme základní pojmy a metody počítačového vidění, které budeme v této práci používat. Více informací o počítačovém vidění je možno najít např. v [1].

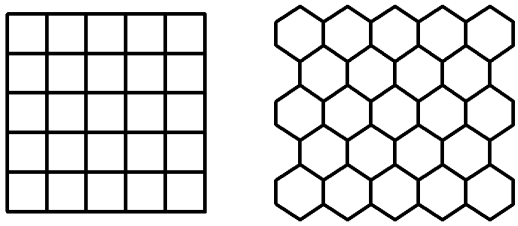
## 2.1 Reprezentace obrazu

Abychom mohli s obrazem pracovat, musíme nejdříve spojitý signál převést do digitální podoby. Digitalizace obrazu je dosaženo vzorkováním obrazu do konečného množství obrazových bodů, které označujeme jako obrazové elementy a kvantováním jasu do několika diskrétních hladin. Velmi důležitá je volba množství obrazových elementů a velikosti vzorkovacího intervalu. Čím větší bude počet obrazových elementů a menší vzorkovací interval, tím věrohodněji bude obraz reprezentovat originální obraz. Digitalizace obrazu je nejčastěji součástí samotného snímacího zařízení, a proto se touto problematikou nebudeme blíže zabývat.

U barevných digitálních obrazů musí každý obrazový element navíc obsahovat informaci o barevné složce. To, jakým způsobem je informace o jasu a barvě v obrazovém elementu uložena, určuje tzv. barevný model. V této práci budeme pracovat s dvěma barevnými modely – RGB a YCbCr. RGB model je aditivní model založený na zastoupení červené, zelené a modré složky. Mícháním těchto složek vzniká výsledná barva a intenzita barevných složek určuje jas. U YCbCr modelu představuje složka Y jas a kombinace složek Cb, Cr udávají barvu.

Důležitým pojmem reprezentace obrazu je vzorkovací mřížka. Vzorkovací mřížka definuje uspořádání jednotlivých obrazových elementů. Nejčastěji používáme mřížky čtvercové a mřížky hexagonální. Čtvercová mřížka je velmi snadno realizovatelná a vychází z kon-

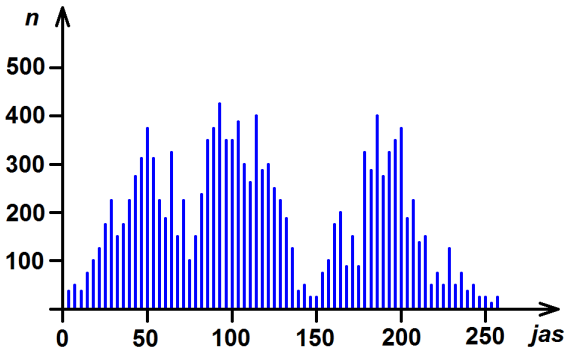
strukce snímacích zařízení. Výhodou hexagonální mřížky je symetričnost vzhledem k okolním obrazovým elementům. Tvar obou vzorkovacích mřížek je znázorněn na obrázku 2.1. V našem případě budeme používat mřížku čtvercovou.



Obrázek 2.1: Čtvercová (vlevo) a hexagonální (vpravo) mřížka

## Histogram

Grafické znázornění zastoupení jednotlivých jasových hladin v obrazu vyjadřuje histogram. Histogram je graf, který na vodorovné ose vyznačuje jasové hladiny a na svislé ose četnosti jednotlivých jasových hladin v celém obrazu. Histogram je důležitým prostředkem pro jasové transformace a korekce v obrazu. Příklad histogramu můžeme vidět na obrázku 2.2.

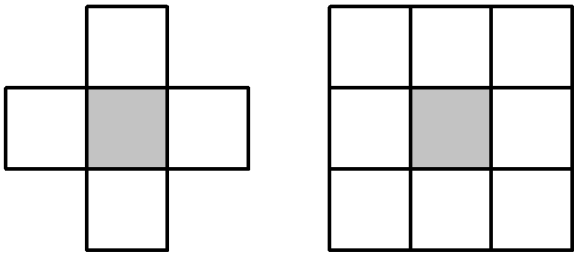



Obrázek 2.2: Histogram

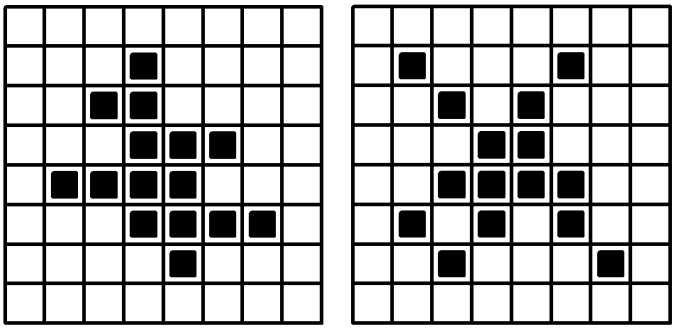
## Oblasti

Pro obrazový element definujeme jeho souseda jako element, který patří do jeho sousedství. Pro čtvercové vzorkovací mřížky uvažujeme nejčastěji 4-sousedství a 8-sousedství

(obr 2.3). Množinu obrazových elementů v binárním obraze, které jsou vzájemně sousedy v daném sousedství a mají společnou hodnotu, budeme označovat jako oblast. Na obrázku 2.4 jsou znázorněny příklady oblastí ve 4-sousedství a v 8-sousedství. Oblasti v obraze, které odpovídají předmětům v reálném prostředí interpretujeme jako objekty. Tyto pojmy budeme často používat pro segmentační techniky.



Obrázek 2.3: 4-sousedství (vlevo) a 8-sousedství (vpravo)



Obrázek 2.4: Příklad oblasti v 4-sousedství (vlevo) a 8-sousedství (vpravo)

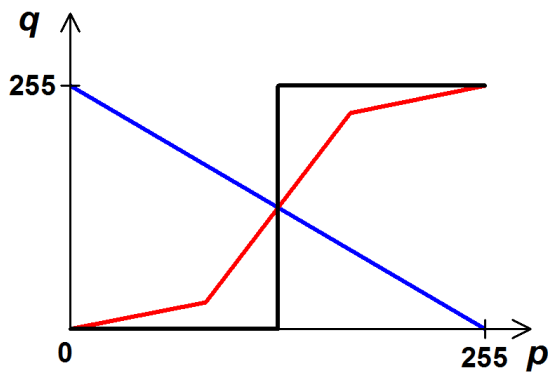
## Transformace jasové stupnice

Cílem transformace jasové stupnice je přiřadit všem jasovým hodnotám novou hodnotu jasu. Je stejná pro všechny obrazové elementy v obraze. Transformace stupnice jasu $q$ na novou stupnici $p$ je dána vztahem (vztah převzat z [1])

$$q = T(p) \tag{2.1}$$

Transformace rozdělujeme na lineární a nelineární. Lineární transformace umožňuje zpětné převedení na výchozí stupnici. Mezi základní transformace jasové stupnice řadíme trans-

formaci pro zvýšení kontrastu, prahování a negativ (obrázek 2.5). Speciálním typem transformace je ekvalizace histogramu.




Obrázek 2.5: Transformace jasové stupnice - transformace pro zvýšení kontrastu (červeně), prahování (černě) a negativ (modře)

## 2.2 Základní postup při rozpoznávání

Základní postup při rozpoznávání obrazu můžeme rozdělit do následujících kroků

- Snímání a digitalizace obrazu
- Předzpracování
- Segmentace obrazu
- Popis objektů
- Klasifikace

Poznamenejme, že použité rozdělení není jednoznačné, protože v různých literaturách se můžeme setkat s jiným dělením. Volba jednotlivých kroků je závislá na konkrétní úloze a potřebách autora.

### Snímání a digitalizace obrazu

Nejdůležitějším krokem je vlastní získání obrazu. Snímáním obrazu rozumíme převod vstupní optické veličiny na elektrický signál. Abychom mohli obraz uložit a následně ho

zpracovat, musíme jeho spojitý vstupní signál digitalizovat. Digitalizací jsme se zabývali v kapitole 2.1. K zachycení obrazu nejčastěji používáme kameru, scanner nebo jiné snímací zařízení.

### Předzpracování

Získaný obraz může být zkreslen a zatížen šumem vlivem způsobu snímání a digitalizace nebo špatných optických podmínek. Cílem předzpracování je toto zkreslení odstranit nebo alespoň částečně potlačit. Dalším úkolem předzpracování může být zvýraznění určitých rysů v obraze nebo jiná úprava obrazu pro snadnější zpracování. Tento krok bývá často vynechán a záleží na kvalitě sejmutého obrazu a našich potřebách.

### Segmentace

Nejtěžším krokem v procesu rozpoznávání je segmentace obrazu na jednotlivé objekty. Segmentace se snaží oddělit objekty od pozadí. Za objekty zde budeme považovat ty části obrazu, které odpovídají předmětům v reálném prostředí. Jen ve velmi ojedinělých případech je možné dosáhnout kompletní segmentace, a proto se většinou snažíme o co nejlepší částečnou segmentaci, která nalezne alespoň nejvýraznější objekty. Výstupem segmentace často bývá binární obraz, kde oblasti s hodnotou jedna odpovídají objektům a oblasti s nulovou hodnotou představují pozadí.

### Popis objektů

Abychom mohli v segmentovaném obraze najít objekty, které nás zajímají, musíme je vhodným způsobem popsat. Způsob popisu by měl být zvolen tak, aby co nejlépe charakterizoval tvar objektu. Nejčastěji objekty popisujeme pomocí souboru číselných charakteristik. Tyto číselné údaje označujeme jako příznaky. Příkladem velmi jednoduchého popisu objektu může být velikost jeho výšky, šířky nebo plochy.

### Klasifikace

Finálním krokem je klasifikace nalezených objektů. Při klasifikaci se snažíme nalezené předměty zařadit do předem definovaných tříd. Testovaný objekt zařadíme do takové třídy, která svým popisem nejlépe odpovídá popisu testovaného objektu. Existuje celá řada metod pro klasifikaci objektů a jsou popsány např. v [7].

# Kapitola 3

# Návrh algoritmu

V této kapitole popíšeme navrhnutý postup rozpoznávání v Matlabu a vysvětlíme použité metody. Celý proces rozpoznávání je rozdělen do několika kroků, které popisují jednotlivé podkapitoly.

