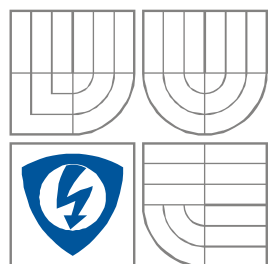


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ  
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH  
TECHNOLOGIÍ  
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION  
DEPARTMENT OF CONTROL AND INSTRUMENTATION

MAPV

## PROJEKT 2 – ROZPOZNÁVÁNÍ REZISTORŮ

AUTOR PRÁCE

Bc. Milan Kořínek  
Bc. Michal Šimberský

VEDOUCÍ PRÁCE  
SUPERVISOR

Ing. Číp Pavel

BRNO 2013

# Obsah

1 Úvod.....	3
2 Detekce rezistorů v obraze.....	5
2.1 Příprava vstupního obrazu.....	5
2.2 Porovnání oblasti se šablonkou.....	5
2.3 Optimalizace algoritmu.....	7
3 Dekódování hodnoty rezistoru.....	9
3.1 HSV MODEL.....	9
3.2 Popis funkce ReadColorCode.....	10
4 Umístění popisků ve scéně.....	13
4.1 Algoritmus Kroužící orel.....	13
4.2 Funkce Kroužící orel.....	14
5 GUI Aplikace.....	16
6 Závěr.....	17
7 Literatura.....	18

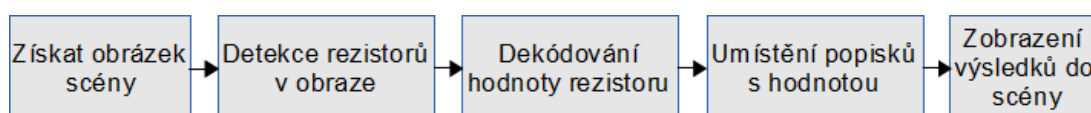
# 1 ÚVOD

V průmyslové praxi se čím dál častěji využívá vizuální kontrola výrobků pomocí kamer. Toto lze využít například pro kontrolu potisků, detekce přítomnosti výrobku a jeho natočení, odhalení viditelných vad atd.

V našem projektu jsme měli za úkol navrhnout algoritmy, které detekují rezistor ve zvolené scéně a poté dekodují jeho barevný kód. Hodnoty rezistorů se poté měly vypsat do obrázku se scénou.

V prvním kroku jsme museli vytvořit vlastní galerii se rezistory, na které jsme odladili navržené algoritmy. Pro snímání scény jsme použili kameru ImagingSource DFK 41BU02. Kamera byla umístěna kolmo nad deskou s rezistory, čímž velikost všech rezistorů byla stejná v zachycené scéně. Ze začátku jsme měli k dispozici jedno halogenové světlo, které jsme umístili těsně na pravé straně kamery. Tímto vznikl jednostranný stín u rezistoru. Pro jeho částečnou eliminaci jsme použili druhé halogenové světlo, které jsme umístili nalevo od kamery. Jako podklad pod rezistory jsme použili bílou podložku, se kterou se nám nejlépe podařilo vysegmentovat rezistory. Bílý proužek na rezistoru byl tmavší než bílé pozadí, a tedy se rezistor po segmentaci nerozpadl na dvě části. Po vytvoření scény jsme vytvořili galerii o velikosti přes 50 fotek.

Na obrázku 1 je blokové schéma programu Resistor Reader, který jsme vytvořili za výše popsáním účelem. V prvním kroku je třeba získat buď obrázek scény z kamery nebo jej lze načíst z připravené galerie. V druhém kroku je nutné nalézt samotné rezistory ve scéně. Výstupem je seznam rezistorů. Když již máme nalezené rezistory, můžeme se pokusit dekodovat hodnotu rezistoru z jeho barevného kódu. Ve čtvrtém kroku je třeba nalézt vhodné umístění popisků s hodnotou rezistorů ve scéně, tak aby se jednak samotné popisky nepřekrývaly, nebo aby nebyly umístěny přes rezistor. Nakonec jsou výsledky vykresleny a zobrazeny uživateli.



Obrázek 1: Blokové schéma programu - Resistor Reader.

Všechny parametry rezistorů jsou uloženy ve struktuře. Jednotlivé funkce si pouze předávají pole těchto struktur. Struktura je popsána v tabulce 1.

*Tabulka 1: Popis struktury rezistoru.*

<b>Parametr</b>	<b>Řešeno ve funkci</b>	<b>Poznámka</b>
resistor	ResistorReader.m	Obrázek rezistoru v horizontální poloze
value	ReadColorCode.m	Hodnota rezistoru
center	ResistorReader.m	Souřadnice středu [x y] rezistoru v obraze
angle	ResistorReader.m	Úhel natečení rezistoru
mask	ResistorReader.m	Maska rezistoru
boundary	ResistorReader.m	Ohraničení rezistoru, souřadnice rohů [X Y]
lblPos	LblSpinningEagle.m	Pozice pravého dolního rohu [x y] popisku
lblRor	LblSpinningEagle.m	Úhel natočení popisku

## 2 DETEKCE REZISTORŮ V OBRAZE

Pro detekci rezistorů v obraze slouží funkce `DetectResistor.m`, jejíž hlavička je ve výpisu 1. Vstupem funkce je barevný obrázek scény a šablona rezistoru. Výstupem funkce pole struktury rezistorů. Vývojový diagram funkce je na obrázku 5.

