České vysoké učení technické v Praze
Fakulta elektrotechnická

BAKALÁŘSKÁ PRÁCE

Kamerové rozpoznávání konfigurace herních prvků v soutěži Eurobot

Praha, 2011 **Autor: Petr Kubizňák**

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 23.5.2011

podpis

Poděkování

Na tomto místě bych chtěl poděkovat Ing. Michalu Sojkovi, PhD. za úsilí, čas a významnou pomoc věnované mně i celému týmu Flamingos. Zároveň chci poděkovat všem členům tohoto týmu za výbornou spolupráci a výsledky v soutěži Eurobot. Poslední dík patří mé rodině a přítelkyni za jejich vytrvalou podporu mé osoby při studiích.

Abstrakt

Práce se zabývá návrhem vlastního klasifikátoru kamerových snímků do několika tříd podle rozložení herních elementů na hracím stole v soutěži Eurobot a jeho implementací v jazyce C/C++ s využitím open-source knihovny OpenCV. Metoda je postavena na principu porovnávání snímků s připravenými snímky referenčními. Několika experimenty je ověřena robustnost klasifikátoru, jehož kvalita je na konci potvrzena výsledky z finále soutěže. V práci je mimo jiné pro porovnání na stejném problému otestován také jednoduchý bayesovský klasifikátor.

Abstract

This work deals with design of a new classifier classifying camera frames to several classes according to configuration of playing elements on playing table in the Eurobot contest, and its implementation in C/C++ language using open-source library OpenCV. The method is built upon a principle of frames comparison with prepared referential images. Several experiments verify the classifier robustness, finally the quality is confirmed by results from the contest final. Simple Bayesian classifier is also tested on the same problem to compare with.

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Petr Kubizňák

Studijní program: Softwarové technologie a management

Obor: Inteligentní systémy

Název tématu: Kamerové rozpoznávání konfigurace herních prvků v soutěži Eurobot

Pokyny pro vypracování:

1. Seznamte se s pravidly soutěže Eurobot 2010 a knihovnou Open CV.
2. Navrhněte a implementujte algoritmus pro rozpoznávání polohy černých kukuřic na herní ploše.
3. Algoritmus důkladně otestujte a integrujte ho do řídicího systému robota.
4. Vše pečlivě zdokumentujte.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí bakalářské práce: Ing. Michal Sojka

Platnost zadání: do konce zimního semestru 2011/2012

L.S.

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

prof. Ing. Boris Šimák, CSc.
děkan

V Praze dne 6. 1. 2011

Obsah

1 Úvod **1**

2 Soutěž Eurobot **3**

2.1 Obecně . . . 3

2.2 Pravidla pro rok 2010 . . . 3

2.2.1 Konfigurace kukuřic . . . 4

3 Robot a strategie našeho týmu **7**

3.1 Architektura robota a použité technologie . . . 7

3.1.1 Hardware . . . 7

3.1.2 Software . . . 9

3.2 Herní strategie . . . 10

4 Rozpoznávání vylosované konfigurace prvků **11**

4.1 Specifikace problému . . . 11

4.2 Analýza . . . 11

4.3 Bayesovský klasifikátor . . . 14

4.3.1 Popis metody obecně . . . 14

4.3.2 Pořízení vstupních dat . . . 16

4.3.3 Předzpracování dat . . . 16

4.3.4 Měřené veličiny . . . 16

4.3.5 Zjednodušení modelu . . . 17

4.3.6 Trénování a testování . . . 19

4.3.7 Výsledky . . . 20

4.4 Použitý algoritmus . . . 22

4.4.1 Princip porovnávání . . . 23

4.4.2 Konstrukce modelu . . . 25

4.5 Implementace . . . 25

4.5.1 Použité funkce knihovny OpenCV . . . 26

4.5.2 Rozpoznávání v praxi . . . 27

4.5.3 Struktura programu . . . 28

4.5.4 Použití . . . 30

4.5.5 Pomocné aplikace . . . 33

4.5.6 Zdrojové soubory 34
4.6 Testování 35
4.6.1 Bezchybová datová sada 36
4.6.2 Experimenty s prahem 37
4.6.3 Experimenty s redundancí 38
4.6.4 Výsledky v soutěži 41

5 Závěr **43**

6 LITERATURA **45**

A Výpisy aplikace rozkuk **I**
A.1 Výpis 6. zápasu I
A.2 Experiment s redundancí IV

B Obsah přiloženého CD **VII**

Seznam obrázků

2.1 Ilustrace rozložení herních prvků na hracím stole. 4
2.2 Příklad boční konfigurace kukuřic (*S1*). 5
2.3 Příklad středové konfigurace kukuřic (*C2*). 6
2.4 Rozdělení hřiště do *oblastí výlučnosti*. 6
3.1 Robot při sběru „pomerančů" na šikmé rampě. 7
3.2 Schéma zapojení hardware robota. 8
3.3 Umístění kamery na robotu. 9
4.1 Běžný záběr z kamery na hřiště. 12
4.2 Záběr z kamery při nevhodných světelných podmínkách. 12
4.3 Ilustrace měření veličiny X_1. 17
4.4 Ilustrace měření veličiny X_1 na maskovaném snímku. 18
4.5 Odhad 2-D gaussovských distribucí. 20
4.6 Skutečné zařazení dat do tříd. 21
4.7 Schéma stavů programu rozkuk. 29
4.8 Screenshot aplikace rozkuk běžící na PC ve stavu `RECOGNIZE`. 31
4.9 Maska masek pro modré startoviště. 33
4.10 Boční a středová maska. 34
4.11 Schéma závislostí zdrojových souborů. 35
4.12 Kamerový snímek hřiště odpovídající snímkům ze soutěže. 40

Seznam tabulek

4.1 Průměrné trénovací (E_{trn}) a testovací (E_{tst}) chyby. 22
4.2 Výsledky klasifikátoru na bezchybové datové sadě. 36
4.3 Výsledky experimentu s prahem. 38
4.4 Výsledky experimentu s redundancí. 39
4.5 Výsledky v soutěži. 41

Seznam zkratek

BSD	Berkeley Software Distribution
GIMP	GNU Image Manipulation Program
GUI	Graphical User Interface
HW	Hardware
OCERA	Open Components for Embedded Realtime Applications
OpenCV	Open Source Computer Vision
ORTE	The OCERA Real Time Ethernet
PC	Personal Computer
PPC	PowerPC
RGB	Red-Green-Blue
SW	Software
USB	Universal Serial Bus

1 Úvod

Na Českém vysokém učení technickém v Praze existují v současnosti (květen 2011) dva amatérské týmy nadšených konstruktérů a programátorů, každoročně se pokoušející sestavit své roboty pro nový ročník mezinárodní robotické soutěže Eurobot. Ročníku 2010 (tj. v akademickém roce 2009/2010) jsem se v rámci týmu Flamingos pod vedením Ing. Michala Sojky, PhD. z Katedry řídicí techniky účastnil též. Tato práce popisuje postup řešení jednoho z mnoha problémů, které musel tým při konstrukci překonat, konkrétně problému kamerového rozpoznávání konfigurací některých herních prvků rozmístěných na hracím stole. Ve stručnosti se tedy zabývá návrhem, vývojem a testováním klasifikátoru kamerových snímků do tříd daných povolenými konfiguracemi.

Práce je rozdělena do několika kapitol podle okruhu zaměření na samotnou soutěž, vyvíjeného robota a vlastní problém.

O soutěži Eurobot a zmiňovaném ročníku pojednává kapitola 2. Ta má za úkol pouze přiblížit pozadí řešeného problému.

Stejnou úlohu hraje i kapitola 3, popisující ve stručnosti hardwarovou konstrukci a softwarové technologie používané v robotu, a dále herní strategii zvolenou týmem Flamingos pro daný ročník.

Můj přínos je zachycen v kapitole 4, zabývající se podrobně řešením vlastního problému. Ten je nejprve zanalyzován (část 4.2), přičemž je navrženo několik metod řešení. Jedna z nich – velmi jednoduchý bayesovský klasifikátor – je v části 4.3 naimplementována v jazyce Matlab® a otestována jen pro srovnávací účely. Druhá – použitý algoritmus – je potom popsána daleko důkladněji. Nejprve je v části 4.4 podrobně vyložen princip zvoleného algoritmu, v části 4.5 pak je popsána jeho implementace v jazyce C++, jeho struktura a použití a konečně v části 4.6 dochází k otestování vzniklé aplikace metodou několika experimentů a uvedení několika výsledků z finále soutěže na hokejovém stadionu v historickém švýcarském městě Rapperswil-Jona.

Práce obsahuje dvě přílohy: příloha A ukazuje příkladové konzolové výpisy produktu této práce (aplikace *rozkuk*), resp. jejich výtažky, a příloha B uvádí obsah CD přiloženého k této práci.

2 Soutěž Eurobot

Tato práce byla vyvíjena přímo za účelem použití v soutěži Eurobot. Proto bude nejprve vhodné o ní uvést několik základních informací.

2.1 Obecně

Eurobot je amatérská robotická soutěž, v níž soupeří týmy vesměs studentů s roboty vlastní výroby. Soutěž probíhá formou krátkých, 90vteřinových utkání dvou týmů. Každý tým postaví svého robota na startovní pole žluté nebo modré barvy, barvu týmu určuje rozhodčí vždy před začátkem zápasu. Po odstartování se tak po hřišti v jeden okamžik pohybují zároveň dva roboti plnící stejný úkol, hodnoceni jsou pak podle jejich úspěšnosti.

Na roboty jsou kladeny poměrně vysoké nároky. Základním požadavkem je jejich plná autonomita. Kontakt člověka s robotem je během hry zcela zakázán, robot musí veškerou svou činnost vykonávat sám, na základě vlastních algoritmů. Musí tak být schopen samostatně plnit úkol, vybírat vhodnou strategii a plánovat trasu tak, aby se vyhnul jakémukoliv kontaktu s protivníkem. Filozofií celé soutěže je totiž myšlenka, že stroje slouží k tomu, aby nám pomáhaly, nikoliv škodily, a tak si ani roboti navzájem nesmí bránit v pohybu ani se jinak omezovat.

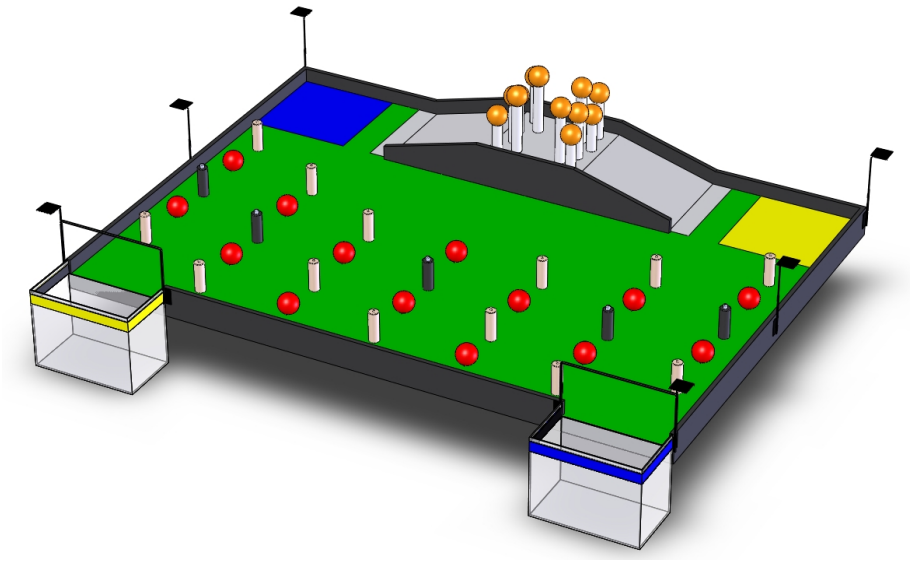
Smysl soutěže je v rozvoji technických dovedností účastníků z celého světa, s cílem podporovat i slabší regiony. Základním principem v boji proti elitismu je každoroční změna tématu soutěže. Z toho důvodu musí být vždy budován nový robot adaptovaný pro konkrétní úlohu, což umožňuje snazší zapojení nových týmů. V předchozích letech se tak už (v přeneseném významu) například hrál golf, sbíraly se odpadky nebo se stavěly chrámy. Kompletní přehled starších úloh viz [1].

2.2 Pravidla pro rok 2010

V názvu pro rok 2010, v němž byla použita tato práce, stálo „Nakrmte svět", v originále „Feed the World". Smyslem bylo nasbírat co nejvíc herních elementů, symbolizujících potraviny, a vysypat je do vlastního koše. Tři základní druhy potravin – ovoce, zelenina a obilniny – byly reprezentovány pomeranči, rajčaty a kukuřicemi, jejichž rozložení ilustruje obrázek 2.1. Celkem bylo před začátkem hry na stole vždy 44 herních prvků:

- 12 pomerančů po 300 bodech (oranžové míčky na „stromech" na rampě vzadu),

- 14 rajčat po 150 bodech (červené míčky definovaně rozmístěné po hrací ploše) a
- 11+7 kukuřičných klasů po 250 bodech, resp. 0 bodech (bílé a černé válečky taktéž definovaně rozmístěné po hrací ploše).