## 3.1 Snímání obrazu

Ke snímání obrazu použijeme barevnou webovou kameru Trust WB-1400T s USB výstupem, rozlišením 352 × 288 a barevnou hloubkou 24 bitů. Rozlišení kamery je pro účely našeho rozpoznávání dostatečné a navíc zaručí menší výpočetní náročnost než u kamer s vysokým rozlišením. Kamera disponuje manuálním ostřením. Pro sejmutí a uložení obrazu v Matlabu používáme následující příkazy

```
cam = videoinput('winvideo',1,'RGB24_352x288');
preview(cam)
obr = getsnapshot(cam);
```

Prvním příkazem zvolíme kameru pro vytvoření video vstupu. Pro určení parametrů zvoleného snímacího zařízení můžeme použít příkaz `imaqhwinfo()`. Druhý příkaz slouží k zapnutí náhledu kamery. Posledním příkazem provedeme sejmutí obrazu a uložení do proměnné `obr`. Výstupní obraz je uložen do trojrozměrné matice 288 × 352 × 3. Matlab používá pro uložení a správu obrázků barevný model RGB. Příklad sejmutého obrázku měřicího přístroje je na obrázku 3.1.


Obrázek 3.1: Příklad sejmutého obrazu

## 3.2 Předzpracování

Cílem předzpracování je upravit obraz pro zpracování a další metody. V tomto kroku se snažíme odstranit šum a zkreslení vzniklé při snímání obrazu nebo zvýraznit či potlačit některé rysy.

### Převod do černobílého formátu

Většina segmentačních metod pracuje s monochromatickým obrazem. Proto v první fázi převedeme uložený barevný obrázek do černobílého formátu. Dopustíme se tím částečné ztráty informace, avšak tato informace není pro naši úlohu podstatná. Barevná informace je navíc závislá na dalších parametrech a má tak pro proces rozpoznávání malou vypovídací hodnotu. Pro převod barevného modelu RGB do černobílého formátu existuje celá řada metod. My vybereme převod, který je popsán v [4] a je určen následujícím vztahem

$$Y(i,j) = 0.299\,R(i,j) + 0.587\,G(i,j) + 0.114\,B(i,j) \tag{3.1}$$

pro každý obrazový element řádku $i$ a sloupce $j$ v obraze. Převedený obraz tak bude uložen v dvourozměrné matici s rozměry $288 \times 352$.

### Filtrace šumu

Cílem filtrace je odstranění náhodného šumu v obrazu, který je způsoben snímacím zařízením, digitalizací obrazu nebo špatnými optickými podmínkami. Mezi nejpoužívanější metody k odstranění šumu patří filtrace průměrováním a filtrace metodou mediánu. Fil-

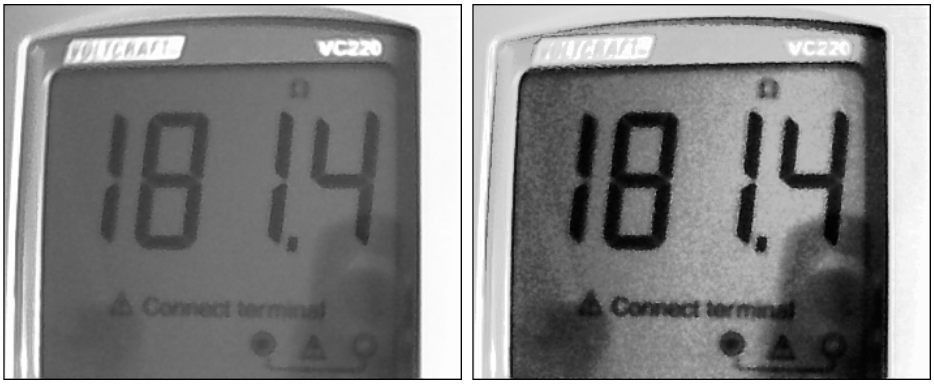
trace průměrováním je založena na určení hodnoty jasu aritmetickým průměrem jeho okolí. U filtrace metodou mediánu určíme hodnotu nového jasu jako medián v lokálním okolí. Obě metody úspěšně vyhlazují obrázek a odstraňují šum, ale dochází při nich k rozmazávání hran v obraze, a proto žádnou filtraci v tomto návrhu aplikovat nebudeme.

## Ekvalizace histogramu

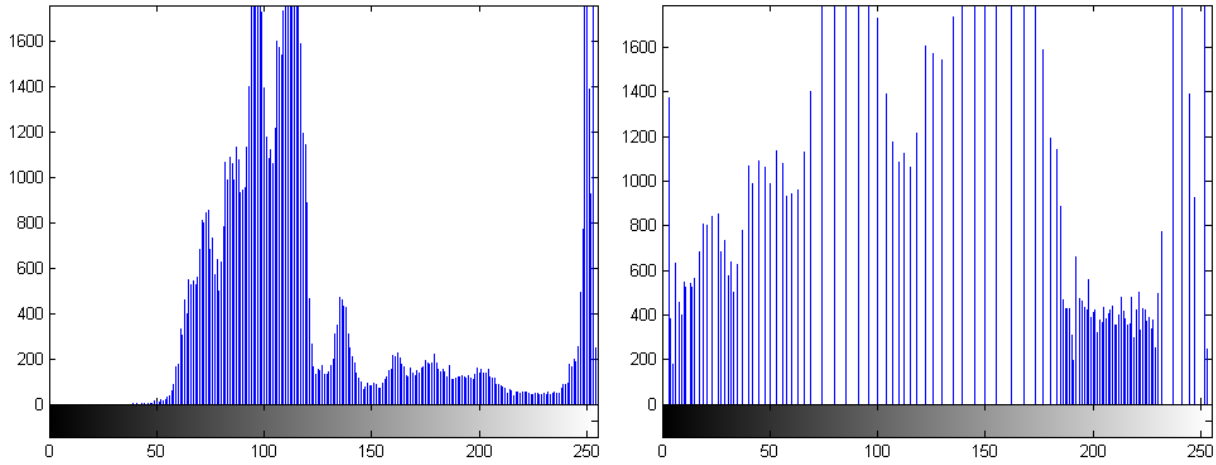
Snímaný obraz může mít vlivem špatných okolních podmínek nízký kontrast. Jednou z metod pro automatické zvýšení kontrastu snímku je ekvalizace histogramu. Jedná se o transformaci jasové stupnice (viz. kapitola 2.1), která se snaží vyrovnat zastoupení všech jasových hodnot. Uvedená transformace je v diskrétním případě dána vztahem (vztah převzat z [1])

$$q = T(p) = \frac{q_k - q_0}{N^2} \sum_{i=p_0}^{p} H(i) + q_0 \tag{3.2}$$

kde $p = \langle p_0, p_k \rangle$ představuje vstupní jasovou stupnici, $q = \langle q_0, q_k \rangle$ představuje výstupní jasovou stupnici, $N^2$ počet obrazových elementů v obraze a $H(i)$ hodnotu histogramu pro jas $i$. Na obrázku 3.2 můžeme vidět příklad výchozího snímku s nízkým kontrastem a snímku s ekvalizovaným histogramem. Ekvalizace histogramu zvyšuje kontrast obrazu, ale zároveň dochází ke zkreslení některých částí obrazu (např. hran). Proto ji ve výsledném algoritmu nepoužijeme.




Obrázek 3.2: Výchozí obraz s nízkým kontrastem (vlevo) a obraz s ekvalizovaným histogramem (vpravo)




Obrázek 3.3: Histogram obrazu s nízkým kontrastem (vlevo) a ekvalizovaný histogram (vpravo)

## 3.3 Segmentace obrazu

V tomto kroku se budeme snažit najít vhodnou metodu pro segmentaci obrazu. Cílem segmentace je oddělit části obrazu, které odpovídají předmětům v reálném prostředí od pozadí. Na zvolené segmentační technice závisí robustnost celého rozpoznávání, a proto jí budeme věnovat zvýšenou pozornost. V této práci se zaměříme na metody využívající prahování a metody založené na detekci hran.

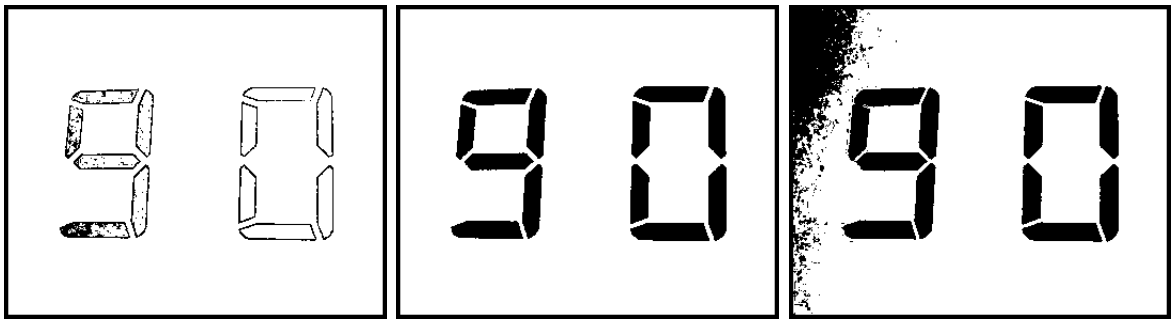
### 3.3.1 Prahování

Nejjednodušší metodou segmentace obrazu je prahování. Princip této metody spočívá v porovnávání jasové hodnoty každého obrazového elementu vzhledem k určité stanovené mezi. Tuto mez označujeme jako práh. Obrazové elementy s hodnotou jasu menší než zvolený práh označíme jako objekty a ostatní obrazové elementy označíme jako pozadí (případně naopak). Prahování je vlastně transformací jasové stupnice, kterou můžeme popsat následujícím vztahem

$$q(i,j) = \begin{cases} 1, & \text{pro } p(i,j) \geq T \\ 0, & \text{pro } p(i,j) < T \end{cases} \tag{3.3}$$

kde $p$ označuje vstupní jasovou stupnici, $q$ výstupní jasovou stupnici a $T$ zvolený práh. Výhodou prahování je velmi malá výpočetní náročnost. Na obrázku 3.4 můžeme vidět

příklad prahování s nízkým prahem, vysokým prahem a se správně nastaveným prahem. Pro správnou funkci prahování je podstatná správná volba prahu.