### Kód 1: Funkce *DetectResistor*

```
function [resistors] = DetectResistors(imCol, template)
% Function detect resistor in image
% Input:
% - imCol - color image
% - template - template of resistor
% Output:
% - resistors - list of resistor structure
```

### 2.1 Příprava vstupního obrazu

Před samotnou detekcí rezistorů je třeba si obrázek scény upravit. Nejdříve je převeden do barevného prostoru HSV. Dále je používána pouze složka S (saturace), se kterou jsme dostali nejlepší výsledky segmentace, jelikož se podařilo odstranit i stíny mezi rezistory, která byly blízko sebe. Obrázek se tedy v dalším kroku vysegmentuje a dostaneme binární obraz. Pomocí morfologických operací jsme odstranili malé oblasti a uzavřely díry.

V dalším kroku jsme pomocí funkce `regionprops` našli všechny oblasti v obrázku. Nalezené oblasti jsme prošli a ponechali pouze ty, které by mohly obsahovat rezistor/y. Rozhodovali jsme se na základě porovnání s velikostí šablony rezistoru.

### 2.2 Porovnání oblasti se šablonkou

Nyní jsme ve fázi, kdy máme oblast kde by se měl nalézat rezistor. Natočení rezistoru je neznáme a tedy přikládanou šablonku musíme natáčet v rozmezí  $0^\circ - 180^\circ$ . Pro jednotlivá natočení šablony, zkoušíme šablonku přikládat do prozkoumávané oblasti ve směru x a y.

Pro každé posunutí šablony vypočítáme korelaci. Výpočet korelace je řešen ve funkci `corr2_wb` viz výpis 2. Vstupem funkce je výřez oblasti o stejné velikosti jako má šablona, samotná šablona a maska rezistoru šablony, která je nutná při natočení šablony, aby se nepočítala korelace z prázdného prostoru (kde by se mohl nacházet další rezistor). Šablona rezistoru je na obrázku 2.

Šablonka je rozdělena na tři zóny. Černá zóna odpovídá místu, kde by mělo být prázdné místo. Šedá a bílá oblast odpovídá rezistoru. Bílá oblast je oblast s velkou citlivostí (dvojnásobná citlivost oproti šedé zóně) na díry. Čím větší díra, tím výrazněji se sníží výsledná korelace. Toto řešení je nutné pokud se v testované oblasti nalézají více než jeden rezistor.

#### Kód 2: Funkce `corr2_wb`

```
function R = corr2_wb(image, template, activeZone)
% Compute correlation between image and template
% Input:
% - image - image for correlation
% - template - template for correlation
% - activeZone - only from this area will be compute correlation
% Output:
% - R - <0, 1>, for 0 - mismatch, for 1 - total match
```

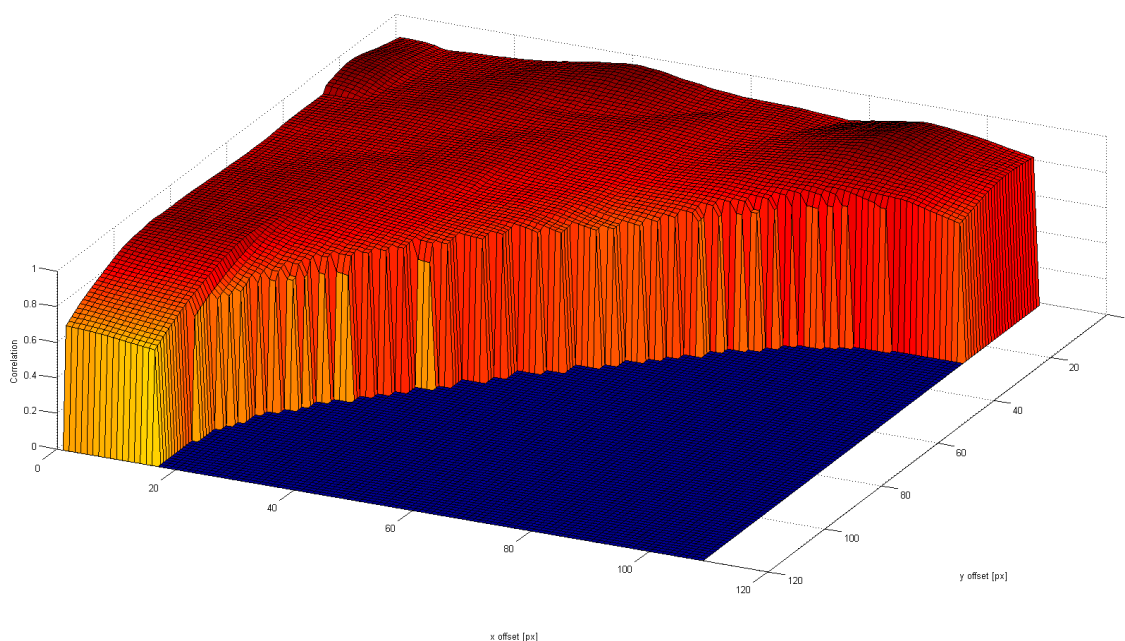


Obrázek 2: Šablonka rezistoru.

Pro každý posuv  $x, y$  šablonky se pouze ukládá nejlepší dosažená korelace a úhel při kterém to bylo dosaženo. Tedy nejsou uloženy výsledky korelace pro všechny úhly, což má za následek, že není možné detekovat oba rezistory v případě, kdy jsou přeloženy přes sebe do kříže. Nalezne se pouze jeden.