Obrázek 2.1: Ilustrace rozložení herních prvků na hracím stole.
(zdroj [2])

Je nutno vysvětlit důvod dvojího zbarvení kukuřic. V každé hře se na hrací ploše vyskytovaly klasy pravé (bílé) i falešné (černé), které byly ke stolu napevno připevněné. Celkem existovalo 36 možných konfigurací, ze kterých se ta aktuální losovala vždy těsně před začátkem hry, v době, kdy už hráči nesměli s roboty nijak interagovat. Rozložení černých kukuřic tak bylo proměnnou herního prostředí. Popis možných konfigurací následuje v části 2.2.1.

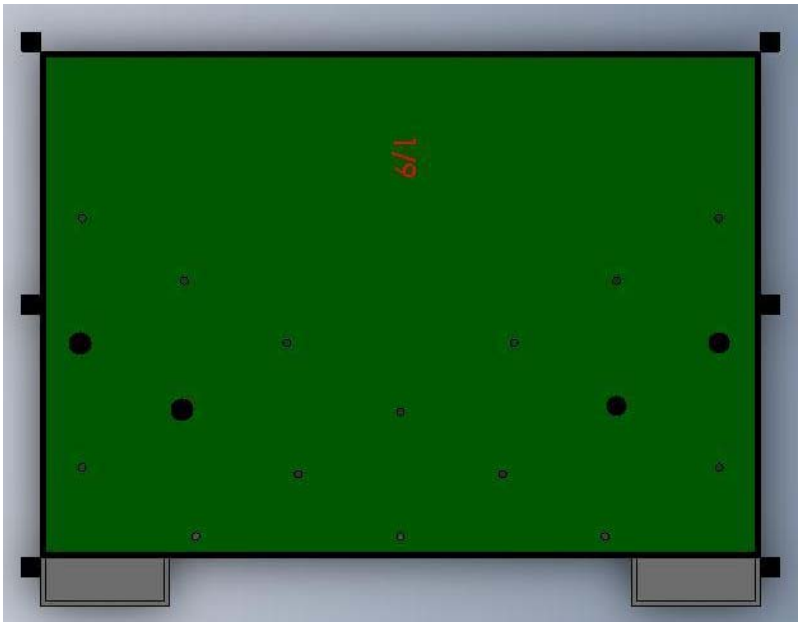
Kompletní oficiální pravidla pro rok 2010 jsou k dispozici v angličtině (viz [2]), pro jejich překlad viz [3].

2.2.1 Konfigurace kukuřic

Kukuřic bylo na hřišti rozmístěno vždy 18, z toho 11 bílých a 7 černých (ilustrační obrázek 2.1 je v jejich počtu chybný). Zasazeny byly do přesně definovaných otvorů ve stolu. Všechny možné konfigurace byly dány pravidly a sestávaly se z kombinace nastavení 2 oblastí – *boční* a *středové*.

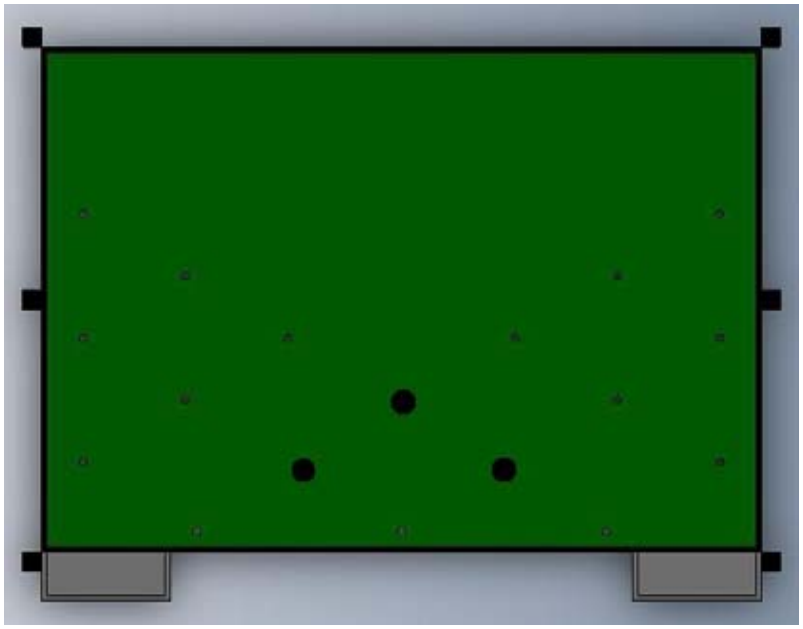
Boční konfigurace popisovala 6 kukuřic na boku hřiště, z nichž vždy 2 byly černé. Na druhé straně bylo rozložení symetrické. Bočních konfigurací bylo definováno 9 (v této práci je budeme označovat *S1* až *S9*). Příklad jedné z nich je na obrázku 2.2.

Středové konfigurace čítaly celkem 6 kukuřic, ale 2 byly opět symetrickým doplňkem, tedy popsat stačilo kukuřice 4 – 2 z nich byly bílé, 2 černé, konfigurací na nich definovaných bylo celkem 4 (označovány *C1* až *C4*, příklad na obr. 2.3).

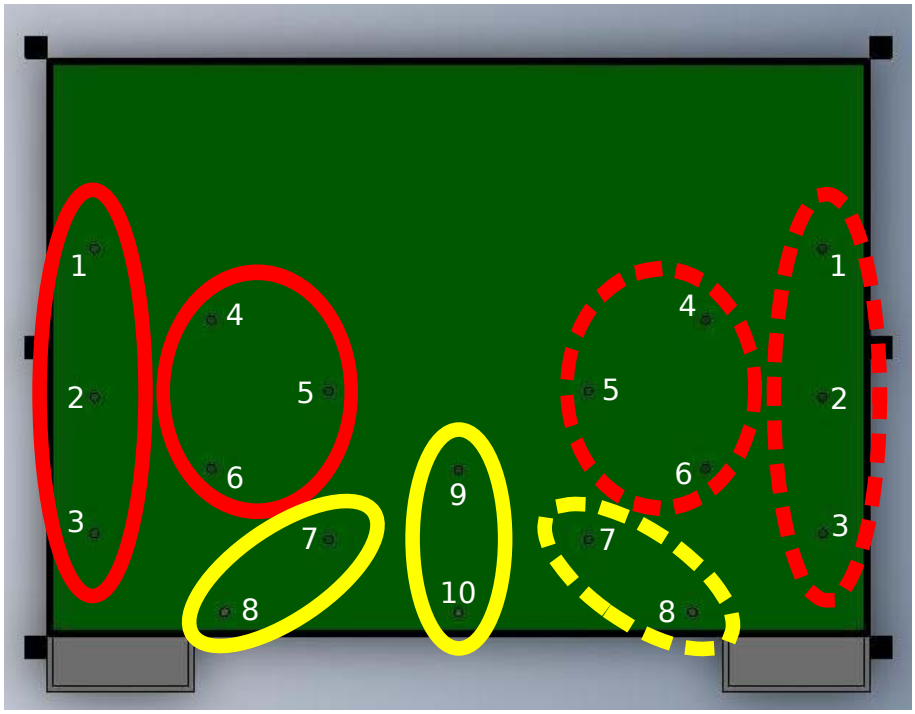


Obrázek 2.2: Příklad boční konfigurace kukuřic (*S1*).
(zdroj [2])

Výčet všech možností je zobrazen v pravidlech [2] na stranách 32–34. Bližším pohledem na ně ovšem zjistíme, že autoři pravidel hřiště rozdělili do 7 disjunktních oblastí, v nichž je vždy právě jedna černá kukuřice, ostatní jsou bílé (rozdělení na boční a středovou část je tedy umělé, vzniklé čistě z organizačních důvodů). Nazývejme tyto útvary **oblastmi výlučnosti**. Naznačuje je obrázek 2.4.



Obrázek 2.3: Příklad středové konfigurace kukuřic (*C2*).
(zdroj [2])



Obrázek 2.4: Rozdělení hřiště do *oblastí výlučnosti*, obsahujících vždy právě jednu černou kukuřici. V červených oblastech pravidla definují boční konfigurace, ve žlutých ty středové. Čárkovaně jsou naznačeny oblasti, jejichž stav je dán stavem symetrické oblasti na opačné straně hřiště. Kukuřice se stejnými čísly mají tedy vždy stejnou barvu.

3 Robot a strategie našeho týmu

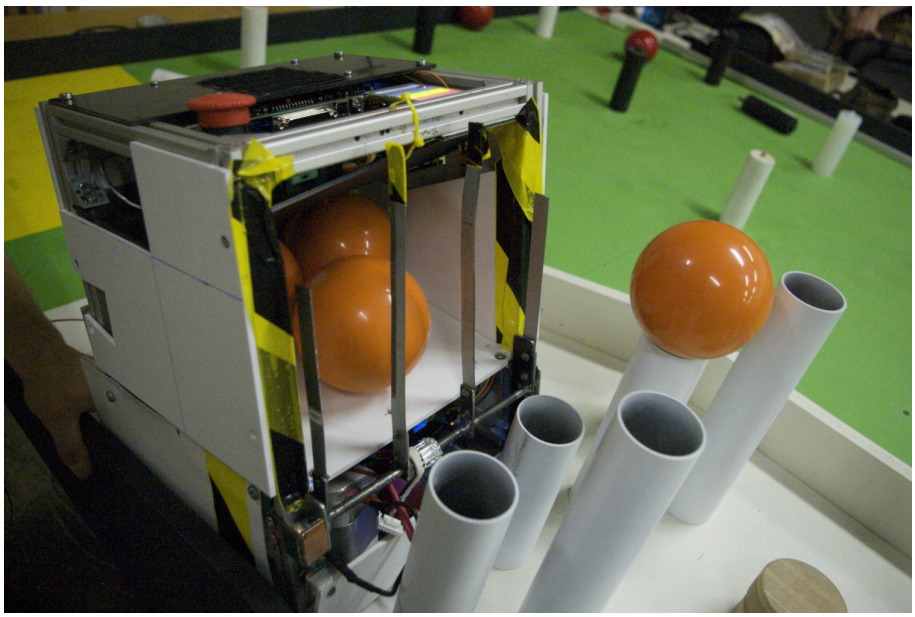
Tato část ve stručnosti popisuje hardware i software robota, použité technologie a herní strategii pro plnění požadovaného úkolu.

3.1 Architektura robota a použité technologie

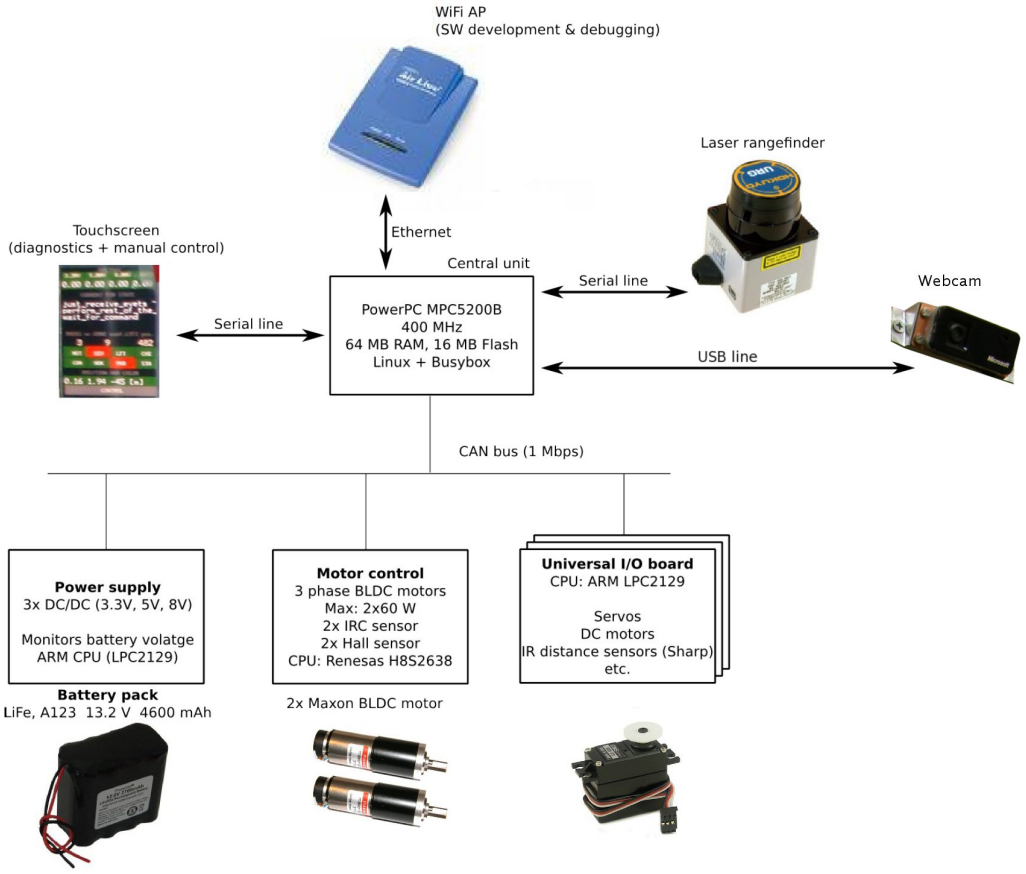
O architektuře a technologiích použitých v robotu týmu Flamingos bylo již napsáno mnoho v pracích [5, 6, 7] dřívějších členů týmu. Zde proto budou především zdůrazněny jen části týkající se řešeného problému.