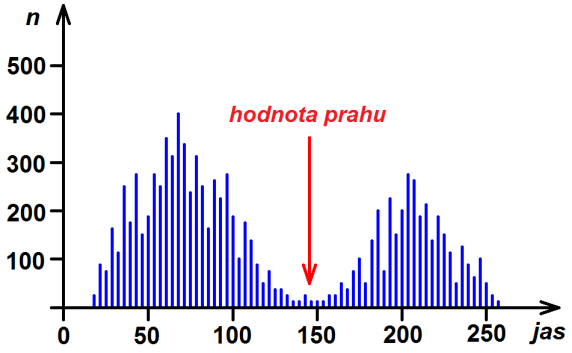


Obrázek 3.4: Výsledek prahování s nízkým prahem (vlevo), se správně nastaveným prahem (uprostřed) a s vysokým prahem (vpravo)

## Metody určování prahu

Pokud známe předpokládané procentuální zastoupení objektů v obraze (např. úloha segmentace tištěného textu na papíře), můžeme použít prahovaní založené na této apriorní znalosti. Tuto metodu pak označujeme jako procentuální prahování. Označme procentuální zastoupení objektů v obraze jako $p$. Hodnotu prahu potom zvolíme tak, aby $p$ procent plochy v obraze mělo menší hodnotu jasu než práh $T$ (případně naopak).

Složitější metoda určení prahu je založena na analýze histogramu. Pokud histogram obrazu vykazuje dva oddělené vrcholy (tzv. bimodální histogram), můžeme práh určit jako hodnotu jasu s nejmenší četností mezi oběma vrcholy (obrázek 3.5). Tato metoda je však použitelná pouze pro obraz, u kterého jsou objekty výrazně jasově odlišeny od pozadí.




Obrázek 3.5: Určení hodnoty prahu z histogramu

Dosud jsme se zabývali pouze prahováním, které používá jednotný práh pro všechny obrazové elementy v obrázku. Tento způsob prahování často označujeme jako prahování globální. Toto prahování můžeme použít jen pro obrazy, u kterých je zaručeno konstantní osvětlení ve všech částech obrazu. Tato podmínka však ve většině případů splněna není. Tento nedostatek odstraňuje tzv. adaptivní prahování.

### 3.3.2 Adaptivní prahování

U adaptivního prahování je hodnota prahu určena pro každou část obrazu zvlášť. Velikost prahu se tak dynamicky mění během procesu segmentace obrazu. Nejčastěji určujeme hodnotu prahu pro každý jednotlivý obrazový element z jeho definovaného okolí. Další možností je rozdělit obraz do několika překrývajících se částí a pro tyto části určit správný práh. Adaptivní prahování využívá následující metoda.

#### Niblackova metoda

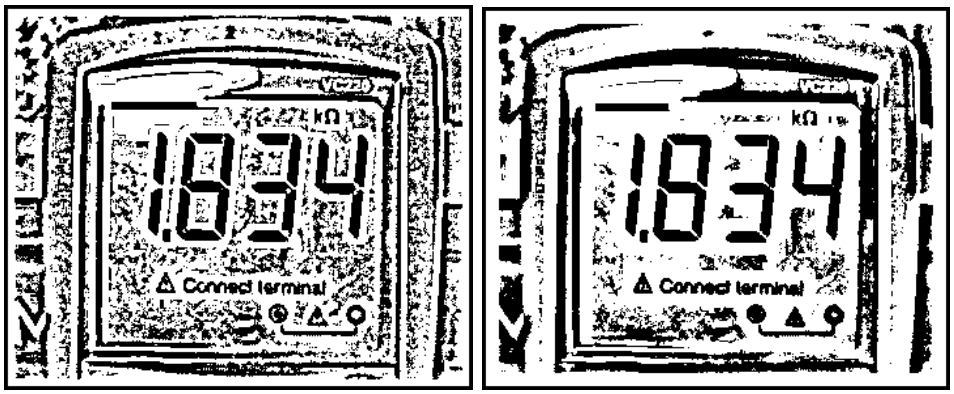
Niblackova metoda je založena na výpočtu prahu ze střední hodnoty a směrodatné odchylky pro každý obrazový element z jeho lokálního okolí. Hodnotu prahu určíme z následujícího vztahu

$$T(i,j) = m(i,j) + p\ s(i,j) \tag{3.4}$$

kde $m(i,j)$ označuje střední hodnotu lokálního okolí, $s(i,j)$ směrodatnou odchylku lokálního okolí, $p$ parametr prahování a $T(x,y)$ výsledný práh. Velikost lokálního okolí musí být tak velká, aby zahrnula dostatečnou část objektu a pozadí a zároveň tak malá, aby byla zaručena konstantní úroveň osvětlení v tomto výřezu. Doporučovaná hodnota parametru $p$ je $-0.2$. Výsledek prahování Niblackovou metodou můžeme vidět na obrázku 3.6.

### 3.3.3 Segmentace založená na detekci hran

Další skupinou metod určené k segmentaci obrazu jsou metody založené na detekci hran. Princip těchto metod vychází z detekce hranic mezi objektem a pozadím. Tyto hranice nalezneme použitím hranového operátoru. V dalším kroku spojíme nalezené hrany, které příslušejí stejnému objektu a odstraníme hrany, které odpovídají šumu a nehomogenitám v obraze.




Obrázek 3.6: Výsledek prahování Niblackovou metodou pro lokální okolí 11×11 (vlevo) a 21×21 (vpravo)

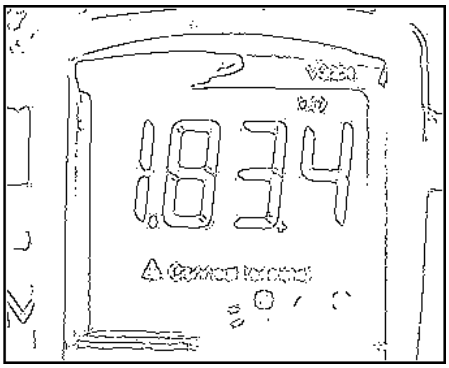
## Cannyho hranový detektor

Jedním z nejlepších a nejpoužívanějších hranových detektorů je Cannyho hranový detektor. Často bývá označován jako optimální hranový detektor. Tento detektor se snaží detekovat všechny významné hrany, minimalizovat počet chybných hran, maximalizovat přesnost polohy hran a zabránit zdvojování hran. Implementace Cannyho detektoru je poměrně složitá, a proto uvedeme jen základní kroky algoritmu

1. odstranění šumu Gaussovým filtrem
2. určení gradientu pomocí vhodné masky
3. nalezení lokálních maxim hodnot gradientů
4. odstranění nevýznamných hran prahováním s hysterezí

Příklad segmentace obrazu Cannyho hranovým detektorem je na obrázku 3.7. Nevýhodou této segmentační metody je velký počet vstupních parametrů a nutnost jejich správného nastavení.

Jako výslednou segmentační metodu pro náš návrh zvolíme Niblackovu metodu prahování, která poskytuje nejlepší výsledky. Experimentálně jsme určili optimální velikost lokálního okolí 21 × 21.



Obrázek 3.7: Cannyho hranový detektor

### 3.3.4 Filtrace binárního šumu

Protože zvolená segmentační metoda zatěžuje obrázek binárním šumem, navrhneme jednoduchý filtr na odstranění tohoto šumu. Navrhnutý postup vychází z algoritmu kFill (více informací v [10]). U této filtrace předpokládáme, že nejmenší velikost šířky a výšky hledaného objektu je větší než 3 pixely. Dále budeme uvažovat pouze přítomnost černého šumu na bílém pozadí (někdy bývá označován jako šum typu „pepř“).

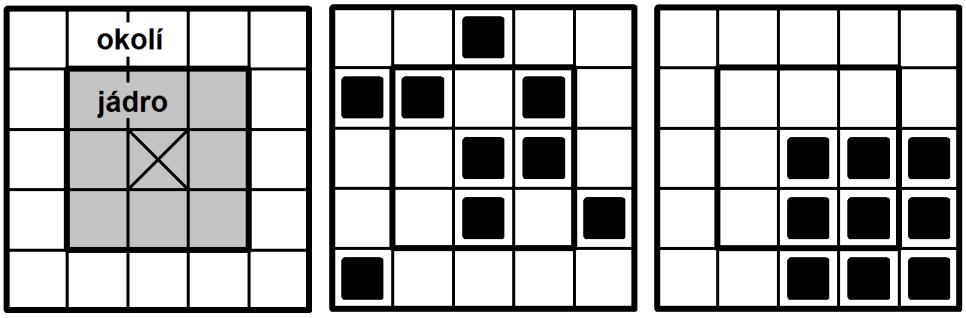
Nejprve navrhneme filtrační masku $5 \times 5$, která se bude skládat z jádra a okolí (obrázek 3.8). Touto maskou projedeme celý obrázek zleva doprava a shora dolů. Označme počet nenulových pixelů v okolí jako $n$ a hodnotu pixelu ve středu jádra jako $s$. Jádro filtrační masky pak vynulujeme, pokud platí podmínka

$$(s = 1) \text{ AND } (n < 5) \tag{3.5}$$

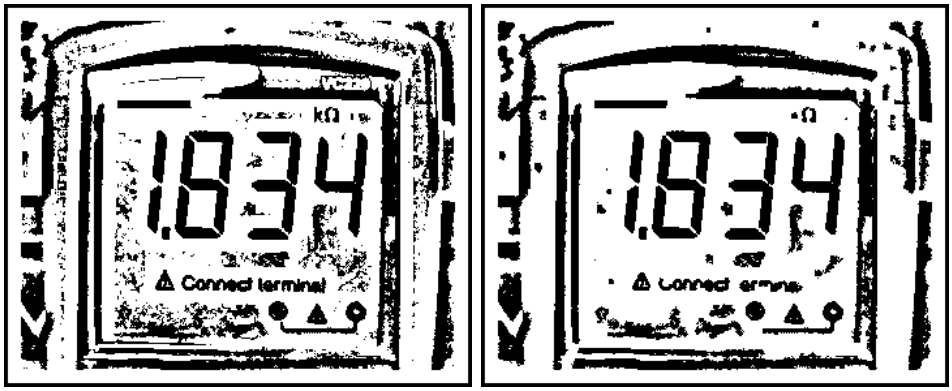
Tímto filtrem úspěšně odstraníme šum do velikosti $3 \times 3$. První podmínka ve vztahu 3.5 zaručuje, že výpočet provedeme jen pro části obrazu s nenulovým středem filtrační masky, čímž snížíme výpočetní náročnost algoritmu zhruba na polovinu. Výsledky navrhnutého postupu ukazuje obrázek 3.9.