Nakonec je třeba analyzovat výslednou matici korelací. Pro lepší názornost je ukázána v 3D grafu na obrázku 3 a byla vytvořena pro oblast s rezistory na obrázku 4. V grafu jsou v zadní části vidět tři maxima odpovídající třem rezistorům pro příslušné posuvy  $x, y$ . Nevýhodou je vysoká korelace, i když není šablonka přiložena přesně a tedy není možné snadno zjistit z tohoto grafu kolik rezistorů se v obraze ve skutečnosti nachází. Z tohoto důvodu funkce pouze nalezne jeden rezistor a nesnaží se o další analýzu. Pokud bychom přece jenom chtěli zjistit všechny rezistory, pak by bylo zapotřebí vypočítat korelaci na jiném principu např. pomocí hran, obecné Houghovy transformace.

Pro oblasti s jedním rezistorem je funkce spolehlivá a bez potíží nalezne rezistor, jenž je lépe či hůře vysegmentovaný.



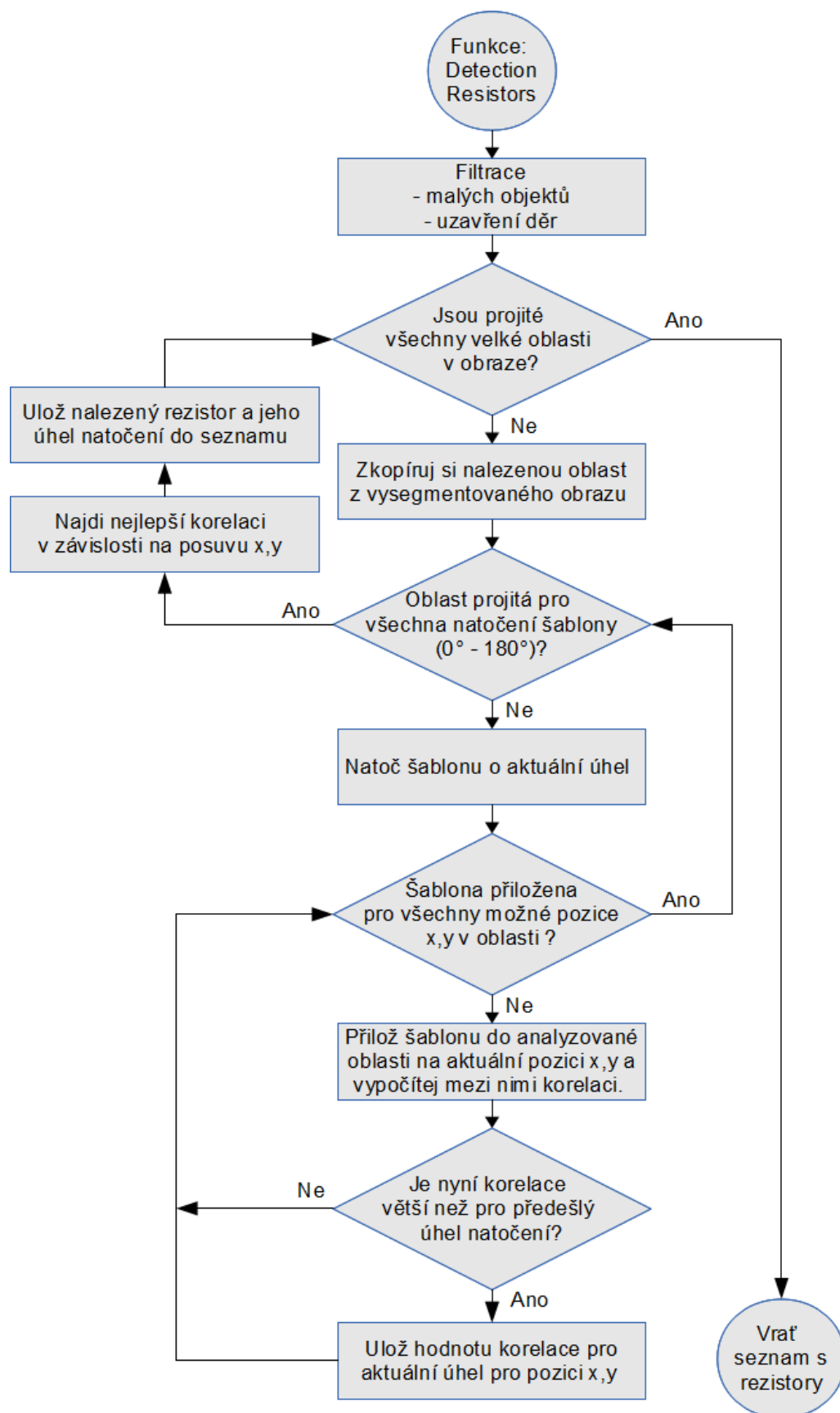
Obrázek 3: Matice korelací pro x,y posuv.



Obrázek 4: Testovací oblast s rezistory.

## 2.3 Optimalizace algoritmu

Problém výše uvedeného algoritmu je rychlost. Nejnáročnější operací je samotná rotace šablony pomocí funkce `imrotate`. V průměru trvalo nalezení úhlu natočení jednoho rezistoru okolo 2 - 3 s. Algoritmus jsme optimalizovali pomocí dynamického kroku úhlu, tak že pokud je rozměr šablony při aktuálním natočení větší než zkoumaná oblast, pak se krok z původních 2° zvětší na 10°. Jeli oblast malá, tedy je zde maximálně jeden rezistor, a zároveň i korelace je nízká, pak můžeme rovnou zkusit jiný úhel natočení s krokem 10°. Detekce jednoho rezistoru po optimalizaci trvá pouze kolem 0,5 s.