3.1.1 Hardware

Robot (obr. 3.1) sestává z velkého množství vzájemně propojených modulárních součástí, z velké části vlastní výroby. Kostra robota je tvořena univerzálním podvozkem a rámem z profilu ITEM. Vnitřek obsahuje všechny komponenty nutné pro autonomní chod robota – napájecí baterii, pohony (motory, serva), senzory (laserový rangefinder, dotykové spínače, *kameru*), komunikační HW (displej, WiFi Access Point) a samozřejmě řídicí počítač a několik dalších desek starajících se o řízení jednotlivých periférií. Schéma zachycuje obrázek 3.2.



Obrázek 3.1: Robot při sběru „pomerančů" na šikmé rampě.
(zdroj [4])



Obrázek 3.2: Schéma zapojení hardware robota.
(zdroj [7], upraveno)

K řízení celého robota slouží řídicí počítač MIDAM s procesorem MPC5200b architektury PowerPC o taktovací frekvenci 400 MHz a s operační pamětí o velikosti 64 MB [7]. Na tomto boardu je spouštěn veškerý aplikační software včetně aplikace *rozkuk* (výsledný klasifikátor, bude popsán v části 4.5.3).

Blíže je třeba popsat použitou kameru. Jednalo se o běžnou USB webkameru značky Microsoft® LifeCam VX-500, poskytující snímky o dostatečném rozlišení 640 x 480 pixelů. Její diagonální úhel záběru je dle specifikace výrobce 56°. Kamera byla na robotu umístěna v přední horní části boční stěny, v níž byl pro ni vytvořen průřez, jak je vidět na obrázku 3.3. Průřezy byly po obou stranách, stejně jako plastové kolejničky, do nichž se kamera zasouvala v závislosti na aktuální hrací barvě.

Hlouběji se tato práce hardwarem použitém v robotu nezabývá – pro více informací viz již zmíněné práce [5, 6, 7].

Obrázek 3.3: Umístění kamery na robotu.

3.1.2 Software

Život robotu dává operační systém Linux běžící na řídicí desce MIDAM. V něm běží velké množství procesů spouštěných procesem řídicím. Jmenujme např. proces *competition*, stavový automat realizující rozhodovací centrum robota.

Nemá patrně smysl vyjmenovávat zde jednotlivé procesy, rozhodně ovšem nesmí být opomenut *ortemanager*, správce systému ORTE [8]. To je real-time komunikační mezivrstva, umožňující zajistit modularitu aplikací, které spolu potřebují komunikovat. Každá aplikace se do systému může zaregistrovat jako *subscriber*, chce-li informace přijímat, a jako *publisher*, chce-li je vysílat. Poté tedy se sdílenými daty nakládá jako s lokálními a o zbytek už se stará *ortemanager*. Systém ORTE je vyvinutým klasifikátorem používán pro přijem řídicích informací a informací o hrací barvě, a dále pro publikování výsledků klasifikace.

Co se týká knihoven, ve vyvinutém programu byla intenzivně používána otevřená knihovna OpenCV (v2.0) pro real-time počítačové vidění. Je uvolněna pod licencí BSD (viz přehled různých verzí v [9]) a je tedy zdarma pro akademické i komerční využití. Obsahuje stovky funkcí používaných v oblasti počítačového vidění, digitální fotografie, strojového učení a rozpoznávání. Sestává ze 4 částí:

cxcore Základní funkcionalita, definice potřebných datových struktur (př. matice) a práce s nimi, tj. funkce pro počítání s vektory a maticemi, funkce pro kreslení geometrických útvarů a podobně.

cv Pokročilé algoritmy pro počítačové vidění, počínaje filtry a transformacemi, konče různými detekcemi a rekonstrukcí 3D obrazu.

highgui Funkce pro práci s obrázky (tj. ne prostými maticemi), kamerou a jednoduchým grafickým rozhraním umožňujícím zobrazovat obraz a interagovat s uživatelem.

ml Balíček funkcí pro strojové učení.

V naší aplikaci byly použity jednak funkce pro čtení z kamery a základní práci se snímky (zobrazení, uložení, ...), tedy balíček *highgui*, jednak funkce pro realizaci základních výpočetních operací, popsaných v části 4.4.1, z balíčku *cxcore*. Druhý případ sice nebyl přímo nutný, ale určitě výhodný, neboť knihovna zaručuje vyšší bezpečnost i efektivitu výpočtů, než jakých by bylo zřejmě dosaženo vlastními silami. Oficiální referenční manuál lze nalézt v [10], užitečným shrnutím nejpoužívanějších struktur a funkcí je [11].

3.2 Herní strategie

Vzhledem k tomu, že náš tým měl za sebou již několik neúspěšných ročníků, bylo rozhodnuto vyvarovat se nespolehlivých, složitých mechanických konstrukcí, a vymyslet jednoduchý mechanismus, který umožní sbírat nejcennější[1] pomeranče.

Veškerý důraz tak byl kladen na funkčnost jednoduchých „vidlí" na zádi robota a přesnost pohybu na šikmé rampě. Dále bylo potřeba řešit uložení HW (především kol, motorů a baterie) tak, aby vznikl co největší úložný prostor. Cílem bylo pojmout všech 6 pomerančů z jedné strany najednou.

Toto bylo realizovatelné, nicméně na úkor jakýchkoliv dalších mechanismů pro sběr rajčat a kukuřic, neboť na ně již nezbyl prostor. Bylo proto rozhodnuto, že naší strategií bude sebrat všech 6 pomerančů na startovní straně, případně dalších 6 na straně soupeřově, a ve zbytku času se bude robot jen snažit nějaká rajčata a kukuřice do koše dotlačit. A protože ke sbírání byly určeny pouze kukuřice bílé, bylo nutné nejprve sestrojit algoritmus, který by jejich rozmístění rozpoznával. O tom pojednává následující kapitola.

[1] Viz bodové ohodnocení herních prvků v části 2.2.

4 Rozpoznávání vylosované konfigurace prvků

Úloha rozpoznávání vylosované konfigurace byla potřebná k tomu, aby robot znal rozložení černých kukuřic, kterým bylo nutné se vyhýbat, protože byly ke hřišti přišroubované a tvořily tak překážku. Jakákoliv srážka robota s jiným předmětem vedla totiž nejenom ke ztrátě cenného času, ale také s sebou nesla riziko ztráty orientace vinou odskoku robota při nárazu.

4.1 Specifikace problému

Problém rozpoznávání vylosované konfigurace je klasifikační úloha, jejímž vstupem je obraz (několik obrazů) hřiště, sejmutý kamerou na robotu, a jejímž výstupem jsou třídy, tj. čísla bočních (1–9, značeno *S1–S9*) a středových (1–4, značeno *C1–C4*) konfigurací tak, jak jsou definována v pravidlech [2] a popsána v části 2.2.1. Obraz je pořizován ze známého bodu v klidovém stavu (v prodlevě před odstartováním), což bylo rozhodnuto proto, abychom znali (relativně) přesné umístění předmětů v obraze, neboť pozici robota před startem lze přesně definovat. Umístění objektů je tedy známé, neznáme pouze jejich barvy.

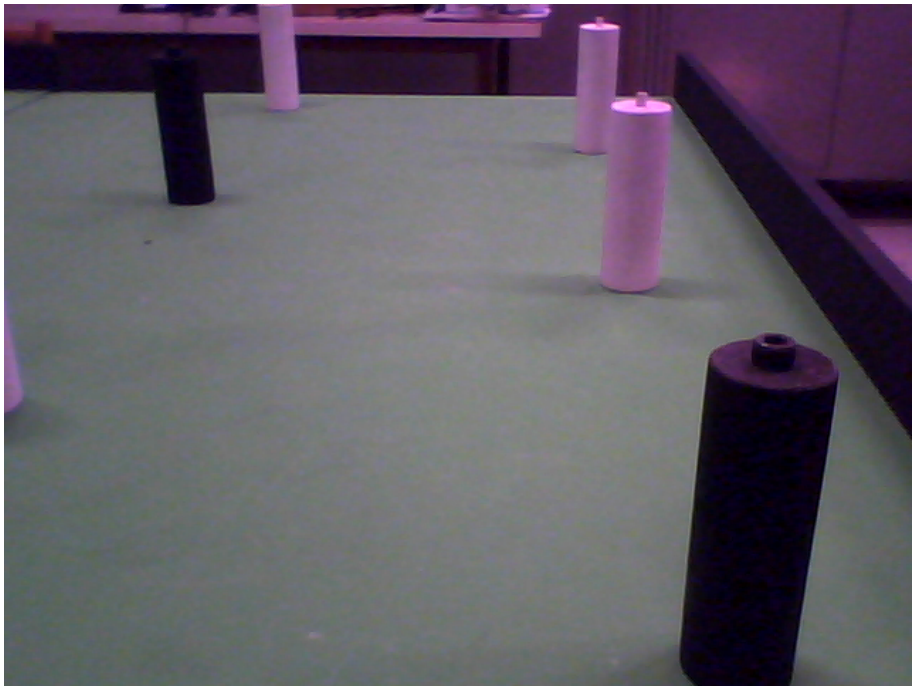
Vzhledem ke znalosti herního prostředí můžeme úlohu omezit na stanovené podmínky. Těmi je především neměnné prostředí (jak bylo vysvětleno v předchozím odstavci), běžné halové osvětlení (tj. větší množství silných, stacionárních zdrojů světla) a zbarvení hřiště a herních prvků dle pravidel. Za standardní vstup algoritmu tak lze považovat např. obr. 4.1, zatímco nevhodný obrázek 4.2 není třeba uvažovat.

4.2 Analýza

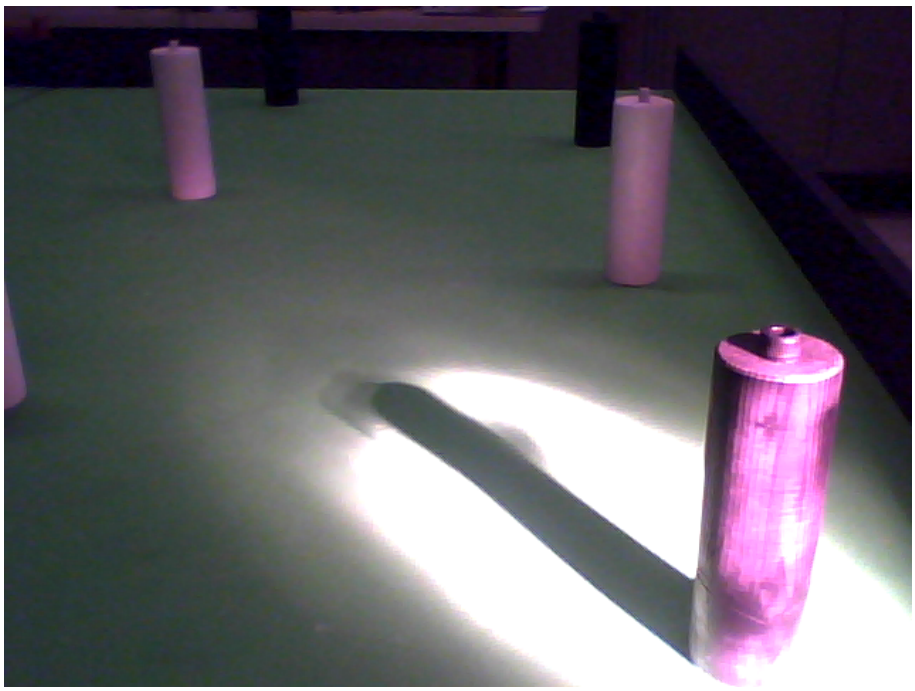
Specifikovaný problém by jistě bylo možné řešit např. použitím speciálních senzorů, my jsme se ale rozhodli využít levné a relativně jednoduché řešení – kamerové vidění. Jak již tedy bylo řečeno, vstupem rozpoznávacího algoritmu byl *obraz* sejmutý běžnou USB kamerou umístěnou na robotu. Výstupem takové kamery byly snímky podobné[2] obrázku 4.1.

Na snímku je možné (a nutné) si všimnout množství nežádoucích jevů – obraz z levné kamery je silně zašuměn, kamera zároveň automaticky provádí jasové a barevné úpravy, jejichž důsledkem je především výrazné zkreslení bílých kukuřic, které

[2] V době analýzy ještě nebylo známo přesné umístění kamery na robotu. V praxi byly proto záběry trochu odlišné (viz obrázek 4.12).



Obrázek 4.1: Běžný záběr z kamery na hřiště.



Obrázek 4.2: Záběr z kamery pořízený při nevhodných světelných podmínkách (ostrý, bodový zdroj světla).

se na snímku jeví spíše jako růžové. Stejně tak i hřiště se jeví spíše jako šedé, ačkoliv v reálu je sytě zelené. Také snímky dvou různých konfigurací, ač za stejných svě-

telných podmínek, se mohou výrazně lišit v celkové světlosti z důvodu automatické akomodace kamery. Nebylo tedy možné spoléhat na nějaký konstantní práh, který by byl hranicí mezi hledanou černou a bílou.

Podstatnou zjednodušující vlastností problému byla stacionarita robota i snímaných předmětů v době pořizování snímků. Díky tomu nebylo třeba řešit žádné detekce a bylo možné přistoupit rovnou k rozpoznávání. Na druhou stranu však bylo třeba si uvědomit, že při každé hře bude robot polohován i rotován s určitou nepřesností, v důsledku čehož se budou polohy objektů na různých snímcích přece jen lišit.