### 3.3.5 Identifikace objektů

Abychom mohli se vzniklými objekty efektivně pracovat, musíme je od sebe vhodným způsobem odlišit. Jednou z možností je každému objektu přiřadit identifikační číslo. Tento způsob identifikace nazýváme barvením. Princip této identifikace spočívá v „obarvení“ každého objektu unikátní barvou. Barvením zde máme na mysli přiřazení určité hodnoty (barvy) všem pixelům odpovídající jednomu objektu. Nejvyšší identifikační číslo pak udává




Obrázek 3.8: Filtrační maska (vlevo), příklad stavu splňující (uprostřed) a nesplňující (vpravo) podmínku 3.5

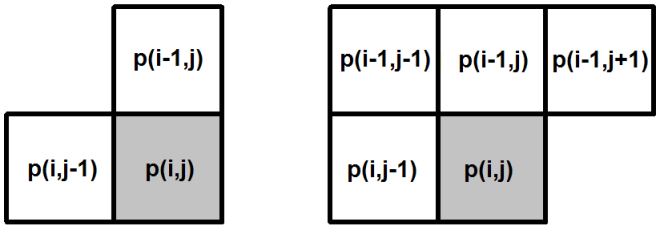



Obrázek 3.9: Výsledek filtrování šumu - vlevo originál, vpravo výstupní obraz

celkový počet objektů v obraze. Algoritmus barvení je detailně popsán např. v [1] a můžeme ho stručně shrnout do následujících kroků

1. procházíme po řádcích všechny nenulové pixely
2. aktuálnímu pixelu přiřadíme hodnotu (barvu) podle jedné z následujících možností
   (a) pokud jsou všechny sousední pixely odpovídající masky (obrázek 3.10) nulové, přiřadíme mu novou dosud nepoužitou hodnotu
   (b) pokud mají všechny sousední nenulové pixely stejnou hodnotu, přiřadíme mu tuto hodnotu
   (c) pokud má více sousedních pixelů různou nenulovou hodnotu, přiřadíme mu jednu z těchto hodnot a zapíšeme všechny různé hodnoty do seznamu ekvivalentních hodnot (tento stav označujeme jako kolize barev)

3. projdeme všechny nenulové pixely obrazu podruhé a přebarvíme oblasti s více ekvivalentními hodnotami novou hodnotou shodnou pro celou aktuální oblast




Obrázek 3.10: Maska pro barvení ve 4-sousednosti (vlevo) a 8-sousednosti (vpravo)

Popsaný postup ilustruje obrázek 3.11. Pro zajištění menší výpočetní náročnosti a snazší implementace použijeme pro náš návrh masku pro barvení ve 4-sousednosti. Vyhneme se tím navíc sjednocení objektů, které jsou spojeny diagonálně pouze jedním rohem a pravděpodobně k sobě nepatří.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | **1** | 0 | **1** | **1** | **1** | 0 | **1** | 0 |
| 0 | 0 | 0 | 0 | **1** | **1** | 0 | **1** | 0 |
| 0 | **1** | **1** | **1** | **1** | **1** | 0 | 0 | 0 |
| 0 | 0 | 0 | **1** | **1** | **1** | 0 | **1** | 0 |
| 0 | **1** | 0 | 0 | **1** | **1** | **1** | **1** | 0 |
| 0 | **1** | **1** | 0 | **1** | **1** | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | **1** | 0 | **2** | **2** | **2** | 0 | **3** | 0 |
| 0 | 0 | 0 | 0 | **2** | **2** | 0 | **3** | 0 |
| 0 | **4** | **4** | **4** | **2** | **2** | 0 | 0 | 0 |
| 0 | 0 | 0 | **4** | **2** | **2** | 0 | **5** | 0 |
| 0 | **6** | 0 | 0 | **2** | **2** | **2** | **5** | 0 |
| 0 | **6** | **6** | 0 | **2** | **2** | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | **1** | 0 | **2** | **2** | **2** | 0 | **3** | 0 |
| 0 | 0 | 0 | 0 | **2** | **2** | 0 | **3** | 0 |
| 0 | **2** | **2** | **2** | **2** | **2** | 0 | 0 | 0 |
| 0 | 0 | 0 | **2** | **2** | **2** | 0 | **2** | 0 |
| 0 | **4** | 0 | 0 | **2** | **2** | **2** | **2** | 0 |
| 0 | **4** | **4** | 0 | **2** | **2** | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Obrázek 3.11: Algoritmus barvení pro 4-sousedství - výchozí obraz (vlevo), obraz po prvním průchodu (uprostřed), obraz po druhém průchodu (vpravo)

## 3.4 Lokalizace sedmisegmentových znaků

Po úspěšné segmentaci obrazu a identifikaci všech objektů můžeme přikročit k procesu nalezení objektů, které odpovídají sedmisegmentovým znakům.

### 3.4.1 Odstranění nevhodných objektů

Segmentovaný obraz obsahuje velké množství objektů a je nanejvýš vhodné v první fázi odstranit objekty, u nichž je předem jasné, že nemohou být částmi hledaných sedmisegmentových znaků. Tato redukce bude mít pozitivní vliv na výpočetní dobu při následném

zpracování. Navrhneme proto jednoduchý algoritmus pro odstranění těchto objektů. U testovaných objektů pro nás bude rozhodující výška objektu $h$, šířka objektu $l$ a poměr mezi vlastní výškou a šířkou $p$. Nejprve je nutné zavést dolní a horní mez pro předchozí parametry. Tyto meze závisejí na použitém vstupním rozlišení kamery, na vzdálenosti snímání měřicího přístroje a na velikosti předpokládaných sedmisegmentových znaků a je proto nutné je nastavit podle potřeby. Nastavení těchto hodnot pro náš návrh shrnuje tabulka 3.1.

| | dolní mez ($x_{min}$) | horní mez ($x_{max}$) |
|---|---|---|
| výška $h$ | 5 | 140 |
| šířka $l$ | 5 | 80 |
| poměr $p$ | 0.2 | 9 |

Tabulka 3.1: Nastavení parametrů objektů

Jelikož může být objektem jen část sedmisegmentového znaku (např. samostatný segment), je třeba podle toho přizpůsobit dolní meze parametrů. Po určení těchto hodnot již můžeme implementovat vlastní metodu pro odstranění nevhodných objektů, která se skládá z následujících kroků

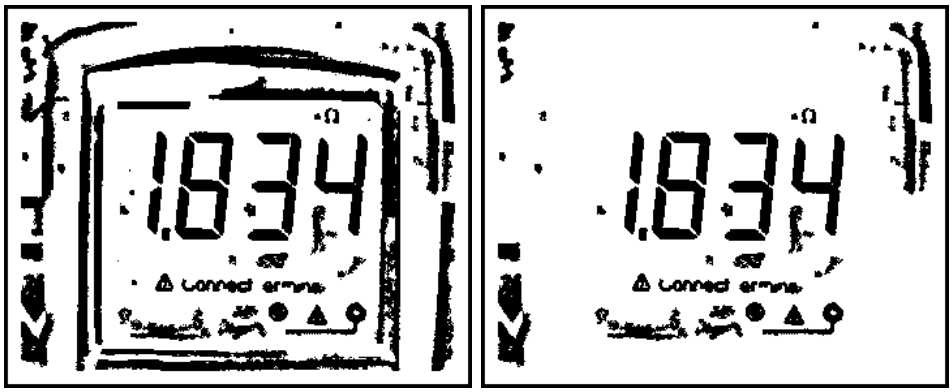
1. projdeme všechny objekty v obraze
2. určíme výšku $h$, šířku $l$ a poměr $p$ aktuálního objektu
3. objekt odstraníme z obrazu, jestliže platí podmínka

$$(h < h_{min}) \text{ OR } (h > h_{max}) \text{ OR } (l < l_{min}) \text{ OR } (l > l_{max}) \text{ OR } (p < p_{min}) \text{ OR } (p > p_{max})$$

Obrázek 3.12 porovnává vstupní a výstupní obraz této metody.

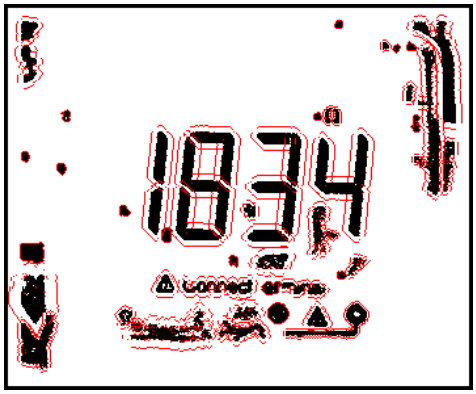
### 3.4.2 Sjednocení objektů

Sedmisegmentové znaky jsou tvořeny jednotlivými segmenty, které jsou v závislosti na kvalitě segmentace nesourodě spojeny. Abychom mohli v obraze naleznout pozici sedmisegmentových znaků, musíme nejprve sjednotit oblasti odpovídající segmentům jednoho znaku. Pro tento účel jsme navrhli vlastní metodu sjednocení objektů. Využíváme při tom apriorní znalosti, že všechny segmenty hledaného znaku mají shodnou šířku.



Obrázek 3.12: Odstranění nevhodných objektů - vlevo originál, vpravo výstupní obraz

V této metodě postupně projdeme všechny objekty v obraze. Předpokládáme, že každý objekt může být částí hledaného znaku. Pro aktuální objekt procházíme hranici jeho oblasti po směru hodinových ručiček a ve vzdálenosti $l$ hledáme přítomnost sousedních objektů (obrázek 3.13). Pokud nalezneme cizí sousední objekt, porovnáme šířku segmentu aktuálního objektu a šířku segmentu nalezeného objektu. Jestliže jsou tyto hodnoty přibližně shodné, sjednotíme oba objekty. Sjednocení je realizováno přebarvením nalezeného objektu stejnou barvou (identifikačním číslem) jako původní objekt. Vzdálenost prohledávání $l$ musí být tak velká, aby zahrnula všechny sousední segmenty, a zároveň tak malá, aby nezasahovala do segmentů jiných znaků. V našem případě je tato hodnota zvolena jako hodnota šířky segmentu aktuálního objektu. Výsledek sjednocení objektů můžeme vizuálně posoudit na obrázku 3.14.