Obrázek 5: Vývojový diagram pro funkci *DetectResistors.m*

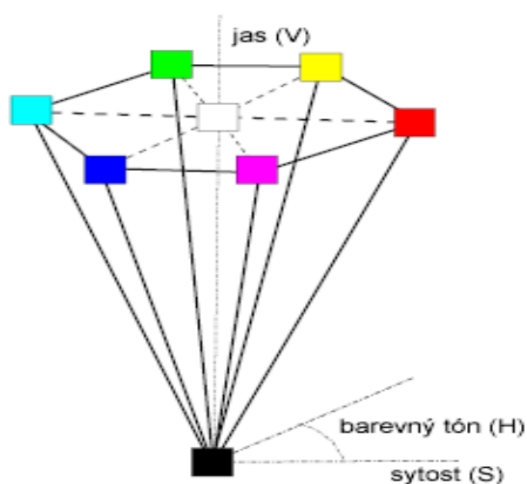


### 3 DEKÓDOVÁNÍ HODNOTY REZISTORU

Po nalezení rezistoru v obraze můžeme přistoupit k dekódování barevného kódu reprezentující hodnotu odporu rezistoru. Hledání barev v obraze probíhá v barevném modelu HSV, který lépe odpovídá popisu barev, na který je člověk zvyklý.

#### 3.1 HSV MODEL

Barevný model HSV, stejně jako barevný model RGB, se skládá ze tří složek. Jednotlivé složky zde již ale nenesou informaci o základních barvách, ale mají jiný význam. Složka **H** (Hue) udává barevný tón, složka **S** (Saturation) sytost a složku **V** (Value), která představuje jas. [2]

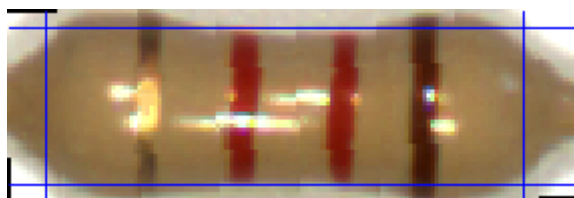


Obrázek 6.: HSV model [1].

## 3.2 Popis funkce ReadColorCode

```
function[ValueOfResistor] = ReadColorCode(AreaWithResistor)
% Funkce čte hodnotu barevného kódu na rezistoru
% Input:
% - AreaWithResistor - color image in RGB
% Output:
% - ValueOfResistor - hodnota odporu rezistoru jako string,
%                   vrátí XX, když neurčí hodnotu
```

Vstupem do funkce je obraz s nalezeným a vhodně natočeným rezistorem (barevné proužky co nejvíc vertikálně). Obraz je následně vyprahován a je nalezena vnitřní oblast rezistoru viz obrázek 7. V této oblasti se hledají barevné proužky reprezentující barevný kód hodnoty odporu rezistoru.



Obrázek 7.: Ukázka nalezené vnitřní oblasti rezistoru.

Funkce je psána pro hnědé rezistory se 4 proužkovým kódem. Vzhledem k tomu, že čtvrtý proužek udává přesnost a v souboru testovacích rezistorů se nachází pouze ty, které mají přesnost označenou zlatým proužkem, jsou hledány pouze první tři barevné proužky.

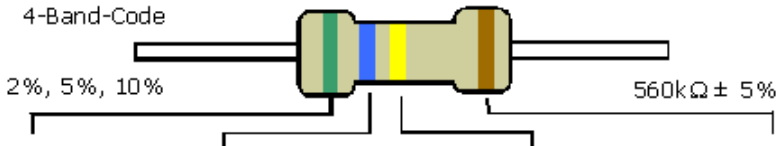
Jádrem funkce je rozhodování podle složek H, S a V o jakou barvu na daném pixelu se jedná. Interval H, S, V pro jednotlivé proužky byly založeny na základě jejich statistického rozložení viz histogramy na obrázku 9. Bylo použito přes 220 rezistorů, kde z každého proužku bylo odebráno několik reprezentativních vzorků jeho barvy. Po nalezení vnitřní oblasti rezistoru se tato oblast prochází pixel po pixelu a vytváří se nová maska stejných rozměrů, kam se zapisují hodnoty nalezených barev.

Po získání masky s barvami je vzhledem k odleskům na hřbetě rezistoru, odstraněna prostřední část v šířce 10 px. V takto redukované masce se provede modus po sloupcích.

Po určení barev jaké se nacházejí v jednotlivých sloupcích masky, se odstraní šum. To jsou např. sloupce na rozhraní přechodu mezi barvou proužku a barvou pozadí, které se v řetězci barev vykytují pouze jednou – ve smyslu, že stojí osamocené.