Významnou roli pro funkci algoritmu by mohl hrát objem redundantních dat. Jak je vysvětleno v části 2.2.1, v každé z *oblastí výlučnosti* se nachází právě jedna černá kukuřice, ostatní jsou bílé. Je tedy zřejmé, že nám stačí znát umístění černých kukuřic, bílé kukuřice tak do snímků vnáší redundanci. Za zvážení pak stálo, zdali redundanci odstranit (pro zrychlení algoritmu), nebo raději využít pro zvýšení spolehlivosti (a jak to udělat).

Autoři článku [12] pracovali na problému podobném tomu našemu, dokonce z obdobné oblasti – vyvíjeli klasifikátor pro určování barev robotů v robotické fotbalové soutěži RoboCup. Snažili se přitom vyrovnat s vlivy proměnlivého osvětlení. V prvním kroku prováděli segmentaci obrazu (ta je základem většiny podobných aplikací, viz např. [14]), v segmentech počítali průměrnou barvu, kterou pak přiřadili barvě s nejmenší Mahalanobisovou vzdáleností (vysvětlena např. v [13]).

Problémem barev v obraze se zabývala práce [15], jejíž autoři se pokoušeli navrhnout algoritmus pro rozpoznávání objektů v obraze na základě barevných vlastností snímku. Snahou bylo eliminovat především vlivy osvětlení, geometrie objektu a úhlu pohledu. Testovali odolnost barevných modelů vůči těmto vlivům a zjistili, že barevná složka (*Hue*) obrazu splňuje vysoké množství kritérií. Rozpoznávání pak probíhalo na základě porovnávání dat z testovacích obrazů s daty trénovacími.

Jeden z autorů předchozí práce řešil tentýž problém také pomocí histogramů v článku [16]. Je ovšem zřejmé, že tento postup by nemohl v naší úloze příliš uspět, neboť intuitivně mají všechny snímky podobné histogramy.

Segmentaci v této práci nebylo třeba řešit, jak již bylo zmíněno, pozice objektů ve snímku byly známé. S přihlédnutím ke všem zmíněným aspektům bylo zřejmé, že bude stačit jednoduchý algoritmus s požadavkem na vysokou odolnost vůči šumu. K tomu by bylo možné použít jeden z mnoha standardních, obecných klasifikátorů z teorie rozpoznávání, nebo navrhnout nějaký „nový", šitý speciálně na míru danému problému. Diskutujme 4 návrhy:

Bayesovský klasifikátor Klasifikátor ze třídy standardních algoritmů. Na snímcích naměříme určité veličiny a ze znalosti trénovacích dat odhadneme třídu aktuálního snímku. Toto znatelně sníží objem dat (komprese tisíců hodnot pixelů do několika čísel) a jejich redundanci. Takový algoritmus byl částečně ozkoušen, podrobnosti viz část 4.3.

Jasy pixelů Naivní metoda, v níž pro klasifikaci použijeme jen hodnoty několika předem stanovených pixelů ve snímku, např. z každé kukuřice jeden. To by zjevně kvůli množství šumu v datech nemohlo fungovat.

Průměrné barvy Algoritmus „na míru" danému problému, inspirovaný prací [12]. Zahrnuje výpočet průměrné barvy každé kukuřice a následné porovnání těchto hodnot v rámci *oblastí výlučnosti*, jak byly popsány v závěru části 2.2.1. Redundantní data (bílé kukuřice) jsou nutná pro samotnou funkci – vzájemné porovnání. Takový algoritmus by měl fungovat, vyžadoval by však o něco individuálnější přístup k jednotlivým částem snímku, než je nezbytné (zvláštní přístup ke každé z kukuřic a jejich zařazení do oblastí výlučnosti), a dostatečně široký záběr kamery, aby byly vidět všechny potřebné kukuřice. Z těchto důvodů tento algoritmus nebyl použit.

Násobení masek Jiný algoritmus „na míru". Zkoumaný snímek porovnáme postupně s předem připravenými maskami a výsledek bude dán tou, jež snímku odpovídá nejlépe. Redundantní data zde zvyšují spolehlivost klasifikátoru. Tento algoritmus byl použit a je detailně popsán v části 4.4.

4.3 Bayesovský klasifikátor

Dříve než přistoupíme k vysvětlení použitého algoritmu (násobení masek, část 4.4), bude popsán jeden ze standardních rozpoznávacích algoritmů, který byl na daném problému testován – běžný bayesovský klasifikátor, rozhodující na základě pravděpodobnostního rozdělení určitých veličin. Implementován byl až po soutěži pouze pro srovnávací účely.

4.3.1 Popis metody obecně

Pro čtenáře méně zběhlé v oblasti rozpoznávacích algoritmů bude nejprve vysvětlen obecný princip bayesovského klasifikátoru a názvosloví.

Bayesovský klasifikátor je standardní rozhodovací algoritmus, který při známých apriorních pravděpodobnostech $p(k)$ všech tříd $k \in K$ a aposteriorních pravděpodobnostních rozděleních $p(x|k)$, $x \in X$ minimalizuje střední ztrátu

$$R(q) = \sum_{x \in X} \sum_{k \in K} p(x|k) \cdot p(k) \cdot W(k,q(x)) \tag{4.1}$$

rozhodovací strategie $q : X \to D$, kde $W : K \times D \to \mathbb{R}$ je ztrátová matice rozhodnutí, D je množina všech možných rozhodnutí, K množina všech tříd (skrytých stavů), X množina všech pozorování.

Tento typ klasifikátoru se používá k odhadu tříd (skrytých stavů, tj. atributů, které lze určit jen těžko, draze, nebo dokonce vůbec) objektů, na nichž lze měřit nějaké veličiny (pozorování; atributy, které jsou snadno měřitelné). Nejprve uskutečněním určitého množství měření zkoumaných objektů vytvoříme datovou sadu, v níž jsou pozorováním přiřazeny jejich skutečné třídy (jedná se tedy o učení s učitelem). Sadu rozdělíme v požadovaném poměru na *trénovací*, na níž klasifikátor naučíme, a *testovací*, která nám umožní provést odhad spolehlivosti modelu.

Pro určení střední ztráty R rozhodnutí q vyžaduje bayesovský klasifikátor znalost *apriorních* a *aposteriorních* pravděpodobností a ztrátové matice. Apriorní pravděpodobnost $p(k)$ určuje, s jakou pravděpodobností patří právě zkoumaný objekt do třídy k, aniž bychom o něm cokoliv věděli (to může být ovlivněno cílovou skupinou objektů). Aposteriorní pravděpodobnost $p(x|k)$ poté odráží „šanci", že na objektu budeme pozorovat hodnotu x, víme-li, že spadá do třídy k. A konečně ztrátová matice (nebo lépe funkce) $W(k,d)$ určuje, jak drahá je klasifikace objektu třídy k do třídy d (musí tedy být definována na všech možných takových kombinacích). To se používá především pro stanovení pokuty za nesprávné rozhodnutí.

Trénování klasifikátoru znamená vytvoření takového rozhodovacího modelu, který minimalizuje hodnotu výrazu (4.1). To probíhá na zmíněné trénovací sadě. Třídy testovacích dat jsou poté nejprve odhadnuty vytvořeným modelem, a poté porovnány se skutečnými třídami, což umožňuje změřit nevychýlený odhad chybovosti klasifikátoru. Ten by tak měl přibližně odpovídat skutečné chybovosti klasifikátoru při jeho nasazení v praxi.

Tímto je obecný princip bayesovského klasifikátoru zběžně vysvětlen, jeho praktické použití ukazují následující sekce.

4.3.2 Pořízení vstupních dat

Konstrukce a použití bayesovského klasifikátoru sestává z několika kroků. Jako první bylo nutné pořídit kamerové snímky, z nichž by bylo možné vygenerovat datovou sadu pro trénování a testování modelu. K tomu byl využit `VIDEO` mód programu *rozkuk* (bude popsán v sekci 4.5) a kamera, připojená k osobnímu PC a upevněná na robotu, aby byla správně umístěna. Pořízeno bylo 5 snímků trénovacího hřiště pro každou konfiguraci a obě herní barvy tak, aby se měnila výchylka od ideální polohy robota, ale přitom byly všechny snímky aplikací *rozkuk* klasifikovány správně. Po jejich pořízení musely být dále zpracovány.

4.3.3 Předzpracování dat

Protože potřebujeme zvlášť rozpoznávat boční a středovou konfiguraci, bylo nutné ve vstupních snímcích zamaskovat nepotřebné kukuřice z druhé oblasti, aby nezanášely do měření šum, a také snímky převést do stupňů šedi, aby bylo možné intenzitu každého pixelu popsat jedním číslem, a tak snížit celkové nároky algoritmu.

Nejprve proto byly v grafickém editoru GIMP vytvořeny zmíněné masky – černé obrázky stejných rozměrů jako snímky, s bílými oblastmi v místech, kde jsou na snímcích kukuřice. Z důvodu určitých odchylek není možné přesně určit, kde na snímcích hranice kukuřic jsou, a proto byly hranice bílých oblastí na maskách rozostřeny, čímž byla vyjádřena určitá neznalost.

Následně byl napsán skript *preparedata.m* pro výpočetní program Matlab®, který načítá masky a převádí je do matice reálných čísel z intervalu $\langle 0; 1 \rangle$, kde do 0 se zobrazují černé pixely a do 1 bílé. Pronásobením s pořízenými snímky, převedenými do stupňů šedi a následně do stejného intervalu reálných čísel, získáme matice zachycující pouze data relevantních kukuřic v podobě nenulových prvků. Matice lze uložit pro využití v algoritmu do souboru zdrojových dat pro matlab (soubor s příponou *mat*).

4.3.4 Měřené veličiny

Následně bylo nutné určit veličiny, které budeme na snímcích měřit. Kukuřice jsou rozptýlené po celém snímku, přičemž zkoumáme v zásadě intenzity odpovídající různým barvám kukuřic v jeho různých částech. Z toho důvodu se jako použitelná veličina jeví být *rozdíl součtů intenzit v obou polovinách snímku*. Znamená to, že jako jednu veličinu X_1 sečteme zvlášť intenzity v levé polovině (S_L) a zvlášť v pravé polovině snímku (S_R) a tyto poté odečteme, tedy $X_1 = S_L - S_R$. Obdobně pak

získáme druhou veličinu $X_2 = S_T - S_B$ pro vertikální rozdělení, kde S_T značí součet intenzit v horní polovině a S_B v dolní polovině snímku. Výpočet první veličiny ilustruje obrázek 4.3, resp. maskovaný snímek 4.4.

Obrázek 4.3: Ilustrace měření veličiny X_1 – od součtu intenzit všech pixelů v levé části snímku (červeně) odečteme součet intenzit všech pixelů v pravé části snímku (zeleně). Výsledkem je celé číslo. V reálu ovšem byly použity maskované snímky, lépe tedy skutečnost ilustruje obrázek 4.4.

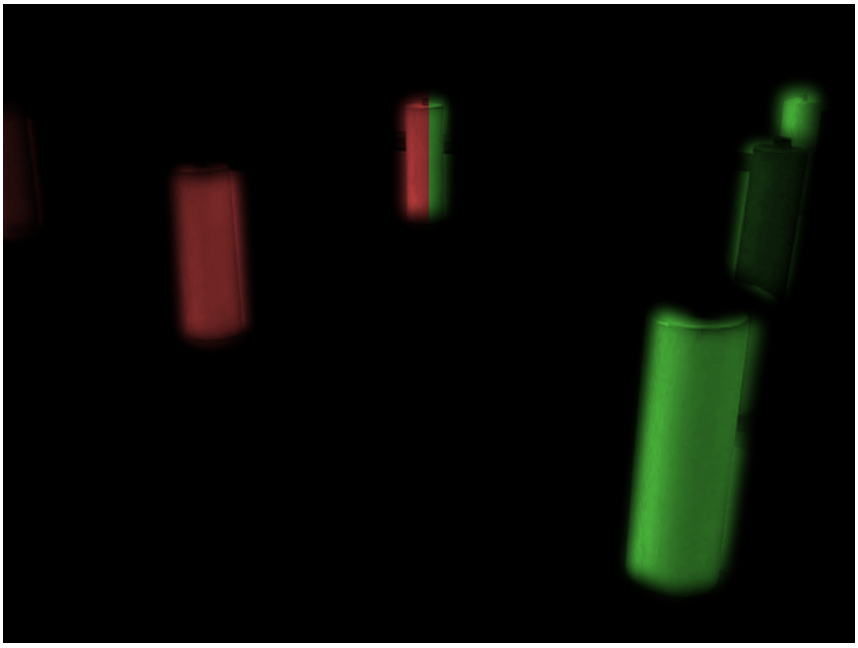
Měřit lze de facto libovolné veličiny, které je možné ze snímků vypočíst. Ty popsané byly vybrány proto, že občas v praxi dobře fungují (příkladem může být úloha ve zdroji [18]).

Naměřením zmíněných veličin byly vytvořeny 4 malé datové sady (zvlášť pro boční/středové konfigurace a obě herní barvy), obsahující pro každou třídu pět pozorování (vektorů $\boldsymbol{x} = (X_1; X_2)$) a jejich „popisky", tj. označení tříd.

4.3.5 Zjednodušení modelu

Naměřením vstupních dat jsme získali rozdělení veličiny $x \in X$ pro každou třídu $k \in K$. Pro výpočet středního rizika podle (4.1) je ovšem nutné ještě zjistit apriorní a aposteriorní pravděpodobnosti a ztrátovou matici. To může vést ke značnému zjednodušení výrazu, jak ukazují následující odstavce.