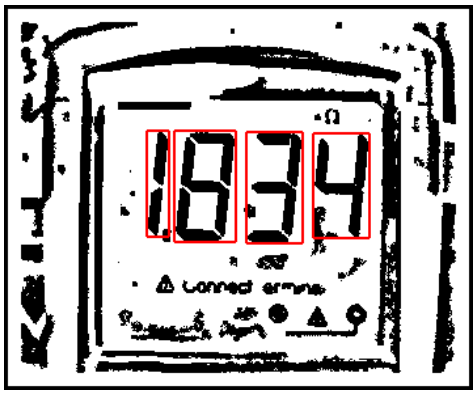
Obrázek 3.13: Procházení hranice objektů (červeně)

Obrázek 3.14: Sjednocení objektů - vlevo originál, vpravo výstupní obraz

### 3.4.3 Nalezení znaků

V tomto kroku určíme pozice všech sedmisegmentových znaků. Protože bude naše lokalizace založená na testování shody objektů, je nutné, aby se číselný údaj na měřicím přístroji skládal minimálně ze dvou znaků (tento předpoklad je pro většinu měřicích přístrojů splněn). Maximální počet znaků není nijak omezen. Při hledaní těchto znaků budeme vycházet z toho, že všechny hledané znaky mají přibližně stejnou výšku a přibližně shodnou šířku jednotlivých segmentů. Dále budeme předpokládat, že zachycený obraz není vůči měřicímu přístroji nijak výrazně otočen a hledané znaky se tak nacházejí vedle sebe, tj. mají shodnou vertikální pozici v obraze.

V navrhnutém postupu testujeme shodu všech možných dvojic objektů. Shodu testujeme vzhledem ke třem parametrům – výšce objektu, šířce jednotlivých segmentů a vertikální pozici v obraze. Při testování těchto parametrů je třeba nastavit určitou rezervu. Jestliže dvojici objektů vyhodnotíme jako shodnou, přidáme ji na seznam ekvivalentních objektů. V další fázi sjednotíme dvojice ekvivalentních objektů ze seznamu do odpovídajících množin. Protože se v zachyceném obraze může nacházet více číselných údajů, bude pravděpodobně vzniklých množin více než jedna. V poslední fázi proto z těchto množin vybereme množinu s největší výškou jednotlivých objektů. Předpokládáme při tom, že žádný číselný údaj na měřicím přístroji není větší než měřený číselný údaj. Objekty v této množině pak seřadíme podle horizontální pozice v obraze. Výsledkem je tak posloupnost objektů odpovídající hledaným znaků. Nalezení a označení výsledných znaků ilustruje obrázek 3.15.



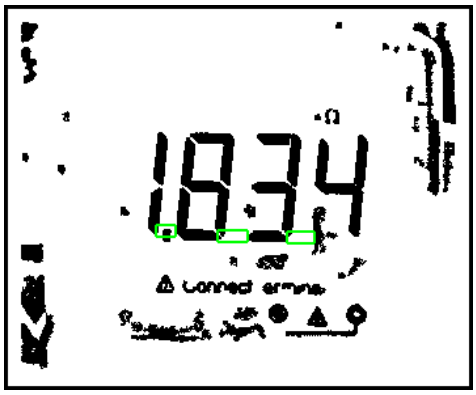

Obrázek 3.15: Nalezení pozice znaků (červeně)

### 3.4.4 Detekce desetinné tečky

Poté co určíme pozici sedmisegmentových znaků, přistoupíme k detekci desetinné tečky. Je třeba poznamenat, že desetinná tečka nemá žádné charakteristické rysy a vlastní tvar může být pro různé měřicí přístroje odlišný. Určení pozice desetinné tečky je tak velice problematické. Objekt odpovídající desetinnou tečce budeme hledat ve výřezech mezi jednotlivými nalezenými sedmisegmentovými znaky (obrázek 3.16). Za desetinnou tečku vyhodnotíme takový objekt, který splňuje následující podmínku

$$(p > 0.7) \text{ AND } (p < 1.8) \text{ AND } (l > 0.7s) \text{ AND } (l < 1.5s) \text{ AND } (A > 0.65s^2)$$

kde $p$ je poměr mezi výškou a šířkou objektu, $l$ šířka objektu, $s$ tloušťka segmentu předchozího znaku a $A$ plocha objektu (tj. počet nenulových pixelů objektu). Informaci o pozici desetinné tečky využijeme při sestavení výsledné klasifikované hodnoty.



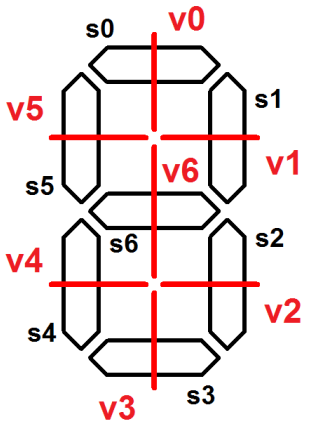

Obrázek 3.16: Předpokládané oblasti desetinné tečky (zeleně)

## 3.5 Klasifikace objektů

V tomto kroku se snažíme identifikovat nalezené objekty, tj. zařadit každý objekt do jedné z možných klasifikačních tříd. Většina klasifikačních metod je založená na vytvoření souboru měřených veličin (tzv. příznaků), které dostatečně popisují daný objekt a podle nichž objekt zařadíme do konkrétní třídy. Posloupnost těchto příznaků tvoří tzv. vektor příznaků. V našem případě je nejvýhodnější jako vektor příznaků zvolit zastoupení jednotlivých segmentů ve znaku. Jestliže správně detekujeme tyto segmenty, dokážeme jednoznačně určit, o jaký znak se jedná.

Každý sedmisegmentový znak na měřicím přístroji, může reprezentovat deset možných hodnot $0, 1, 2, \ldots, 9$. Definujeme proto množinu klasifikačních tříd $C = \{0, 1, 2, \ldots, 9\}$ (obrázek 3.17). Protože se sedmisegmentový znak skládá ze sedmi segmentů, vektor příznaků bude mít sedm prvků $\mathbf{v} = (v_0, v_1, v_2, \ldots, v_6)$, kde prvek $v_i$ představuje zastoupení segmentu $s_i$ ve znaku. Dále definujeme, že $v_i = 1$, pokud je segment $s_i$ v objektu přítomen, jinak $v_i = 0$. Jednotlivé segmenty detekujeme analýzou výřezu objektu ve středu předpokládané pozice segmentu (obrázek 3.18). Speciálním případem je znak 1, který se od ostatních znaků liší svou šířkou a předpokládané pozice segmentů by tak byly vyhodnoceny špatně. Tento znak však snadno rozpoznáme podle specifického poměru mezi výškou a šířkou objektu.


Obrázek 3.17: Klasifikační třídy




Obrázek 3.18: Detekce segmentů ve znaku

Navrhnutý klasifikátor potom můžeme zapsat následujícím pseudokódem

$$
\begin{aligned}
\textbf{if} \quad & \mathbf{v} = (1,1,1,1,1,1,0) \quad \textbf{then} \quad \text{znak} \leftarrow 0 \\
\textbf{elseif} \quad & \mathbf{v} = (0,1,1,0,0,0,0) \quad \textbf{then} \quad \text{znak} \leftarrow 1 \\
\textbf{elseif} \quad & \mathbf{v} = (1,1,0,1,1,0,1) \quad \textbf{then} \quad \text{znak} \leftarrow 2 \\
\textbf{elseif} \quad & \mathbf{v} = (1,1,1,1,0,0,1) \quad \textbf{then} \quad \text{znak} \leftarrow 3 \\
\textbf{elseif} \quad & \mathbf{v} = (0,1,1,0,0,1,1) \quad \textbf{then} \quad \text{znak} \leftarrow 4 \\
\textbf{elseif} \quad & \mathbf{v} = (1,0,1,1,0,1,1) \quad \textbf{then} \quad \text{znak} \leftarrow 5 \\
\textbf{elseif} \quad & \mathbf{v} = (1,0,1,1,1,1,1) \quad \textbf{then} \quad \text{znak} \leftarrow 6 \\
\textbf{elseif} \quad & \mathbf{v} = (1,1,1,0,0,0,0) \quad \textbf{then} \quad \text{znak} \leftarrow 7 \\
\textbf{elseif} \quad & \mathbf{v} = (1,1,1,1,1,1,1) \quad \textbf{then} \quad \text{znak} \leftarrow 8 \\
\textbf{elseif} \quad & \mathbf{v} = (1,1,1,1,0,1,1) \quad \textbf{then} \quad \text{znak} \leftarrow 9
\end{aligned}
$$

Výslednou rozpoznanou hodnotu číselného údaje měřicího přístroje získáme sestavením z jednotlivých klasifikovaných znaků a zpracováním informace o pozici desetinné tečky.

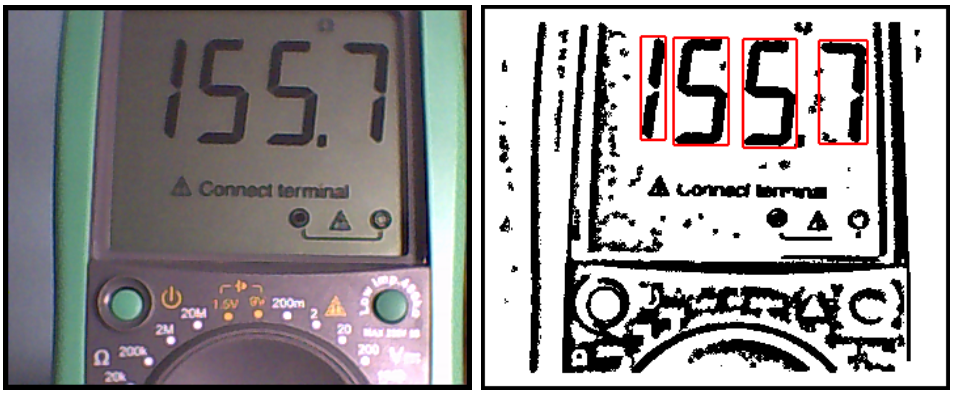
## 3.6 Výsledky návrhu v Matlabu

Pro testování navrhnutého algoritmu jsme použili dva různé měřicí přístroje – multimetr Voltcraft VC220 a multimetr Solid RE830D. Oba přístroje se mírně liší charakterem a velikostí zobrazovaných sedmisegmentových znaků. Pro určení úspěšnosti rozpoznání číselných údajů na měřicích přístrojích bylo použito 100 testovacích snímků pro každý měřicí přístroj. U přístroje Voltcraft VC220 program správně určil pozici znaků, klasifikoval všechny znaky a detekoval desetinnou tečku pro 92 snímků. Pro měřicí přístroj Solid RE830D bylo stejným způsobem správně rozpoznáno 89 snímků. Dosaženou úspěšnost v rozpoznávání shrnuje tabulka 3.2.