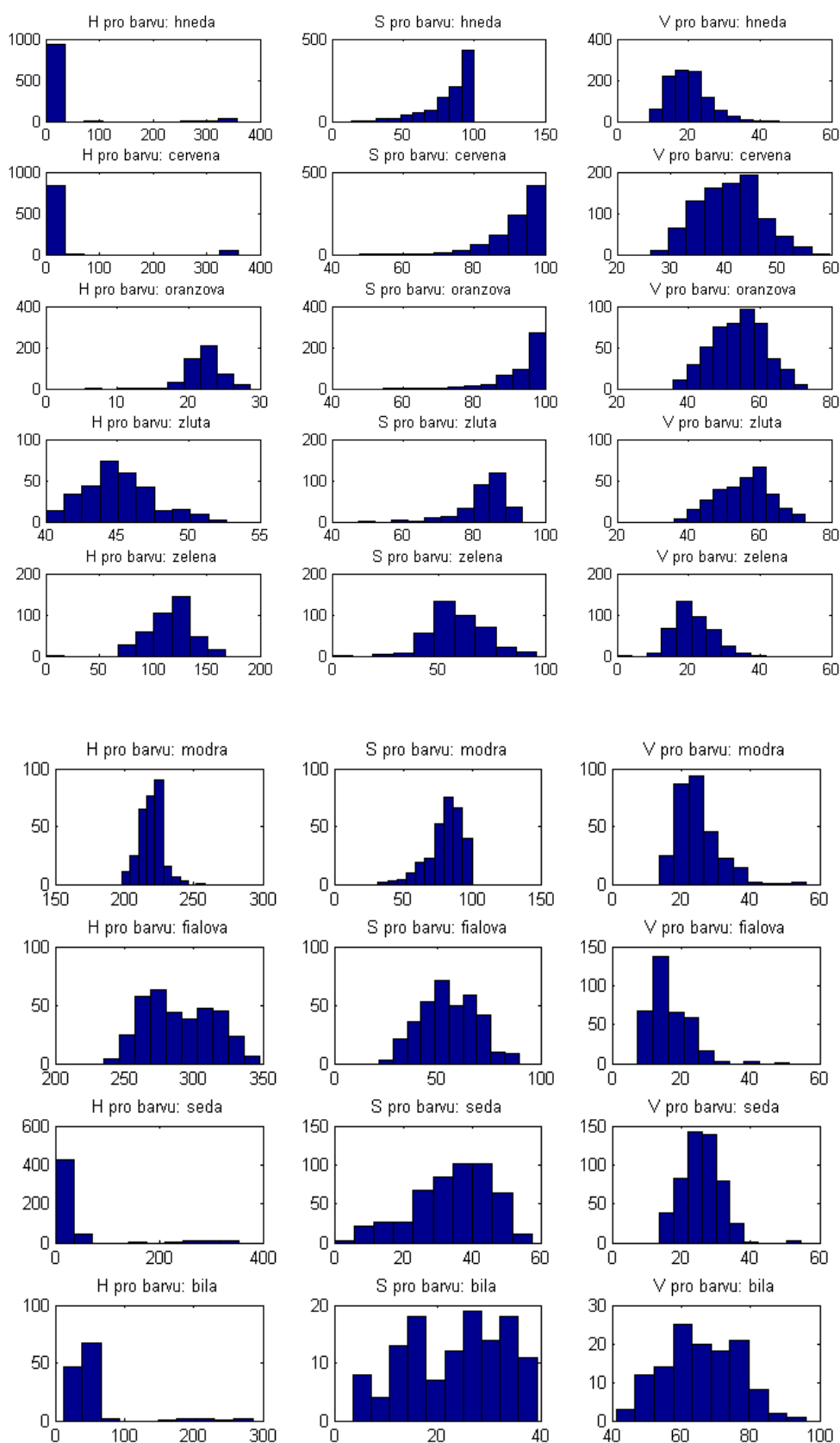
Výsledkem je řetězec čísel reprezentující barvy, které převládají v jednotlivých sloupcích. Z tohoto řetězce se následně vyjme z každé barevné oblasti jedna hodnota. Je získán řetězec ve formátu 99 X 99 X 99 X 99, kde 99 reprezentuje barvu pozadí a X reprezentuje barvu proužku. Ze získaného řetězce se poté určí hodnota odporu rezistoru podle barevného kódu.

Výstupem z funkce je hodnota odporu rezistoru v zavedeném značení ( např. 47R, 2K2, 12K,..).



COLOR	1st BAND	2nd BAND	3rd BAND	MULTIPLIER	TOLERANCE
Black	0	0	0	1Ω	
Brown	1	1	1	10Ω	± 1% (F)
Red	2	2	2	100Ω	± 2% (G)
Orange	3	3	3	1KΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100KΩ	±0.5% (D)
Blue	6	6	6	1MΩ	±0.25% (C)
Violet	7	7	7	10MΩ	±0.10% (B)
Grey	8	8	8		±0.05%
White	9	9	9		
Gold				0.1	± 5% (J)
Silver				0.01	± 10% (K)

Obrázek 8.: Barevný kód rezistorů [2].



Obrázek 9: Rozložení HSV pro jednotlivé barevné proužky.