Za prvé, v naší aplikaci nás nezajímalo, jaká chyba nastala, ale pouze, jestli nastala. Proto byla použita ztrátová funkce L_{01}, která za chybné rozhodnutí vnáší



Obrázek 4.4: Ilustrace měření veličiny X_1 na maskovaném snímku (pro boční konfiguraci). Je vidět, že většina pixelů je černá a nemá tak v měření žádný přínos, do výsledku se promítají pouze pixely v odstínech červené a zelené.

pokutu 1 a za správné 0 (ztrátová matice W má potom na hlavní diagonále nuly, všude jinde jedničky). Tím se problém redukuje na minimalizaci pravděpodobnosti nesprávného rozhodnutí, jak je ukázáno v [17, str. 21], z čehož plyne, že stačí v každém bodě x vybírat třídu k s nejvyšší podmíněnou pravděpodobností $p(k|x)$, tedy

$$q(x) = \underset{k \in K}{\operatorname{argmax}}\, p(k|x), \tag{4.2}$$

jak je dokázáno v [17, str. 22].

Další zjednodušení plyne ze skutečnosti, že všechny třídy mají stejné apriorní pravděpodobnosti $p(k)$, neboť konfigurace jsou vybírány náhodně vždy z celého počtu tříd. Z rovnosti (4.2) tak lze vztahem (4.3) odvodit nový výraz (4.4), protože $p(k)$ je konstanta v celé úloze a $p(x)$ v daném bodě x také, a tak nemají na argument maxima vliv.

$$q(x) = \underset{k \in K}{\operatorname{argmax}}\, p(k|x) = \underset{k \in K}{\operatorname{argmax}}\, \frac{p(x|k) \cdot p(k)}{p(x)} = \underset{k \in K}{\operatorname{argmax}}\, \frac{p(k)}{p(x)} \cdot p(x|k) \tag{4.3}$$

$$q(x) = \underset{k \in K}{\operatorname{argmax}}\, p(x|k) \tag{4.4}$$

Tímto docházíme k závěru, že třídu $d = q(x)$ je možné volit pouze na základě maximální aposteriorní pravděpodobnosti $p(x|d)$ v daném bodě x.

4.3.6 Trénování a testování

Na základě předchozích úvah je zřejmé, že trénování celého modelu spočívá v nalezení rozhodovacích hranic mezi oblastmi, v nichž jsou měření klasifikována vždy do třídy k s nejvyšší aposteriorní pravděpodobností $p(x|k)$. Za předpokladu této znalosti na tom není nic složitého.

Tyto pravděpodobnosti ale dopředu známé nebyly, a bylo proto nutné provést jejich odhad na základě naměřených dat. Distribuce dat jsme považovali za dvourozměrná normální rozdělení, jejichž parametry je možné odhadnout využitím metody maximální věrohodnosti (popsána v [19]).

Celé trénování tedy spočívalo v odhadu středních hodnot μ_i a kovariančních matic Σ_i 2-D gaussovských distribucí splňujících kritérium maximální věrohodnosti

$$[\mu_i,\Sigma_i] = \underset{\mu_i,\Sigma_i}{argmax} \prod_{x \in X_i} p(x|\mu_i,\Sigma_i) \tag{4.5}$$

zvlášť pro každou třídu i. To bylo prováděno na trénovací množině, vzniklé náhodným rozdělením naměřených dat na trénovací a testovací sadu v poměru 4:1 pro každou třídu (pro zachování shodné mohutnosti všech tříd). Výsledkem byla gaussovská směs, jejíž příklad zobrazuje obrázek 4.5. Je na něm vidět devět elips značících odhady rozložení tříd – šest v levé dolní a tři v pravé horní části grafu.

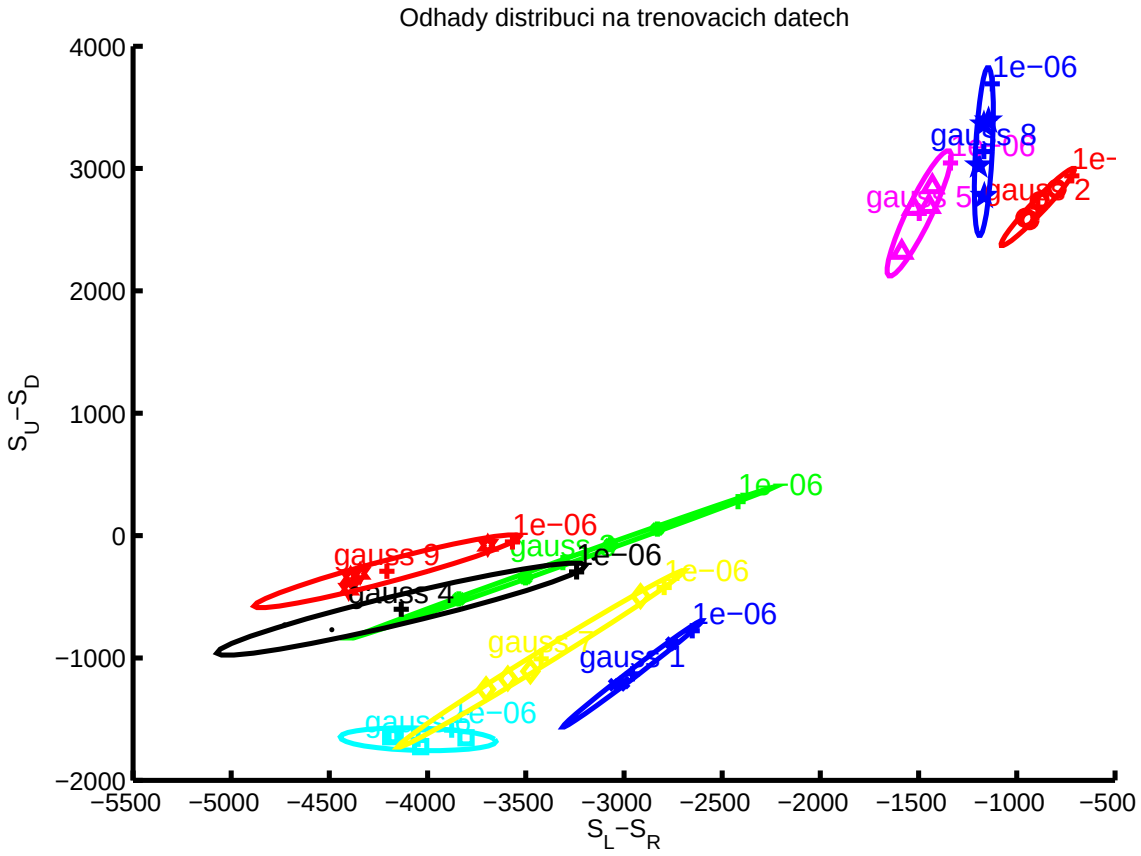
Tři oddělené třídy v pravém horním rohu odpovídají těm konfiguracím, kdy je kameře nejbližší kukuřice černá. Měření z těchto třech konfigurací se liší především v x-ové souřadnici, což souhlasí s tím, jak se na snímcích „pohybuje" druhá černá kukuřice, tedy především ve vodorovném směru. Tyto konfigurace jsou tak od sebe celkem dobře separovatelné.

Podíváme-li se blíže na skupinu šest elips v levé dolní části grafu, zjistíme, že je tvořena dvěma shluky po třech třídách. Shluky opět odpovídají umístění černé kukuřice v postranní části snímku – je-li černá kukuřice z pohledu robota uprostřed, nebo vzadu. U těchto měření ale zaznamenáváme větší rozptyly dat a překryvy řezů distribucí[3], což zákonitě vede na větší chybovost modelu.

Klasifikace dat z obou množin probíhala na základě vztahu (4.4) popsaného v části 4.3.5, jehož důsledkem jsou rozhodovací hranice, které jsou vidět v horní

[3]Distribuce se samozřejmě prolínají všechny, v uvedené oblasti ovšem i pro vysoké hodnoty pravděpodobností, které reprezentují.

části obrázku 4.6. Z nich je dobře patrný důvod výsledku klasifikace v dolní části téhož obrázku, kde můžeme pozorovat i chybně klasifikovaná trénovací a testovací data.

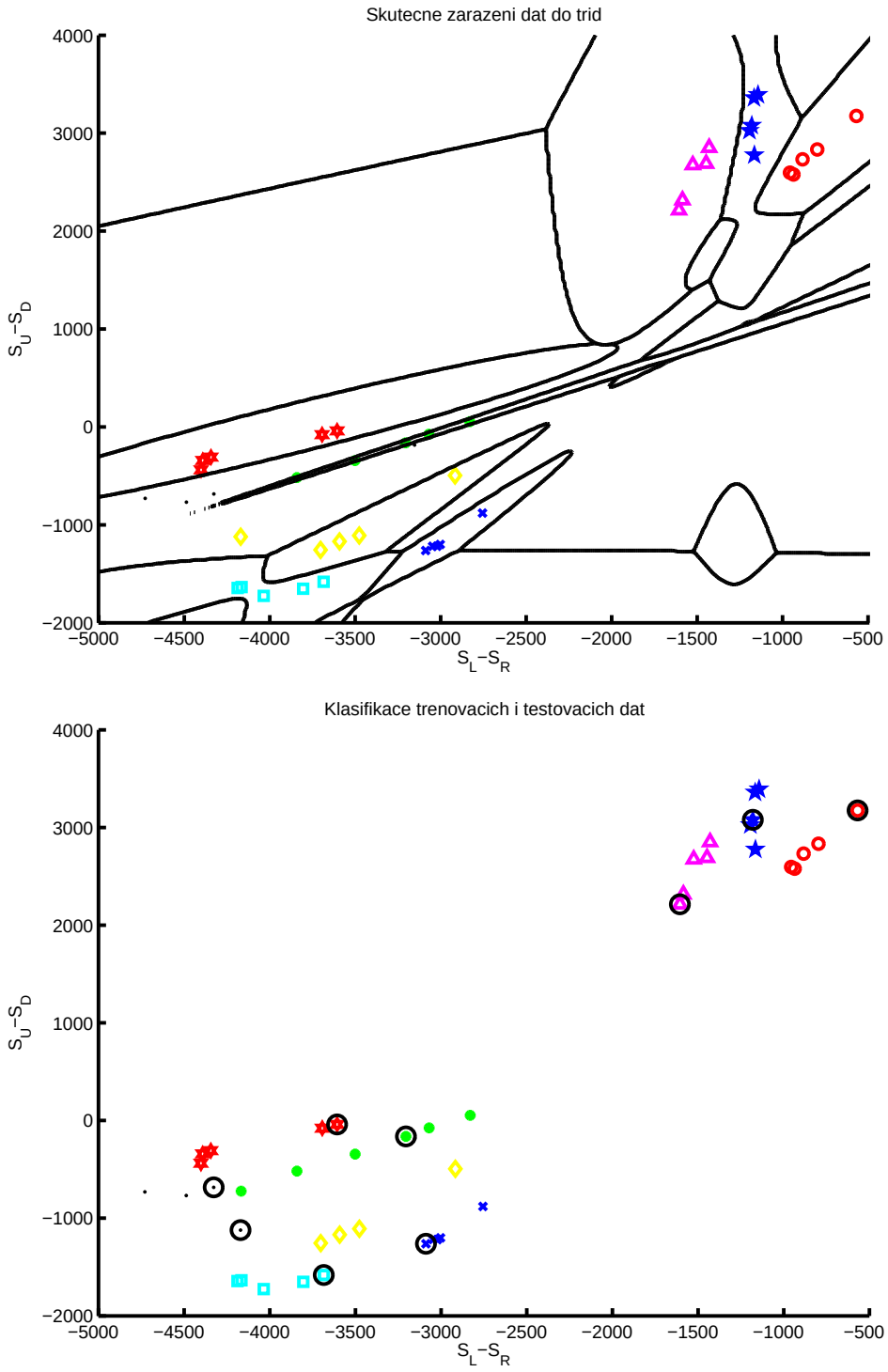


Obrázek 4.5: Odhad 2-D gaussovských distribucí na základě náhodně vybraných trénovacích dat. Elipsy znázorňují řezy distribucemi pro hodnoty pravděpodobností $p(x|k) = 10^{-6}$. Čísla v názvech elips (tj. „gauss X") odpovídají číslům bočních konfigurací.

4.3.7 Výsledky

Abychom získali průměrnou chybovost modelu, bylo jeho trénování a testování v cyklu opakováno 10000krát, přičemž rozdělení sady na trénovací a testovací probíhalo náhodně. To bylo provedeno postupně pro všechny čtyři datové sady (žlutá/modrá herní barva, boční/středové konfigurace). Výsledkem byly průměrné trénovací a testovací chyby, jak ukazuje tabulka 4.1.

Výsledky překvapí snad jen v jediné věci – že pro dvě datové sady byla dosažena zcela nulová trénovací chyba. Ne že by to bylo něco nemožného, ale při pohledu



Obrázek 4.6: Skutečné zařazení dat do tříd s rozhodovacími hranicemi klasifikátoru (nahoře) a klasifikace dat na základě těchto hranic (dole). Kroužkem jsou zvýrazněny testovací body.

E_{trn}/E_{tst} [%]	žlutá	modrá
boční	0,00 / 6,30	0,68 / 17,90
středové	5,47 / 19,22	0,00 / 14,78

Tabulka 4.1: Průměrné trénovací (E_{trn}) a testovací (E_{tst}) chyby bayesovského klasifikátoru na 4 různých datových sadách.

na blízkost dat např. v sadě „žlutá-boční" to opravdu překvapí. Zbytek už ale ničím překvapivý není. Nízká chyba na trénovacích datech společně s vysokou chybou na datech testovacích je jasným úkazem přetrénování klasifikátoru, což je ale pochopitelné, uvědomíme-li si, jak malá datová sada byla použita.