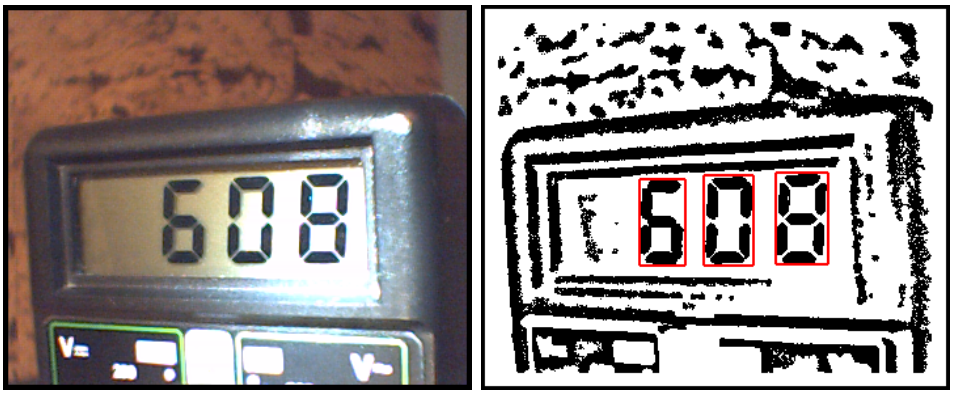
| Měřicí přístroj | Počet testovaných hodnot | Počet číslic na displeji | Přesnost rozpoznávání |
|---|---|---|---|
| Voltcraft VC220 | 100 | 3.5 | 92% |
| Solid RE830D | 100 | 3.5 | 89% |

Tabulka 3.2: Výsledky rozpoznávání

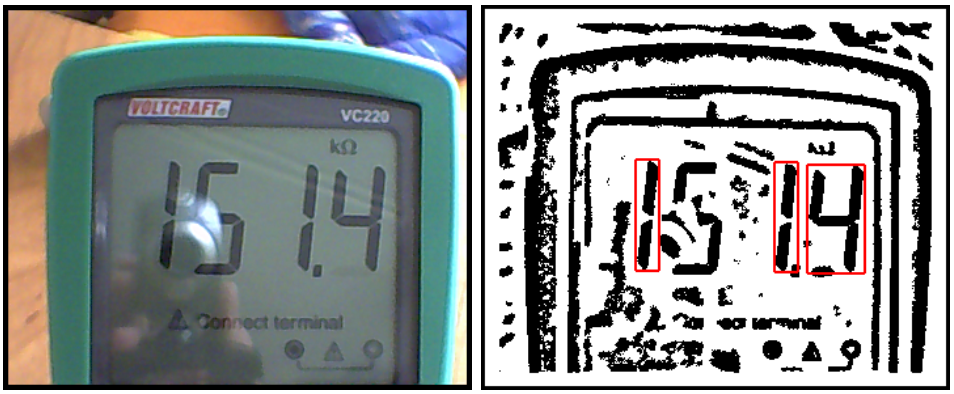
Příklady správně a nesprávně rozpoznaných měřených hodnot ilustrují obrázky 3.19 až 3.23. Doba výpočtu algoritmu v Matlabu závisí na konkrétním vstupním obrázku a pohybuje se průměrně kolem 25 sekund. Je však nutné poznamenat, že v této fázi jsme se nezabývali žádnou časovou optimalizací, ale pouze jsme se snažili ověřit funkčnost navrhnutého algoritmu. Program navíc obsahuje řadu grafických metod pro ilustraci procesu rozpoznávání a snadnější ladění, které ve výsledném zařízení nebudou.



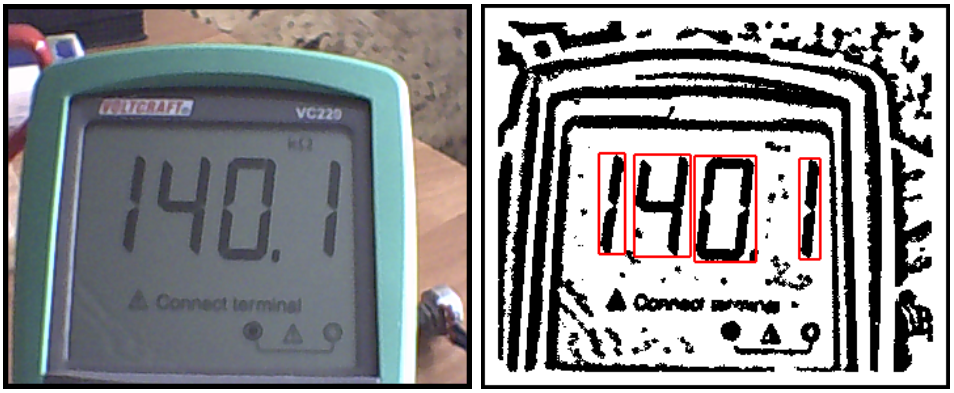

Obrázek 3.19: Příklad správně rozpoznaného snímku - rozpoznaná hodnota 155.7



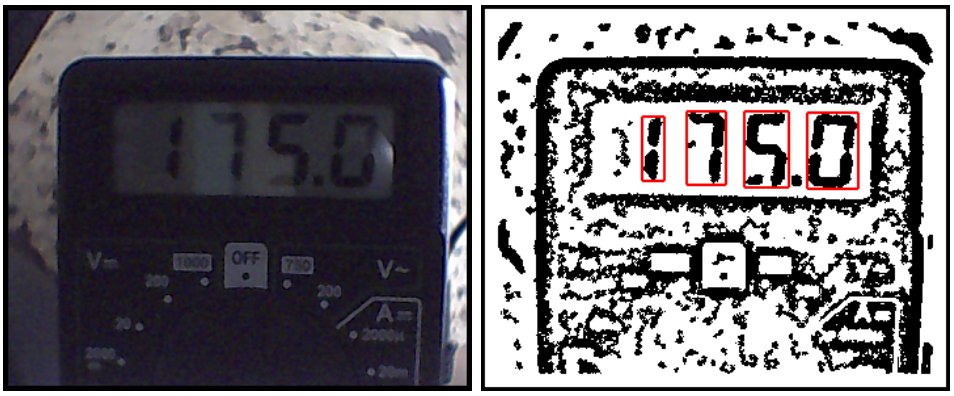

Obrázek 3.20: Příklad správně rozpoznaného snímku - rozpoznaná hodnota 608.0




Obrázek 3.21: Příklad špatně rozpoznaného snímku - nenalezen druhý znak vlivem odrazu předmětu v displeji, rozpoznaná hodnota 11.4




Obrázek 3.22: Příklad špatně rozpoznaného snímku - nesprávně detekovaná desetinná tečka, rozpoznaná hodnota 1.401




Obrázek 3.23: Příklad špatně rozpoznaného snímku - chybná klasifikace třetího znaku vlivem nízkého kontrastu, rozpoznaná hodnota 17X.0

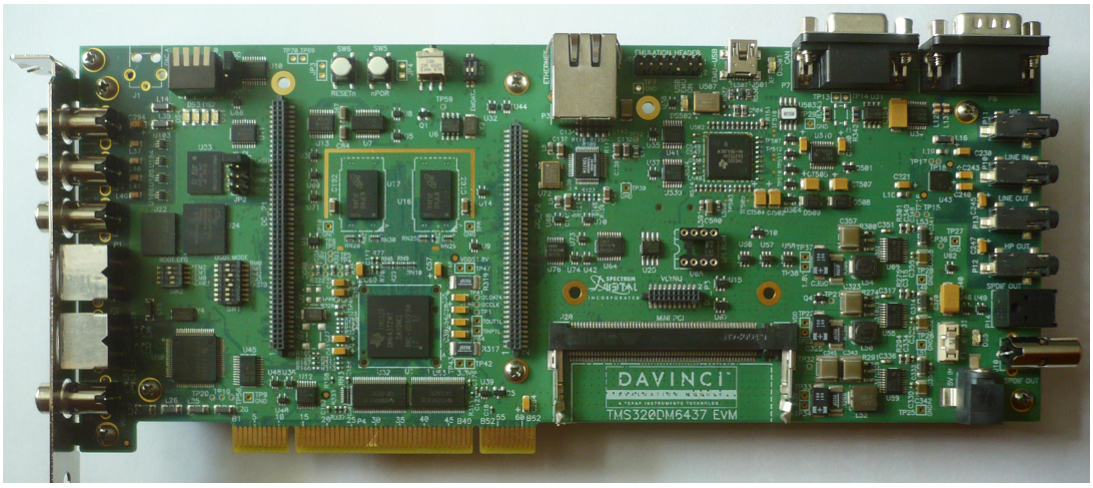
# Kapitola 4

# Implementace

V této kapitole představíme zvolené zařízení pro otestování našeho rozpoznávání v reálném čase a popíšeme konkrétní implementaci algoritmu. Na závěr se budeme zabývat časovou optimalizací programu.

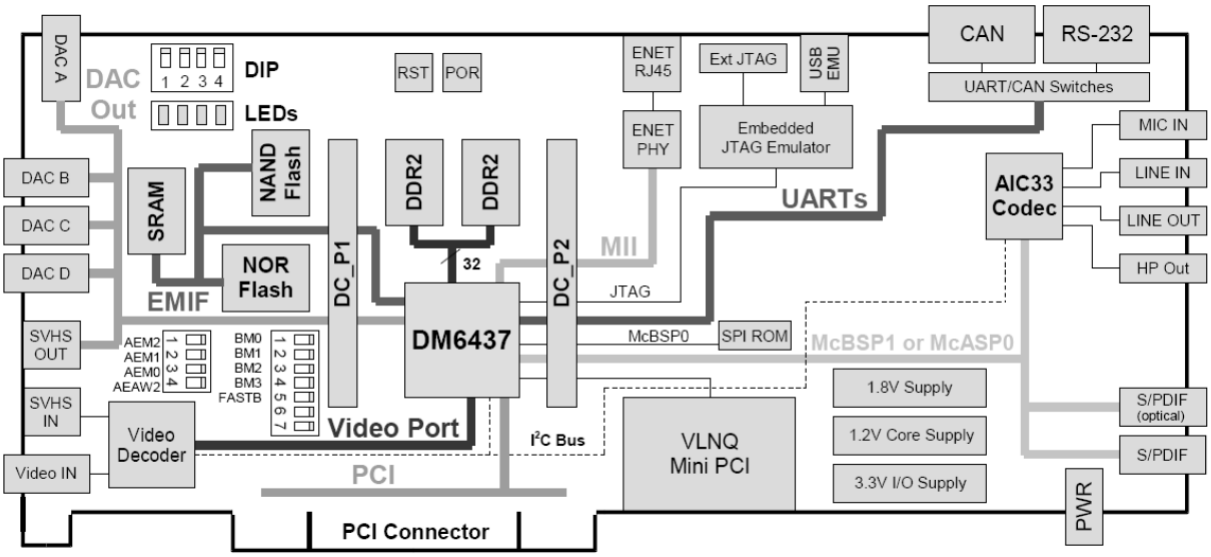
## 4.1 Seznámení s procesorem

Jako zařízení, na kterém ověříme funkčnost našeho návrhu v reálném čase, jsme zvolili vývojový modul TMS320DM6437 EVM s digitálním signálovým procesorem s technologií DaVinci od firmy Texas Instruments (obrázek 4.1). Procesory s technologií DaVinci poskytují velký výpočetní výkon a jsou tak určené pro video aplikace v oblastech počítačového vidění, robotiky, bezpečnostní techniky a dopravy.