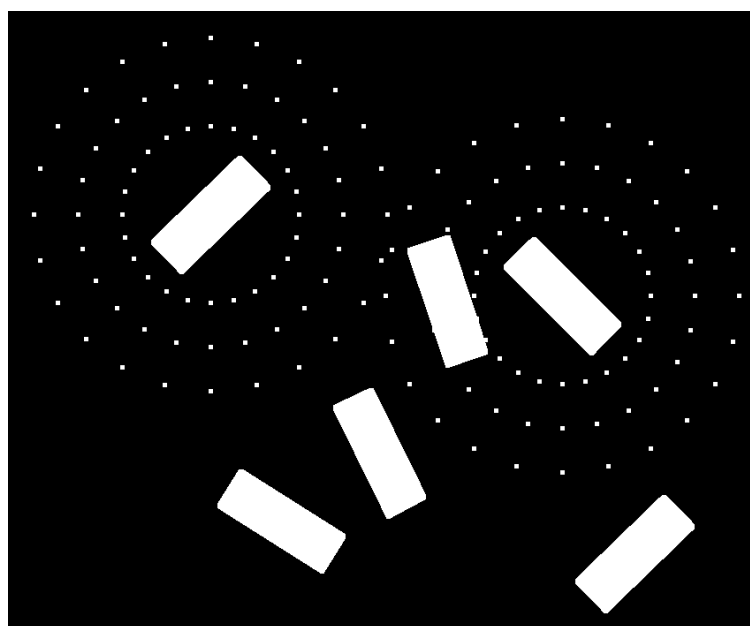
## 4 UMÍSTĚNÍ POPISKŮ VE SCÉNĚ

Nyní již známe hodnoty rezistorů, které je třeba vypsát do scény pro co nejjednodušší orientaci uživatele. Obecně by se měly popisky nacházet, co nejbližší k rezistoru. Podle normy by se měli nacházet ideálně vlevo nebo nad rezistorem. Nakonec popisky by se neměly navzájem překrývat, či by neměly překrývat objekty zájmu ve scéně. Abychom zohlednili výše popsané skutečnosti, tak jsme vymysleli algoritmus „Kroužící orel“ pro umísťování popisků do scény.

### 4.1 Algoritmus Kroužící orel

Byla implementována nejjednodušší verze algoritmu, která se snaží umístit popisek na kružnici kolem středu rezistoru. Vhodnější implementace spočítá v umístění popisku na hranici obdélníku kolem rezistoru, čímž by byl popisek umístěn blíže k rezistoru.

Na obrázku 10 je vidět způsob umísťování popisků. Bílé obdélníky jsou masky všech rezistorů ve scéně. Bílé tečky odpovídají testovaným pozicím popisku. Abychom pokud možno umísťili popisek v levé části, tak první umístění popisku je pro úhel  $270^\circ$ . Pak se postupuje po směru hodinových ručiček. Testovaná pozice vždy odpovídá pravému dolnímu rohu popisku, a tedy na pravé straně od rezistoru je možné umístit popisek až pro druhý, či třetí zkoušený poloměr kružnice. Tímto algoritmus výrazněji preferuje levou a horní oblast pro umístění popisků. V případě, že neexistuje volné místo pro umístění popisku, pak se popisek jednoduše umístí přes pravou část rezistoru.



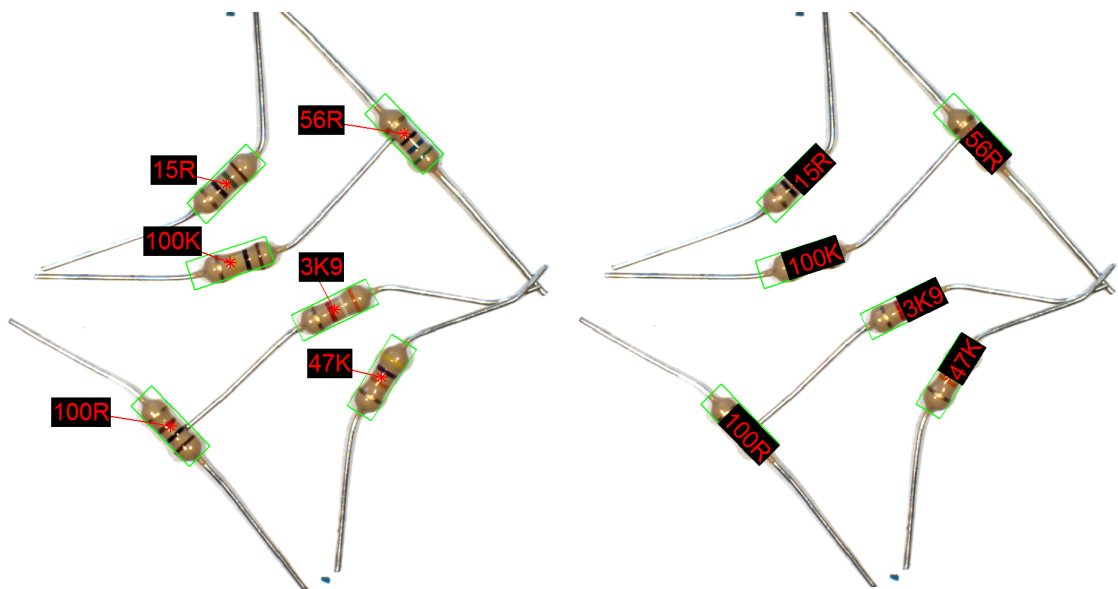
Obrázek 10: Umísťování popisků kolem rezistoru.