Velikost datové sady by byla skutečně závažným problémem, který by znamenal naprosté selhání v řešení problému. A samozřejmě, vlastní numerické výsledky z tabulky 4.1 jsou zcela nepoužitelné. Zde to ale nevadí, neboť tato metoda má v této práci pouze podpůrnou hodnotu – významné nejsou přesné výsledky, šlo pouze o to zjistit, jestli by takovýto klasifikátor byl pro řešení úlohy použitelný. Vzhledem k tomu, jak je i takovéto malé množství dat částečně promíseno, je zřejmé, že v této podobě by tento model úplně dobře nefungoval. Na druhou stranu je vidět, že přímo špatným směrem se zvolený postup také neubíral. Stačilo by zřejmě zvolit vhodnější měřené veličiny, resp. nějaké přidat (zvýšit dimenzi prostoru příznaků) a model by již mohl fungovat s dostatečně vysokou spolehlivostí. To už ovšem leží mimo rámec této práce.

Závěrem této části budiž touto kapitolou částečně podložená hypotéza, že bayesovský klasifikátor by bylo možné pro řešení zkoumané úlohy použít, nicméně jedná se o zbytečně složité řešení obnášející pořízení dostatečně velkého množství trénovacích dat, jejich netriviální zpracování, odhad parametrů vícedimenzionálních normálních rozdělení a rozhodování na základě výpočtů relativně složité funkce.

4.4 Použitý algoritmus

Z důvodů uvedených v závěru předchozí podkapitoly jsme použili algoritmus jednodušší, v části 4.2 nazvaný *algoritmus násobení masek*. Jeho základní myšlenkou je představa, že kamerový snímek bude porovnán s trénovacími snímky (dále *maskami*), z nichž bude vybrán ten, který snímku z kamery odpovídá nejlépe. Tato podkapitola vysvětluje princip funkce zvoleného algoritmu.

4.4.1 Princip porovnávání

Základním problémem algoritmu je otázka, jak vhodně porovnávat kamerové snímky s maskami. Jako vhodná se zdála být myšlenka přeškálovat hodnoty pixelů masek ze stupňů šedi[4] do reálných čísel (obecně jakoukoliv) lineární transformační funkcí $T : \{0;\ldots;255\} \to \langle -1{,}0; 1{,}0\rangle$. Totéž provádět pro kamerové snímky, ty pak po „pixelech" násobit s maskami, všechny součiny sečíst a hledat masku, pro níž je takový součet největší.

Formálně: Mějme N (transformovaných) masek o rozměrech $h \times w$ odpovídajících N různým třídám (konfiguracím) z množiny $K = \{1;\ldots;N\}$, kde k-tou masku $\mathbb{M}_k$ lze zapsat jako vektor $\boldsymbol{m_k} \in \langle -1{,}0; 1{,}0\rangle^{h\cdot w}$. Mějme dále (transformovaný) snímek $\mathbb{X}$ o rozměrech $h \times w$, zapsaný jako vektor $\boldsymbol{x} \in \langle -1{,}0; 1{,}0\rangle^{h\cdot w}$. Vektory $\boldsymbol{m_k}$ a $\boldsymbol{x}$ jsou řádkovými zápisy matic $\mathbb{M}^k$ a $\mathbb{X}$.

Definujme nyní novou vektorovou operaci $[\boldsymbol{a} \cdot \boldsymbol{b}]$ dvou vektorů $\boldsymbol{a} = (a_1; a_2; \ldots; a_n)$ a $\boldsymbol{b} = (b_1; b_2; \ldots; b_n)$ nějakého stejného lineárního prostoru dimenze n pravidlem $[\boldsymbol{a} \cdot \boldsymbol{b}] = (a_1 \cdot b_1; a_2 \cdot b_2; \ldots; a_n \cdot b_n)$ a nazvěme ji *součin prvek po prvku.*

Za součin snímku $\mathbb{X}$ s maskou $\mathbb{M}_k$ budeme označovat produkt $\mathbb{P}_k$, zapsatelný jako vektor $\boldsymbol{p_k} = [\boldsymbol{x} \cdot \boldsymbol{m_k}]\,, \boldsymbol{p_k} \in \langle -1{,}0; 1{,}0\rangle^{h\cdot w}$.

Nakonec c_k bude označovat součet všech prvků vektoru $\boldsymbol{p_k}$, tj. $c_k = \sum_{i=1}^{h\cdot w} p_{k,i}$. Vzhledem k definici vektoru $\boldsymbol{p_k}$ lze číslo c_k počítat jednoduše jako skalární součin $c_k = \langle \boldsymbol{x} \cdot \boldsymbol{m_k} \rangle$.

Za porovnání snímku $\mathbb{X}$ s maskou $\mathbb{M}_k$ tak budeme označovat výpočet skaláru c_k podle předpisu v předchozím odstavci. Číslo c_k budeme nazývat koeficientem podobnosti snímku $\mathbb{X}$ a masky $\mathbb{M}_k$.

Nyní je třeba ukázat, že je-li maska $\mathbb{M}_i$ snímku $\mathbb{X}$ „podobnější", než maska $\mathbb{M}_j$, pak platí $c_i > c_j$. Prozkoumejme tedy vlastnosti těchto čísel. Koeficient podobnosti c_k snímku $\mathbb{X}$ a masky $\mathbb{M}_k$ je prostým skalárním součinem řádkových zápisů těchto matic. Je proto třeba zabývat se vlastnostmi vektorů $\boldsymbol{x}$ a $\boldsymbol{m_k}$ a jejich interpretacemi. Víme, že jednotlivé prvky těchto vektorů vznikají přeškálováním hodnot pixelů (tedy barev) do reálných čísel. V nich hraje vždy význačnou roli hodnota nula. Tu můžeme interpretovat jako hodnotu referenční šedé barvy. Důsledkem této interpretace bude fakt, že každý pixel je buď „spíše černý", nebo „spíše bílý" (s výjimkou pixelů barvy referenční šedé). To znamená, že nás nezajímají barvy v absolutním měřítku, ale relativně vzhledem k referenční šedé.

[4] I v tomto algoritmu jako první převádíme všechny barevné snímky do šedotónů, abychom zredukovali paměťové nároky a zjednodušili operace.

Vezměme si nyní vektor $\boldsymbol{x}$ snímku $\mathbb{X}$ a zapišme ho jako součin $\boldsymbol{x} = [\boldsymbol{s_x} \cdot \overline{\boldsymbol{x}}]$ vektorů $\boldsymbol{s_x} \in \{-1{,}0; 1{,}0\}^{h \cdot w}$ a $\overline{\boldsymbol{x}} \in \langle 0{,}0; 1{,}0\rangle^{h \cdot w}$. To odpovídá rozkladu na vektor znamének $\boldsymbol{s_x}$ a vektor absolutních hodnot $\overline{\boldsymbol{x}}$ prvků vektoru $\boldsymbol{x}$ (pro jednoznačnost definujme, že všechny prvky x_i vektoru $\boldsymbol{x}$ s hodnotou 0,0 mají kladné znaménko, tedy $\forall i \in \{1; \ldots; h \cdot w\} : x_i = 0{,}0 \Rightarrow s_{x,i} = +1{,}0$). To je možné interpretovat tak, že prvky vektoru znamének $\boldsymbol{s_x}$ udávají, zda jsou odpovídající pixely snímku $\mathbb{X}$ černé, nebo bílé, a prvky vektoru absolutních hodnot $\overline{\boldsymbol{x}}$ pak udávají intenzity těchto barev. Dále tedy můžeme $\boldsymbol{s_x}$ nazývat vektorem barev a $\overline{\boldsymbol{x}}$ vektorem intenzit vektoru $\boldsymbol{x}$.

Stejným způsobem můžeme rozložit i všechny masky, pro k-tou masku $\mathbb{M}_k$ tak dostaneme rozklad $\boldsymbol{m_k} = [\boldsymbol{s_{m_k}} \cdot \overline{\boldsymbol{m_k}}]$. Pro vektor $\boldsymbol{p_k}$ potom platí $\boldsymbol{p_k} = [\boldsymbol{x} \cdot \boldsymbol{m_k}] = [[\boldsymbol{s_x} \cdot \overline{\boldsymbol{x}}] \cdot [\boldsymbol{s_{m_k}} \cdot \overline{\boldsymbol{m_k}}]] = [[\boldsymbol{s_x} \cdot \boldsymbol{s_{m_k}}] \cdot [\overline{\boldsymbol{x}} \cdot \overline{\boldsymbol{m_k}}]] = [\boldsymbol{s_{p_k}} \cdot \overline{\boldsymbol{p}}_{\boldsymbol{k}}]$ (důkaz je triviální).

Vektor $\overline{\boldsymbol{p}}_{\boldsymbol{k}}$ je vektorem intenzit (je zřejmé, že platí $\overline{\boldsymbol{p}}_{\boldsymbol{k}} \in \langle 0{,}0; 1{,}0\rangle^{h \cdot w}$) produktu $\mathbb{P}_k$. Vidíme, že intenzita každého pixelu produktu $\mathbb{P}_k$ závisí na intenzitách téhož pixelu ve snímku $\mathbb{X}$ a masce $\mathbb{M}_k$ takto: jsou-li obě vstupní barvy intenzivní, i výstupní barva je intenzivní; je-li jedna ze vstupních barev méně intenzivní, intenzita výstupní barvy klesá; jsou-li obě barvy neintenzivní, i výstupní barva je neintenzivní (vše je třeba uvažovat relativně, protože výstupní barva bude *vždy* méně intenzivní, než obě barvy vstupní, pokud nebude aspoň jedna z nich intenzivní plně).

Znaménka prvků vektoru $\boldsymbol{p_k}$ určuje vektor barev $\boldsymbol{s_{p_k}}$. Barvy jsou výsledkem „násobení barev", kdy rovnost barev vede vždy na barvu jednu (říkejme jí pozitivní; skutečná barva záleží na interpretaci a je nepodstatná) a nerovnost vždy na barvu druhou (negativní), jak je zřejmé z násobení libovolné dvojice hodnot z množiny $\{-1{,}0; 1{,}0\}$.

Snadno již nyní nahlédneme, že horní mez součtu prvků vektoru $\boldsymbol{p_k}$ (tedy koeficientu podobnosti c_k) je dána případem, kdy všechny pixely snímku $\mathbb{X}$ i masky $\mathbb{M}_k$ jsou plně intenzivní a odpovídající si pixely mají stejnou barvu. A to je přece případ, kdy se maska $\mathbb{M}_k$ podobá snímku $\mathbb{X}$ nejvíce (jsou shodné). Pokud dále připustíme, že podobnost nedefinujeme jako stav „nejbližších barev" (jak je tomu obvyklé), ale stav „nejjistěji stejných barev", kdy míra jistoty je dána právě vektorem intenzit, můžeme konečně stanovit *kritérium maximální podobnosti* (4.6) pro nalezení třídy k^* odpovídající masce $\mathbb{M}_k$, nejvíce se podobající snímku $\mathbb{X}$.

$$k^* = \underset{k \in K}{\operatorname{argmax}}\, c_k = \underset{k \in K}{\operatorname{argmax}}\, \langle \boldsymbol{x} \cdot \boldsymbol{m_k} \rangle \tag{4.6}$$

4.4.2 Konstrukce modelu

Kritérium (4.6) a rozklad vektorů $\boldsymbol{x}$ a $\boldsymbol{m_k}$ na intenzitní a barevnou složku dává návod, jak konstruovat masky $\mathbb{M}_k$. Neboť míra podobnosti není definována na základě podobnosti barev ve schématu RGB, ale na základě míry jistoty, že se barvy shodují[5], můžeme barvami masek říci, jaké barvy v jakých pixelech očekáváme, a intenzitami těchto barev, jaký pro nás mají tyto pixely význam.

Lze tak v maskách jednoduše vymezit oblasti, které nemají pro řešení úlohy význam žádný – to je stůl, tedy plochy, na nichž není žádná z kukuřic – a ty vyplnit referenční šedou, tedy barvou zobrazovanou do nuly. Oblasti, kde očekáváme kukuřice, budou různě vyplněny bílou a černou barvou tak, aby každá maska graficky odpovídala *ideálnímu* snímku jedné konfigurace, přičemž budou použity plně intenzivní barvy, značící, že takové pixely mají velkou váhu. A protože nemáme přesnou znalost hranic kukuřic na reálných snímcích, na maskách lze jejich hranice rozostřit, což sníží význam hraničních pixelů, kde lze očekávat nerovnosti barev.

Je intuitivní, že všechny masky by měly být „stejné až na barvy", tzn. hranice kukuřic by měly být na všech maskách shodné. Kdyby toto neplatilo, mohlo by to vést ke zhoršeným výsledkům klasifikátoru, protože by každá maska započítávala jiné oblasti snímku. To vnáší požadavek, aby všechny masky byly generovány z nějaké stejné „masky masek", která by zajistila, že vektor intenzit bude pro všechny masky stejný (což odpovídá právě stejným „tvarům" masek). Tento požadavek ovšem neznamená žádnou přítěž, neboť i bez něj je výhodnější všechny masky nějakým způsobem generovat podle jednoho předpisu (viz 4.5.5), než každou masku tvořit zvlášť.