Obrázek 4.1: Vývojový modul TMS320DM6437 EVM

Tento vývojový modul disponuje procesorem DM6437 pracujícím na frekvenci 600MHz, video dekodérem TVP5146M2, čtyřmi video DAC výstupy, dynamickou RAM pamětí DDR2 s kapacitou 128MB, statickou RAM pamětí s kapacitou 2MB, 16MB NOR Flash pamětí, 64MB NAND Flash pamětí a rozhraními USB, JTAG, UART, CAN I/O a Ethernet. Blokové schéma použitého modulu je zobrazeno na obrázku 4.2.




Obrázek 4.2: Blokové schéma vývojového modulu TMS320DM6437 EVM (převzato z [11])

Výhodou tohoto modulu je zahrnutí video dekodéru a kodéru, nemusíme se proto zabývat dekódováním a kódováním video signálu. Pro snímání obrazu je použita barevná CMOS kamera s rozlišením 720 × 480 a standardem kódování NTSC.

## 4.2 Přepis kódu

Pro vývoj a ladění programu je určeno prostředí Code Composer Studio, které je součástí vývojového modulu. Zdrojový kód můžeme vytvořit v jednom ze čtyř podporovaných programovacích jazyků – strojový kód, assembler, lineární assembler a C/C++. Vzhledem k rozsahu našeho algoritmu jsme se rozhodli pro vyšší programovací jazyk C.

Pro náš program jsme použili zdrojový kód „Video Preview“, který je volně poskytován společností Texas Instrument. V tomto kódu je implementováno v nekonečné smyčce dekódování vstupního video signálu a kódování výstupního video signálu. Vlastní algoritmus rozpoznávání umístíme do této smyčky mezi krokem dekódování a kódování. Tím zajistíme opakující se zpracování a rozpoznávání vstupního obrazu.

Protože vstupní rozlišení obrazu je zbytečně velké a zvyšuje výpočetní náročnost programu, provedeme úpravu velikosti rozlišení. Snížení rozlišení realizujeme průměrováním čtyř sousedních pixelů. Tímto řešením navíc eliminujeme náhodný šum. Ze vstupního obrazu s rozlišením $720 \times 480$ tak získáme obraz s rozlišením $360 \times 240$. Toto rozlišení je již zhruba shodné s rozlišením používaném v návrhu algoritmu v Matlabu.

Další úpravu provedeme pro metodu zabývající se sjednocením objektů. Po implementaci prahování a analýze segmentovaného obrazu jsme zjistili, že na rozdíl od předchozího návrhu jsou segmenty znaků ležící kolmo k sobě vždy spojeny (tz. jsou součástí stejné oblasti). Tento fakt je způsoben jinými optickými vlastnostmi nové použité kamery. Rozpojení nastává pouze u segmentů, které leží proti sobě. Tento případ se vyskytuje pouze u znaků 1, 7 a 0. Při hledání možných kandidátů pro sjednocení objektů proto nemusíme procházet celou hranici objektu, ale jen jeho horní hranici. Díky této úpravě zvýšíme robustnost metody, neboť snížíme počet možných kandidátů na sjednocení.

Jelikož bude naše zařízení pracovat v reálném čase, budeme se snažit minimalizovat výpočetní dobu jednoho cyklu rozpoznávání. Z toho důvodu jsme pro přístup do pole využili pointrové aritmetiky namísto indexace. Jestliže přistupujeme k prvkům pole pomocí pointrové aritmetiky, pro získání dalšího prvku stačí k aktuální adrese přičíst konstantu odpovídající velikosti prvku pole. Naopak pro přístup k prvkům pole pomocí indexů, musíme nejprve vynásobit index touto konstantou a ten pak přičíst k bázové adrese. První způsob přístupu do pole je tedy mnohem rychlejší. Na druhou stranu je však zápis kódu využívající tento způsob méně přehledný, což nám ale nebude vadit.

V našem programu budeme často používat dynamické seznamy. Nabízelo by se proto využít dynamické pole, které je možno během chodu dynamicky zvětšovat nebo zmenšovat. Alokace a uvolnění tohoto pole je však časově velmi náročná. Proto pro většinu seznamů použijeme pole statické, u něhož nastavíme jeho rozměry s dostatečnou rezervou.

Celý algoritmus rozpoznávání navrhnutý v kapitole 3 je rozdělen do následujících metod

**create_input_image** – Metoda pro vytvoření vstupního obrazu. Zahrnuje snížení vstupního rozlišení obrazu a uložení hodnot jasu do statického pole.

**thresholding** — Obsahuje prahování vstupního obrazu Niblackovou metodou a filtraci binárního šumu vzniklého prahováním.

**coloring** — Metoda pro identifikaci objektů algoritmem barvení.

**remove_inappropriate_objects** — Odstraní objekty, u nichž je jisté, že nemohou být částmi hledaných znaků.

**join_objects** — Metoda pro spojení objektů odpovídající jednotlivým segmentům do celého znaku.

**find_digits** — Obsahuje určení pozice sedmisegmenotvých znaků a detekci desetinné tečky. Nalezené znaky jsou vizuálně označeny v obraze.

**classify_digits** — Klasifikuje nalezené objekty a sestaví výslednou rozpoznanou hodnotu. Tuto hodnotu zobrazí ve výstupním obraze.

**convert_image_to_output** — Metoda pro vytvoření obrazu, který bude zobrazen na výstupní obrazovce.

## 4.3 Časová optimalizace

Po vytvoření našeho programu se můžeme pokusit o jeho optimalizaci. Optimalizaci nejčastěji provádíme s cíli snížit výpočetní dobu programu nebo snížit velikost paměti, potřebnou k vykonání programu. Protože má naše zařízení pracovat v reálném čase, bude naším cílem snížit výpočetní dobu, tz. pokusíme se o optimalizaci časovou.

Abychom mohli posoudit výsledek časové optimalizace, musíme nejdříve změřit dosavadní čas potřebný k vykonání programu. Prostředí Code Composer Studio neumožňuje přímo změřit časový úsek, poskytuje však informaci o počtu vykonaných hodinových cyklů. Přibližnou dobu výpočtu pak můžeme vypočítat z následujícího vztahu

$$\text{doba vypoctu} = \frac{\text{pocet cyklu}}{\text{frekvence procesoru}} \tag{4.1}$$

Tímto způsobem změříme výpočetní dobu pro všechny použité metody. Tyto hodnoty shrnuje tabulka 4.1. Poznamenejme, že uvedené hodnoty jsou pouze orientační a přesné hodnoty jsou závislé na konkrétním vstupním snímku z kamery.

Nyní můžeme přistoupit k samotné optimalizaci. Code Composer Studio nabízí několik možností optimalizace. My se rozhodneme pro automatickou optimalizaci v sekci „build options“. Máme zde na výběr typ optimalizace (časová vs. paměťová) a úroveň optimalizace. Zvolíme optimalizaci časovou s nejvyšší úrovní a provedeme kompilaci programu. Následně změříme výpočetní dobu programu stejným způsobem jako v předchozím případě bez optimalizace. Výsledek časové optimalizace můžeme posoudit z tabulky 4.2. Výpočetní dobu jednotlivých metod před optimalizací a po optimalizaci porovnává tabulka 4.3. Více informací o optimalizaci v prostředí Code Composer Studio nalezneme v [12].

| Metoda | Počet hodinových cyklů ($\times 10^3$) | Výpočetní doba pro CPU 600MHz (ms) |
|---|---:|:---:|
| `create_input_image` | 17 140 | 28.57 |
| `thresholding` | 175 988 | 293.31 |
| `coloring` | 90 800 | 151.33 |
| `remove_inappropriate_objects` | 15 086 | 25.14 |
| `join_objects` | 51 | 0.09 |
| `find_digits` | 598 | 1.00 |
| `classify_digits` | 200 | 0.33 |
| `convert_image_to_output` | 14 185 | 23.64 |
| Celkem | 314 048 | 523.47 |

Tabulka 4.1: Výpočetní doba algoritmu před optimalizací

| Metoda | Počet hodinových cyklů ($\times 10^3$) | Výpočetní doba pro CPU 600MHz (ms) |
|---|---:|:---:|
| `create_input_image` | 3 761 | 6.27 |
| `thresholding` | 17 760 | 29.60 |
| `coloring` | 62 538 | 104.23 |
| `remove_inappropriate_objects` | 8 844 | 14.74 |
| `join_objects` | 62 | 0.10 |
| `find_digits` | 539 | 0.90 |
| `classify_digits` | 650 | 1.08 |
| `convert_image_to_output` | 6 282 | 10.47 |
| Celkem | 94 352 | 157.25 |

Tabulka 4.2: Výpočetní doba algoritmu po optimalizaci

| Metoda | Výpočetní doba před optimalizací (ms) | Výpočetní doba po optimalizaci (ms) |
|---|:---:|:---:|
| `create_input_image` | 28.57 | 6.27 |
| `thresholding` | 293.31 | 29.60 |
| `coloring` | 151.33 | 104.23 |
| `remove_inappropriate_objects` | 25.14 | 14.74 |
| `join_objects` | 0.09 | 0.10 |
| `find_digits` | 1.00 | 0.90 |
| `classify_digits` | 0.33 | 1.08 |
| `convert_image_to_output` | 23.64 | 10.47 |
| Celkem | 523.47 | 157.25 |

Tabulka 4.3: Srovnání výpočetní doby algoritmu před a po optimalizaci

Z výsledků je patrné, že celková výpočetní doba programu se snížila více než třikrát. K největší optimalizaci došlo v metodě `thresholding`, ve které je implementováno prahování obrazu. Výsledná průměrná doba výpočtu jednoho cyklu rozpoznávání je tedy asi 0.16 s.