## 4.2 Funkce Kroužící orel

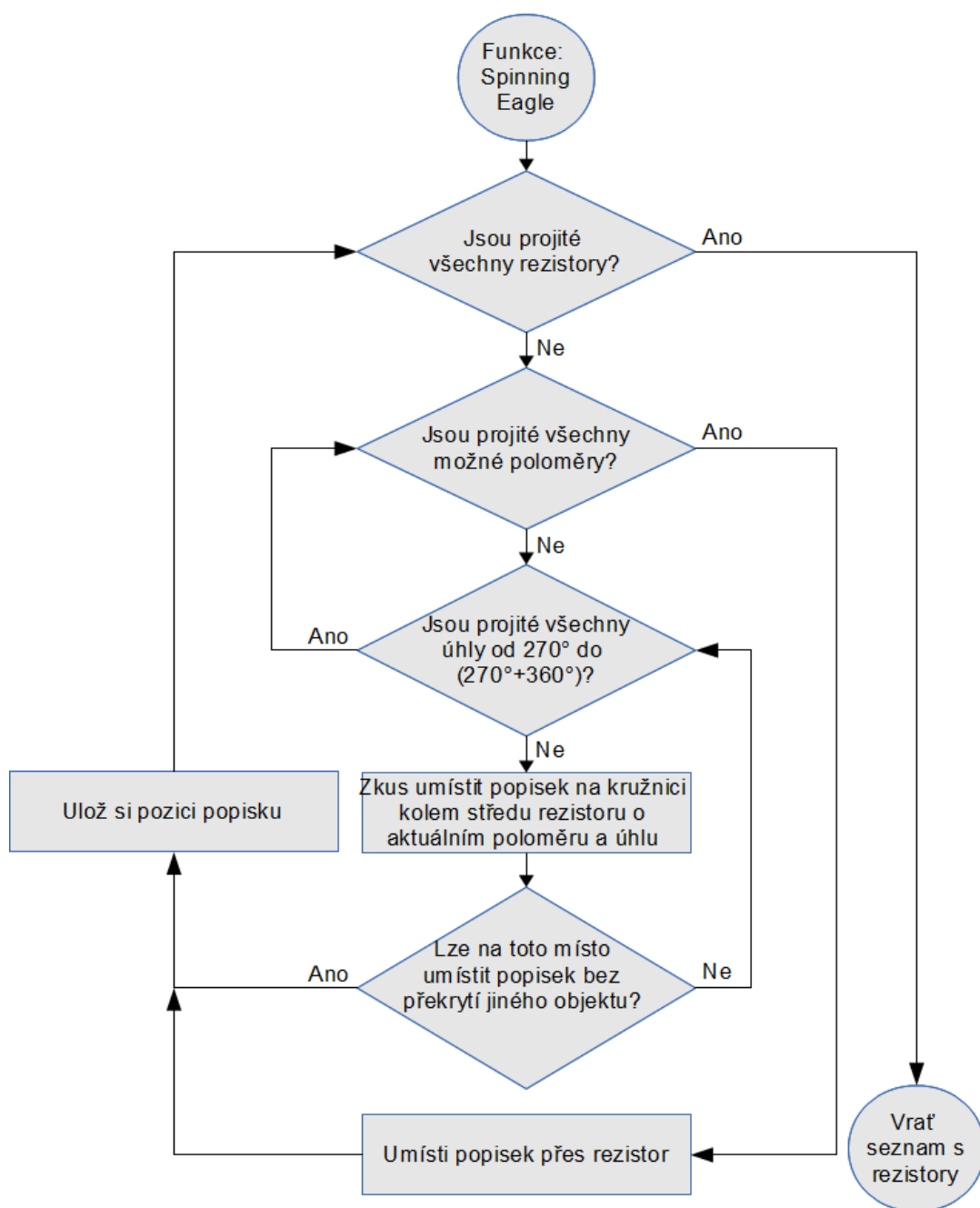
Popsaný algoritmus je implementován ve funkci `LblSpinningEagle.m` viz výpis 3. Vstupem funkce je seznam struktur s rezistory, rozměry obrazu scény, rozměry popisky a mód funkce. Mód umožňuje zvolit, jestli se popisky budou umisťovat okolo rezistorů, nebo se mají rovnou umístit přes rezistory viz 11. Vývojový diagram je na obrázku 12.

### Kód 3: Funkce `LblSpinningEagle`

```
function [out] = LblSpinningEagle(resistors, imageSize,  
    lblSize, mode)  
% Algorithm "Krouzici orel" ;), try place label around  
resistor (from  
% 270deg). If not free space, than replace resistor with  
label.  
% Input:  
% - resistors - structure with resistor parameters  
% - imageSize - size of origin image [x y]  
% - lblSize - size of label [x y]  
% - mode - 0 - labels are placed around resistors,  
%         1 - labels are placed over resistor  
% Output:  
% - out - resistor structure
```



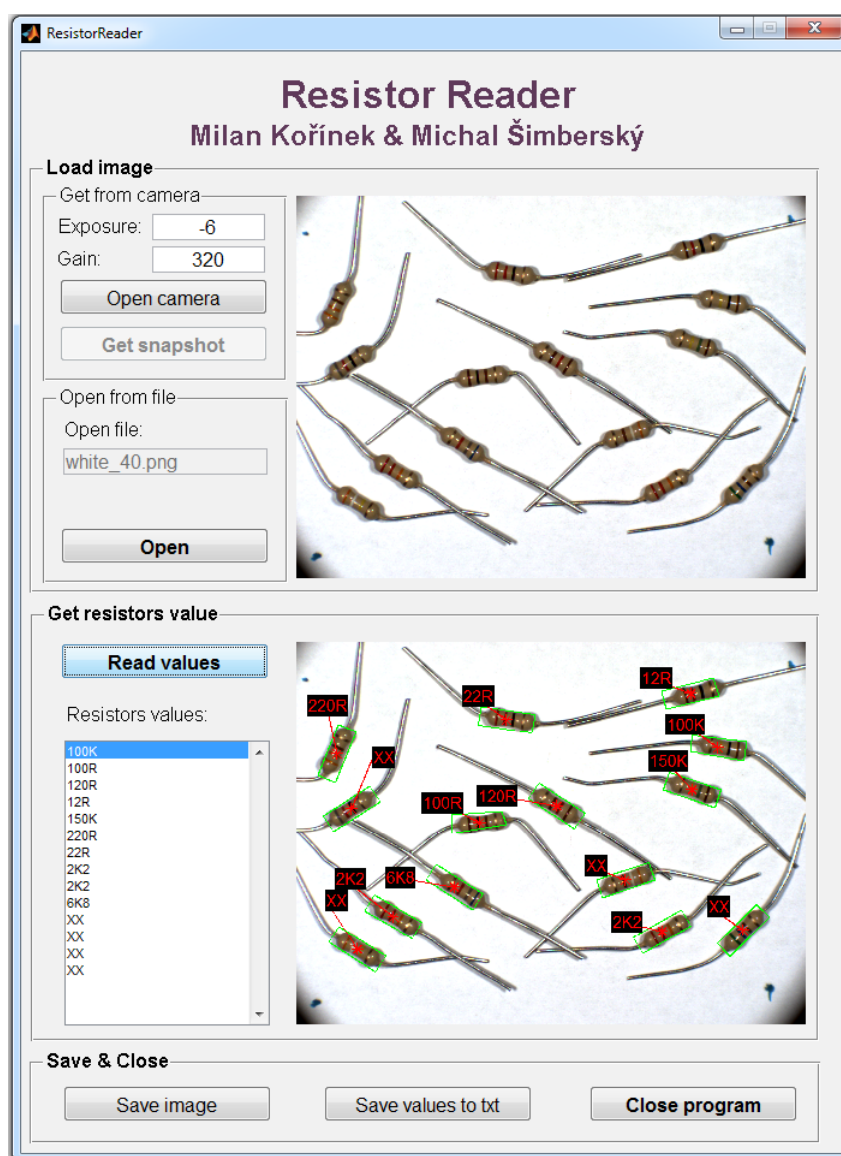
Obrázek 11: Zobrazení popisků a) vedle rezistoru, b) přes rezistor.