4.5 Implementace

Předchozí sekce ukázala řešení problému na teoretické úrovni a mělo by podle ní být možné sestrojit funkční řešení. Součástí naší práce ale bylo i právě vytvoření takové aplikace, což je předmětem této kapitoly.

Požadavky na vlastní aplikaci byly následující: spustitelnost na cílové platformě, rychlost, stabilita, snadný vývoj a testování na PC. Z toho důvodu byl použit efektivní, kompilovaný jazyk C++. Aplikace byla rozdělena na 2 programy – **rozkuk** (viz 4.5.3 a 4.5.4) , tj. hlavní program realizující rozpoznávání, překládaný pro obě platformy, a **maskgen** (viz 4.5.5), pomocnou aplikaci sloužící ke generování masek, překládanou pouze pro PC. Pro urychlení vývoje a zároveň zajištění vyšší rych-

[5]Toto pojetí může připomínat fuzzy logiku a při pohledu na každý samostatný jeden pixel to je vlastně správná představa.

losti i stability byla pro mnohé výpočty použita knihovna OpenCV, jak popisuje následující podkapitola.

4.5.1 Použité funkce knihovny OpenCV

Základní informace o knihovně OpenCV byly uvedeny v části 3.1.2. Zde budou stručně popsány jen některé funkce použité v této práci (pochází z balíčků *cxcore* (struktury a práce s nimi na abstraktní úrovni), *cv* (konkrétní grafické operace) a *highgui* (ovládání kamery, přístup do filesystemu, GUI)). Podrobný manuál lze nalézt ve zdroji [10].

Pro samotné řešení úlohy, jak ukazuje kritérium (4.5), potřebujeme pouze skalární součin dvou vektorů, k čemuž slouží funkce `double cvDotProduct(const CvArr* src1, const CvArr* src2)`. Jejím vstupem jsou dvě stejně velká pole knihovního typu `CvArr*`. To je generický typ, zahrnující např. `CvMat*` (matici) nebo `IplImage*` (obrázek). Tyto struktury lze vytvořit voláním `CvMat* cvCreateMat(int rows, int cols, int type)`, resp. `IplImage* cvCreateImage(CvSize size, int depth, int channels)`, nebo je zkopírovat pomocí `CvMat* cvCloneMat(const CvMat* mat)`, resp. `IplImage* cvCloneImage(const IplImage* image)`.

K libovolné lineární transformaci všech prvků pole je přímo určena funkce `void cvConvertScale(const CvArr* src, CvArr* dst, double scale, double shift)`, což je využito při normalizaci hodnot. Argumenty `scale` a `shift` určují multiplikativní a aditivní členy funkce, ty lze vypočítat ze znalosti minimální a maximální hodnoty pole, k čemuž slouží užitečná funkce `void cvMinMaxLoc(const CvArr* arr, double* minVal, double* maxVal, CvPoint* minLoc, CvPoint* maxLoc, const CvArr* mask)` (poslední tři argumenty jsou volitelné a pro tento účel nepotřebné). Funkci `convertScale` by bylo možné použít i pro realizaci transformace T (popsané v (4.4.1)), nicméně protože je to operace, kterou je nutné vykonat nad každým kamerovým snímkem, bylo vhodné najít rychlejší alternativu – tou je funkce `void cvLUT(const CvArr* src, CvArr* dst, const CvArr* lut)`, která provádí transformaci pole `src` pomocí předem připravené vyhledávací tabulky `lut` (lookup table).

Pro ladicí účely se ve variantě rozkuku pro PC zobrazují i mezivýsledky (obr. 4.8), které dávají lépe pochopit příčiny chování klasifikátoru. V některých podmíněně překládaných funkcích se proto provádí např. i definovaný *součin prvek po prvku*, k čemuž slouží funkce `void cvMul(const CvArr* src1, const CvArr* src2, CvArr* dst, double scale)`. Obdobně *součet prvek po prvku*, funkce `void cvAdd(const`

`CvArr* src1, const CvArr* src2, CvArr* dst, const CvArr* mask)`.

Dosud se jednalo o abstraktní operace z nejdůležitějšího balíčku *cxcore*. Pro několik operací s obrázky bylo ale vhodnější použít funkce pro to určené, z balíčku *cv*. Jednalo se především o převod obrázku z modelu RGB do stupní šedi funkcí `void cvCvtColor(const CvArr* src, CvArr* dst, int code)`. Její poslední argument slouží ke specifikaci typu barevné konverze (lze tedy provádět i převody z/do jiných modelů). K prahování šedotónového obrázku byla použita funkce `double cvThreshold(const CvArr* src, CvArr* dst, double threshold, double maxValue, int thresholdType)`. A při generování masek v programu maskgen byla velmi užitečná funkce `void cvSmooth(const CvArr* src, CvArr* dst, int smoothtype, int param1, int param2, double param3, double param4)` pro rozostření obrázku (a tedy hran kukuřic).

Častěji byl využit balík *highgui*. Pro přístup ke kameře totiž nejprve bylo třeba inicializovat strukturu `CvCapture` jeho funkcí `CvCapture* cvCaptureFromCAM(int index)`, poté teprv mohly být snímky získávány voláním funkce `IplImage* cvQueryFrame(CvCapture* capture)`. Pro jejich zobrazení bylo nutné vytvořit grafické okno funkcí `int cvNamedWindow(const char* name, int flags)`, v němž pak mohly být zobrazovány funkcí `void cvShowImage(const char* name, const CvArr* image)`, přičemž hodnota prvního argumentu specifikovala název okna a musela být v obou funkcích stejná. Pro zajištění interakce s uživatelem bylo třeba periodicky volat `int cvWaitKey(int delay)`, čekající na uživatelský vstup z klávesnice. Za zmínku stojí ještě funkce `IplImage* cvLoadImage(const char* filename, int iscolor)` a `int cvSaveImage(const char* filename, const CvArr* image)` pro načtení, respektive uložení obrázku z/do souboru.

4.5.2 Rozpoznávání v praxi

Princip rozpoznávání konfigurací kukuřic byl vysvětlen v sekci 4.4.1. Zde bude popsán v té podobě, jak byl naimplementován.

Prvním rozpoznávacím krokem pochopitelně musí být pořízení barevného kamerového snímku scény. Z něj je nejprve vypočítán práh jako aritmetický průměr všech barevných složek všech pixelů obrázku sečtený s nějakou aditivní konstantou, nalezenou experimentálně (viz 4.5.3, popis stavu `IMAGE`). Práh je použit jako hodnota referenční šedé pro převod snímku do potřebné datové reprezentace (matice reálných čísel; viz 4.4.1 a 4.5.1). Důsledkem toho je fakt, že obecně nelze využít celý uváděný cílový rozsah hodnot $\langle -1{,}0; 1{,}0\rangle$, aby některá barva nebyla transformována do hod-

noty v absolutní hodnotě větší než jedna, jak je žádáno. V praxi je proto obvykle část rozsahu nevyužita tak, aby byl vždy jeden z extrémů použit a hodnota 0.0 odpovídala vypočtenému prahu. (K využití celého rozsahu dojde pouze ve speciálním případě, kdy je práh roven hodnotě 127,5.)

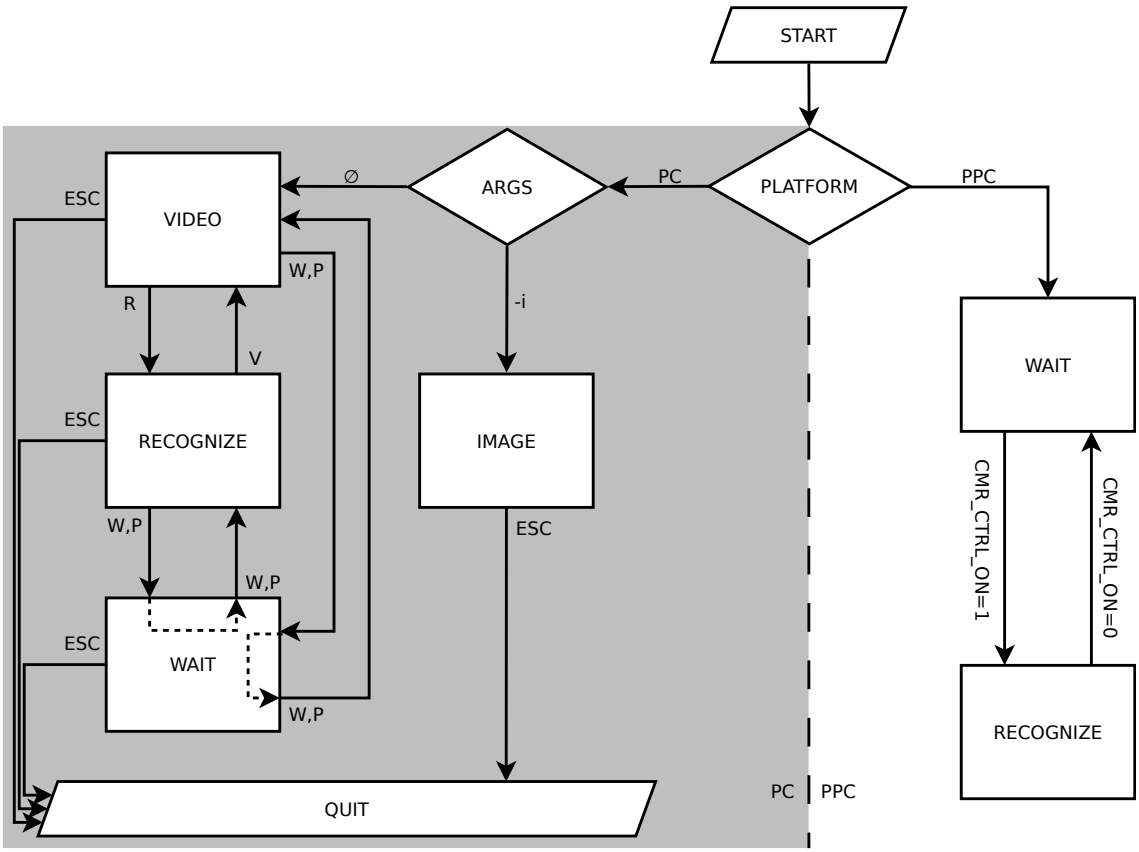
Následně je provedeno porovnání transformovaného snímku se všemi maskami bočních konfigurací a výběr nejpravděpodobnější třídy dle kritéria (4.6). Výsledkem je celé číslo z množiny $\{1; \ldots; 9\}$ odpovídající rozpoznané boční konfiguraci. Publikovaným výstupem ovšem není tato hodnota, nýbrž její „vážený modus" s výsledky z několika předchozích měření. Váženým modem rozumějme prvek nikoliv nejčastější, ale s největším součtem vah. K vážení jsou přitom použity „míry spolehlivosti dílčích výsledků", tj. vzdálenosti nejlepších řešení od druhých nejlepších. Pro vysvětlení: je-li snímku $\mathbb{X}$ nejvíce podobná maska $\mathbb{M}_i$ a druhá nejvíce podobná je maska $\mathbb{M}_j$, pak pro koeficienty podobnosti (definované v 4.4.1) c_i a c_j těchto masek platí $c_i > c_j$, a tedy rozdíl $c_i - c_j > 0$ můžeme považovat za vzdálenost nejlepšího řešení od druhého nejlepšího řešení, neboli určitou míru správnosti výsledku. Totéž proběhne zvlášť i pro středové konfigurace, jen s rozdílem výstupní množiny $\{1; \ldots; 4\}$.

4.5.3 Struktura programu

Jak již bylo zmíněno, výkonnou rozpoznávací jednotkou se stal program nazvaný *rozkuk*. Byl koncipován jako stavový automat, jehož schéma ilustruje obrázek 4.7. Je nutno podotknout, že pro snadné porovnání obou variant programu (tj. pro PC i PPC) je toto schéma záměrně zobrazuje dohromady a „výběr" platformy je zde naznačen jako rozhodovací blok uvnitř programu, ačkoliv se ve skutečnosti jedná o dva samostatné binární soubory, každý spustitelný jen na jedné platformě, tzn. výběr probíhá již v době překladu.

Výchozím bodem programu je blok `START`. V něm dochází k inicializaci, registraci do systému ORTE a připojení kamery. Ve variantě pro PC je ještě zinicializováno základní GUI (okno videa). Poté již dochází k přepnutí do některého ze stavů programu, v závislosti na cílové platformě.

Jako první bude popsán nejdůležitější stav `RECOGNIZE`. Po přepnutí do něj jsou nejprve ze souborů načteny obrázky masek (v závislosti na aktuální herní barvě, přečtené z ORTE) vygenerované programem maskgen (viz 4.5.5). Ty jsou převedeny do potřebné datové reprezentace (matice reálných čísel; viz 4.4.1 a 4.5.1). Poté již nastartuje rozpoznávací cyklus, jehož jedna iterace byla popsána v sekci (4.5.2). Cyklus se opakuje, dokud ORTE nesignalizuje žádost o ukončení rozpoznávání. Před opuš-



Obrázek 4.7: Schéma stavů programu rozkuk. Zachycuje možnosti přechodů mezi stavy pro testovací (PC; na obrázku šedě) i cílovou (PPC) platformu.

těním stavu ještě dochází k uvolnění všech masek z paměti, aby se snížily paměťové nároky programu, je-li ve stavu, v němž masky nejsou potřeba.