# Kapitola 5

# Experimentální výsledky

Navrhnuté zařízení pro rozpoznávání sedmisegmentových znaků jsme testovali v reálném prostředí na dvou měřicích přístrojích – multimetru Voltcraft VC220 a multimetru Solid RE830D. Správnost rozpoznávání jsme vizuálně posoudili na výstupní obrazovce, která uživatele informuje o nalezených znacích a rozpoznané hodnotě. Při snímání měřicího přístroje jsme kameru vždy nastavili tak, aby nedocházelo k přímému odrazu světla od displeje měřicího přístroje.

Testování rozpoznání každého přístroje jsme provedli pro 100 měřených hodnot. Pro každou měřenou hodnotu jsme změnili umístění měřicího přístroje vzhledem ke kameře. Správnost rozpoznání jsme hodnotili vzhledem ke čtyřem bodům

1. Nalezení pozice znaků – Nalezení prohlásíme za neúspěšné, pokud jeden nebo více znaků nebudou nalezeny nebo pokud bude jako znak vyhodnocen jeden a více objektů, které znaky nejsou.

2. Klasifikace znaků – Jestliže jeden nebo více znaků budou klasifikovány chybně, prohlásíme tento bod za nesplněný.

3. Detekce desetinné tečky – Tento bod bude splněn, pokud bude správně nalezena a vyhodnocena desetinná tečka.

4. Celkové rozpoznání - Celkové rozpoznání prohlásíme za úspěšné, jestliže budou splněny všechny tři předchozí body.

V případě testování rozpoznání měřicího přístroje Voltcraft VC220 zařízení správně nalezlo pozici všech sedmisegmentových znaků pro 95 měřených hodnot. Z těchto případů

byly znaky správně klasifikovány pro 94 hodnot a desetinná tečka byla správně detekována pro 91 hodnot. Celkové rozpoznání bylo úspěšné pro 90 měřených hodnot.

Následně jsme testovali rozpoznání na měřicím přístroji Solid RE830D. Pozice všech znaků byla správně určena pro 92 hodnot. Z těchto případů zařízení správně klasifikovalo všechny znaky pro 91 hodnot a správně detekovalo desetinnou tečku pro 86 hodnot. Pro tento přístroj bylo celkové rozpoznání úspěšné v 85 případech.

Dosažené experimentální výsledky v rozpoznávání shrnují tabulky 5.1, 5.2 a 5.3. Poznamenejme, že tyto výsledky jsou pouze orientační, protože úspěšnost rozpoznávání závisí na konkrétních podmínkách reálného prostředí.

| **Voltcraft VC220** | Nalezení pozice znaků | Klasifikace znaků | Detekce desetinné tečky | Celkové rozpoznání |
|---|---|---|---|---|
| Počet testovaných hodnot | 100 | 95 | 95 | 100 |
| Počet správně určených hodnot | 95 | 94 | 91 | 90 |
| Úspěšnost | 95% | 99% | 96% | 90% |

Tabulka 5.1: Výsledky rozpoznávání pro měřicí přístroj Voltcraft VC220

| **Solid RE830D** | Nalezení pozice znaků | Klasifikace znaků | Detekce desetinné tečky | Celkové rozpoznávání |
|---|---|---|---|---|
| Počet testovaných hodnot | 100 | 92 | 92 | 100 |
| Počet správně určených hodnot | 92 | 91 | 86 | 85 |
| Úspěšnost | 92% | 99% | 93% | 85% |

Tabulka 5.2: Výsledky rozpoznávání pro měřicí přístroj Solid RE830D

| Měřicí přístroj | Počet testovaných hodnot | Počet číslic na displeji | Přesnost rozpoznávání |
|---|---|---|---|
| Voltcraft VC220 | 100 | 3.5 | 90% |
| Solid RE830D | 100 | 3.5 | 85% |

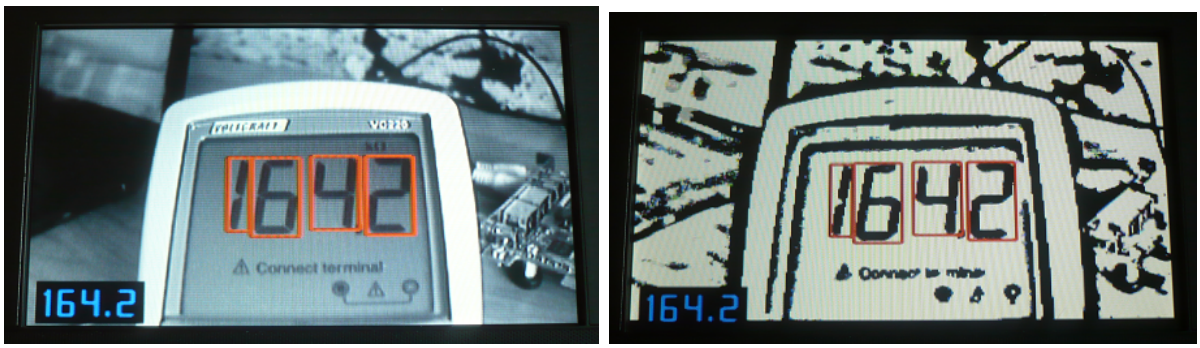
Tabulka 5.3: Shrnutí výsledků rozpoznávání

Z dosažených výsledků je patrné, že k největší chybovosti dochází při hledání pozice znaků a při detekci desetinné tečky. V prvním případě je chyba nejčastěji způsobena odrazem jiných předmětů v displeji měřicího přístroje nebo velmi špatnými podmínkami při

snímání obrazu (např. velmi nízkou intenzitou osvětlení). Takto způsobenou chybovost můžeme snížit správným umístěním snímacího zařízení a zajištěním dobrých optických podmínek.

V druhém případě je chyba způsobena faktem, že desetinná tečka nemá žádné charakteristické rysy a je tak velice obtížné ji v obraze detekovat. Vlivem nedokonalé segmentace navíc vznikají v obraze falešné objekty, které se mohou tvarem podobat hledané desetinné tečce. K obtížnosti přispívá i to, že číselný údaj desetinnou tečku může obsahovat, ale zároveň nemusí. Tuto chybovost můžeme snížit použitím dokonalejší segmentační metody.

Výsledek úspěšného rozpoznávání a výstupní obraz zařízení ilustrují obrázky 5.1 a 5.2.




Obrázek 5.1: Výsledek rozpoznávání zobrazený na výchozím obrazu (vlevo) a na obrazu po segmentaci (vpravo)


Obrázek 5.2: Výsledek rozpoznávání zobrazený na výchozím obrazu (vlevo) a na obrazu po segmentaci (vpravo)

# Kapitola 6

# Závěr

V této práci jsme se zabývali návrhem a vývojem algoritmu pro zařízení určené k rozpoznávání sedmisegmentových znaků na měřicích přístrojích. Toto zařízení by mohlo najít uplatnění převážně pro laboratorní účely. Bližší specifikace jsou popsány v první kapitole.

Druhá kapitola je věnována základním pojmům počítačového vidění a vysvětlení jednotlivých kroků procesu rozpoznávání.

V třetí kapitole jsme se zabývali návrhem algoritmu v prostředí Matlab a detailně jsme popsali použité metody. Výsledky tohoto návrhu jsme testovali pro dva různé měřicí přístroje a dosažená průměrná úspěšnost rozpoznávání se pohybuje kolem 91 %.

Navrhnutý algoritmus jsme úspěšně implementovali do vývojového modulu TMS320-DM6437 EVM ve čtvrté kapitole. Protože zařízení pracuje v reálném čase, provedli jsme v prostředí Code Composer Studio časovou optimalizaci. Před touto optimalizací se průměrná výpočetní doba jednoho cyklu rozpoznávání pohybovala kolem 0.52 s, po optimalizaci jsme tuto hodnotu snížili na 0.16 s.

Dosažené výsledky implementovaného rozpoznávání jsme zhodnotili v páté kapitole. Průměrnou úspěšnost jsme opět testovali na dvou přístrojích a tato hodnota se pohybuje kolem 87 %. K největší chybovosti dochází při hledání pozice znaků a při detekci desetinné tečky. Chybovost v prvním případě je nejčastěji způsobena odrazem jiných předmětů v displeji snímaného měřicího přístroje a můžeme ji snížit vhodným umístěním kamery. V druhém případě je chybovost způsobena obtížnou charakteristikou desetinné tečky. Naopak velké úspěšnosti dosahuje navrhnutá metoda klasifikace znaků, která je založená na určení jednotlivých segmentů, z nichž jsou znaky složeny.

Výsledkem této práce jsou funkční programy v prostředí Matlab a v jazyce C.

# Literatura


[1] V. Hlaváč, M. Šonka: *Počítačové vidění*, Grada, 1992, ISBN 80-854-2467-3

[2] V. Hlaváč, M. Sedláček: *Zpracování signálů a obrazů*, ČVUT, 2005, ISBN 80-010-3110-1

[3] Z. Kotek, V. Mařík: *Metody rozpoznávání a jejich aplikace*, Academia, 1993, ISBN 80-200-0297-9

[4] J. Žára: *Moderní počítačová grafika*, Computer Press, 1998, ISBN 80-722-6049-9

[5] Ch. M. Bishop: *Pattern Recognition and Machine Learning*, Springer Science Business Media, 2006, ISBN 03-873-1073-8

[6] E. R. Davies: *Machine Vision: Theory, Algorithms, Practicalities*, Morgan Kaufmann, 2005, ISBN 01-220-6093-8

[7] R. O. Duda, P. E. Hart, D. G. Stork: *Pattern Classification*, Wiley, 2001, ISBN 04-710-5669-3

[8] D. A. Forsyth: *Computer Vision: A Modern Approach*, Prentice-Hall, 2003, ISBN 01-319-1193-7

[9] D. G. Stork: *Computer Manual in MATLAB to Accompany Pattern Classification: Theory, Algorithms, Practicalities*, Wiley, 2004, ISBN 04-714-2977-5

[10] K. Chinnasarn, Y. Rangsanseri, P. Thitimajshima, *Removing Salt-and-Pepper Noise in Text/Graphics Images*, IEEE Computer Society Press, 1998

[11] Texas Instrument Inc.: *TMS320DM6437 DVDP Getting Started Guide*, 2007

[12] Texas Instrument Inc.: *Code Composer Studio v3.0 Getting Started Guide*, 2004

# Příloha A

# Seznam použitého software

- Matlab R2010a
- Code Composer Studio v3.3
- Flashburn DSK 3.11
- Texmaker 3.1