Obrázek 12: Vývojový diagram algoritmu Kroužící orel.

## 5 GUI APLIKACE

Pro přívětivé ovládání a nastavování programu bylo vytvořeno uživatelské rozhraní v Matlab GUI, jenž je na obrázku 13. V horní části si uživatel může buď otevřít kameru a získat z ní snímek scény, nebo si jednoduše otevřít obrázek ze souboru. Náhled načteného obrázku se zobrazí v pravé horní části. Pomocí tlačítka „Read values“ program analyzuje předložený obrázek, nalezne všechny rezistory a zjistí jejich hodnoty. Všechny přečtené hodnoty jsou jednak vypsány v ListBoxu a taktéž jsou vypsány přímo do scény vedle rezistorů. Uživatel má možnost si uložit obrázek s hodnotami rezistorů či si může uložit samostatně hodnoty do textového souboru.



Obrázek 13: Uživatelské rozhraní programu Resistor Reader.



## 6 ZÁVĚR

Před psaním algoritmů jsme museli vytvořit vhodnou scénu. Pro osvit byly použity halogenové lampy, které nejsou vhodné pro nasvícení objektů, ze kterého se mají číst barvy. Ideální osvětlení by mělo mít bílou studenou barevnou teplotu, tak aby neovlivňovalo barvy proužků na rezistoru. Dalším problémem byly stíny, zejména mezi rezistory, která zhoršovali detekce. A naopak na horní straně rezistorů vznikaly silné odlesky znemožňující v této části rozeznat barvy proužků. Řešením těchto problémy by byl difúzní stan.

Detekce rezistorů v obraze byla většinou bezproblémová až na případy, kdy jsou rezistory těsně vedle sebe a po vysegmentování tvoří jednu souvislou oblast. Algoritmus je schopný rozeznat pouze jeden rezistor ze skupiny. Abychom rozeznali i další rezistory, musela by se korelace mezi šablonkou a testovanou oblastí vypočítat jiným způsobem. Dalším problémem byla rychlost, která byla omezena omezená používanou funkcí `imrotate`. Provedli jsme optimalizace, který výrazně algoritmus urychlily. Původně trvalo nalezení úhlu natočení rezistoru okolo 2 – 3 s a po optimalizaci 0,5 s.

Při čtení barevného kódu z rezistorů jsme se potýkali s problémem, jak určit barvy podle složek H,S a V. Zatím co některé barvy ( žlutá, modrá, zelená, fialová) šlo rozlišit od ostatních bez problému, tak u dalších není rozlišení tak jednoznačné. Největším problémem se ukázaly barvy hnědá, červená a oranžová, které se svými rozsahy v H,S a V překrývaly. Volba rozsahů těchto barev je tedy kompromisem mezi jednotlivými hodnotami.

Dalším problémem při určování rozsahů barev bylo obecně prostředí, kde byly pořizovány snímky rezistorů, hlavně tedy světelné podmínky, které ovlivňovaly snímanou scénu. Jak už bylo psáno, osvětlení scény nebylo ideální a daleko vhodnější by bylo použití rovnoměrného zdroje světla. Dále by bylo vhodné umístit snímanou scénu do boxu, který by odstínil vnější světelné podmínky a tak se snímalo za stále stejných světelných podmínek.

Zjištění správné hodnoty odporu rezistoru není vzhledem k vyjmenovaným problémům 100%. Úspěšnost algoritmu se pohybuje kolem 80 %.

Projekt byl vyvíjen přes verzovací systém GitHub, který je veřejně přístupný na adrese: [https://github.com/Aqarel/MAPV\\_resistors/](https://github.com/Aqarel/MAPV_resistors/)

## 7 LITERATURA

- [1] Barevné modely: HSV. [online]. [cit. 2013-05-08]. Dostupné z: <http://web.vscht.cz/kalcicoa/POCPRE/hsv.html>
- [2] Barevné značení rezistorů. [online]. [cit. 2013-05-08]. Dostupné z: <http://www.ondrovo.com/index.php?k=elektronika&s=vypocty&c=rpZnaceni>