V praxi mohou nastat dvě situace – buď rozpoznávat chceme, a program proto běží ve stavu `RECOGNIZE`, nebo nechceme, a proces můžeme vlastně ukončit. Teoreticky tak žádný jiný stav nepotřebujeme. Protože je ale vytváření a ukončování procesů výpočetně relativně náročná operace, zdá se být vhodnější proces raději uspat, aby bylo možné ho v případě potřeby zase rychle uvést do provozu. Kromě toho na cílové platformě neustále běží kontrolní proces, který zajišťuje restart procesů v případě jejich pádu. Žádný proces se proto standardně neukončuje, protože by to bylo považováno za jeho pád a byl by tedy znovu nastartován. Z pohledu operačního systému by zřejmě bylo vhodné proces jako takový na systémové úrovni zastavit a v případě potřeby ho zase probudit. Výhodou by bylo nulové využití procesorového času v době čekání, nevýhodou ale potřeba řízení zvnějšku jiným procesem. Bylo proto vhodnější přidat čekací stav `WAIT`, ve kterém se jen periodicky testuje,

nebyla-li splněna ukončovací podmínka nastavitelná přes ORTE, a do té doby se čeká. Po jejím splnění je vždy přepnuto do stavu, ze kterého se do čekacího stavu přešlo.

Více stavů již v reálném nasazení skutečně nepotřebujeme. Jak je také vidět ze schématu 4.7, na cílové platformě jiné stavy využity nejsou. Výchozím stavem po startu procesu je `WAIT`, v němž se čeká na potřebu rozpoznávání. Pomocí ORTE je pak možné se stále přepínat mezi stavy `WAIT` a `RECOGNIZE`, nelze však program standardní cestou ukončit.

Pro platformu PC jsou ovšem k vývojovým a testovacím účelům implementovány ještě dva další stavy – `VIDEO` a `IMAGE`. Začněme stavem `VIDEO`, výchozím pro testovací platformu. Jeho jediným úkolem je zobrazovat v reálném čase obraz z kamery. Slouží tedy pouze ke sledování kamerového obrazu, což bylo užitečné především v období vývoje aplikace. Hlubší využití tento stav nemá, slouží jen jako brána do stavů `WAIT` a `RECOGNIZE`.

Poslední stav – `IMAGE` – stojí poněkud stranou. Ke svému běhu vůbec nevyužívá kameru, slouží totiž k „offline" sledování chování aplikace a ladění jejích parametrů. Jedná se vlastně o stav `RECOGNIZE` spuštěný na jediném snímku, předaném jako argument programu (podrobnosti viz 4.5.4). Umožňuje interaktivně korigovat prahovou hodnotu (intenzitu referenční šedé) nastavováním aditivní konstanty (viz 4.5.2) a souběžným sledováním dopadů upraveného prahu na kvalitu rozpoznávání.

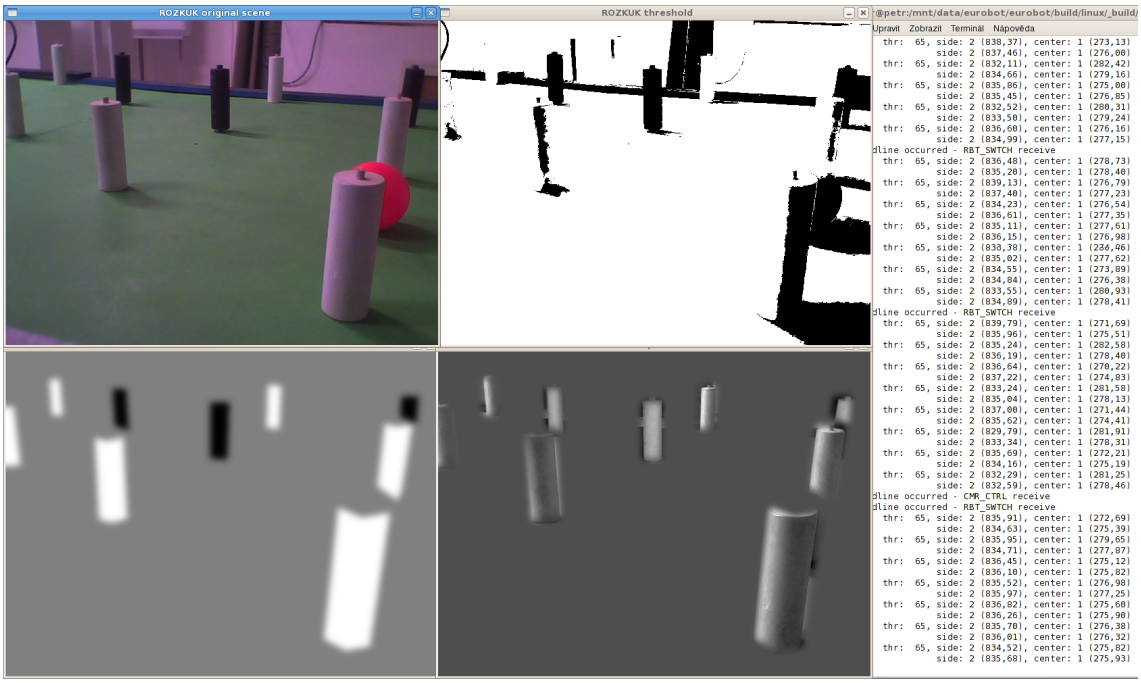
4.5.4 Použití

Funkce aplikace rozkuk vyplývají z předchozí kapitoly, kde byly popsány její jednotlivé stavy. Zde již proto bude popsáno pouze její ovládání, sloužící především k přepínání mezi nimi.

Začněme pro změnu verzí pro PC. Jak je vidět na schématu 4.7, výchozí stav není přímo určen. Záleží totiž na argumentech programu. Běžně je ovšem spouštěn bez argumentů – první operací pak je kontrola připojení kamery. Není-li kamera nalezena, vypisuje se v příkazové řádce periodicky chybové hlášení *„NULL capture (is camera connected?)"*. Po jejím nalezení program přechází do stavu `VIDEO` a zobrazí se okno s titulkem *„ROZKUK original scene"*, v němž je přehráváno video, tj. aktuální obraz z kamery. To je užitečné pro jeho prosté sledování (např. při sestavování scény) a případné ukládání snímků stiskem klávesy **S** na klávesnici. Snímky se ukládají do aktuálního pracovního adresáře pod názvy *„snapshotXX.png"*, kde *XX* zastupuje následující, ještě nevyužité číslo snímku. Dále klávesa **R** slouží pro přepnutí do stavu

`RECOGNIZE`, klávesy **W** a **P** (rovnocenně) do stavu `WAIT`.

Nejzajímavější je samozřejmě stav `RECOGNIZE`, do kterého se dá vstoupit jedině výše zmíněným způsobem přes stav `VIDEO`. V rozpoznávacím stavu se na obrazovce zobrazují 4 okna, jak je zachyceno na obrázku 4.8.



Obrázek 4.8: Screenshot aplikace rozkuk běžící na PC ve stavu `RECOGNIZE`.

Okno *„ROZKUK original scene"* již známe ze stavu `VIDEO`, jeho funkce se nikterak nezměnila. Zbylá tři okna poskytují informace užitečné pro sledování průběhu rozpoznávacích výpočtů. Ačkoliv lze s jednou maskou snímek porovnat použitím jediné funkce knihovny OpenCV (`double cvDotProduct(...)`, viz 4.5.1), jak tomu také v praxi je, pro ladicí účely jsou v těchto oknech účelně zobrazovány „mezivýpočty".

Vpravo nahoře tak vidíme okno *„ROZKUK threshold"* zobrazující oprahovaný snímek, prahem je přitom hodnota referenční šedé použité pro daný snímek (její nalezení je popsáno na začátku části 4.5.2). Černé části jsou programem viděny jako „spíše černé" a tudíž transformovány do hodnot z intervalu $\langle -1{,}0; 0{,}0\rangle$, zatímco ty zbylé, „spíše bílé", jsou transformovány do $(0{,}0; 1{,}0\rangle$. To umožňuje sledovat, jestli je předpoklad funkčnosti algoritmu – vhodný práh – dobře zvolen.

Levé dolní okno (*„ROZKUK mask"*) pro přehlednost zobrazuje rozpoznanou konfiguraci, resp. sjednocení konfigurace boční a středové, je to tedy grafická podoba výstupu aplikace.

Ten je podložen součinem snímku s vybranou maskou, zobrazeným ve čtvrtém

okně, pojmenovaném *„ROZKUK product"*. Na něm je vidět, jak dobře vítězná maska odpovídá zkoumanému snímku. Plocha mezi kukuřicemi odpovídá referenční šedé (číselně, nikoliv barevně!), vlastní kukuřice (resp. jejich produkty) by pak měly být ideálně ve všech pixelech světlejší, než zobrazená referenční šedá (což by odpovídalo „kladným hodnotám" pixelů). Jak ale vidíme, některé části jsou tmavší – konkrétně jde o oblasti bílých kukuřic, které jsou v okně *„ROZKUK threshold"* zobrazeny černě, a naopak. To jsou místa nepřesností, která snižují spolehlivost výsledku, a je proto snaha se jejich výskytu vyvarovat.

V příkazové řádce (na obrázku 4.8 vpravo) se při rozpoznávání vypisují průběžné výsledky. Pro každý snímek jsou vypsány 2 řádky. První pro samotný snímek (aktuální práh, indexy obou nejlepších konfigurací a jejich vzdálenosti od druhých nejlepších). Druhý pro „vážený modus" několika posledních snímků (opět oba indexy a vzdálenosti), tj. skutečný výsledek publikovaný přes ORTE, jak je vysvětleno v sekci 4.5.2.

I v rozpoznávacím stavu je možné uložit aktuální snímek klávesou **S**. Zpět do stavu `VIDEO` se přechází stiskem klávesy **V**, čekací stav se zapne opět klávesami **W** nebo **P**.

Do stavu `WAIT` se tedy z obou dříve uvedených stavů přepíná klávesami **W** či **P**, tytéž se ovšem používají i pro návrat zpět do stavu předchozího. Klávesy ovšem jen emulují řízení pomocí ORTE, které lze použít (a testovat) také.

Tímto končí popis standardní větve programu přeloženého pro PC. Zbývá zde ještě samostatná větev, reprezentovaná stavem `IMAGE`, k jejímuž startu dojde uvedením argumentů programu **-i *filename***, kde *filename* je cesta ke snímku v souborovém systému. Jak již bylo zmíněno, stav `IMAGE` je vlastně totéž jako stav `RECOGNIZE`, s tím rozdílem, že vstupní obraz není přijímán z kamery, ale je simulován snímkem ze souboru. Pro ladění aditivní složky prahu slouží klávesy + a `-`. Z tohoto stavu není možné se přepnout do jiného, je pouze možné ukončit program klávesou **Esc**, stejně jako ve všech ostatních stavech na PC.

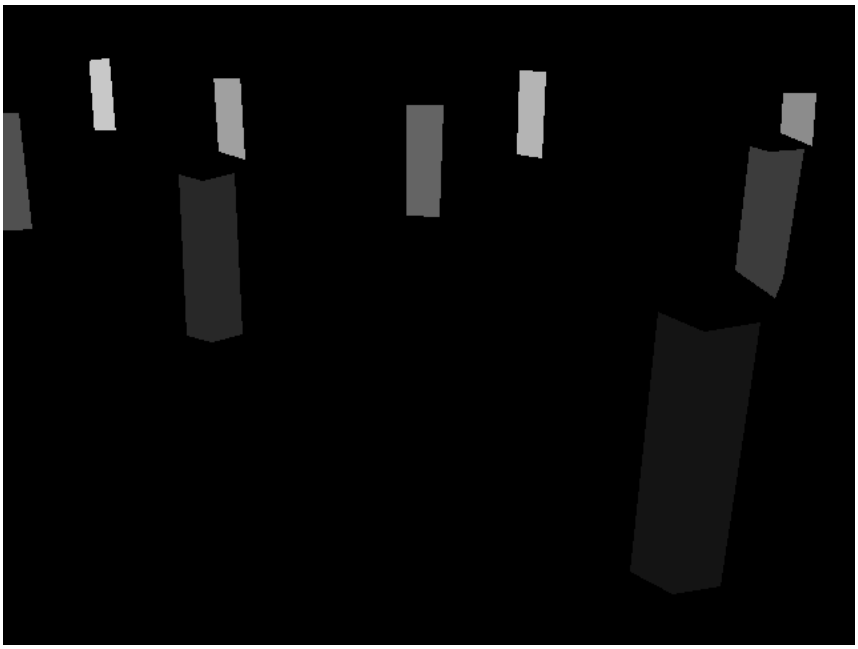
Na cílové platformě je aplikace oproštěna od všech nedůležitých funkcionalit, neposkytuje žádné GUI ani jiné vlastní uživatelské rozhraní. Ke komunikaci dochází pouze prostřednictvím ORTE – proměnná `bool orte.camera_control.on` slouží k vlastnímu řízení, tedy zapínání/vypínání rozpoznávání, což je realizováno přechodem z/do čekacího stavu `WAIT`. Aktuální týmová barva je uložena v proměnné `CORBA_octet orte.robot_switches.team_color` (při změně hodnoty v průběhu rozpoznávání je třeba načíst masky pro opačnou stranu).

Výstup aplikace je publikován opět pomocí ORTE, čísla rozpoznaných konfigurací jsou ukládána do proměnných `CORBA_octet orte.camera_result.side`

a `CORBA_octet orte.camera_result.center`, hodnoty důvěryhodnosti výsledků pak do `CORBA_double orte.camera_result.sideDist`, resp. `CORBA_double orte.camera_result.centerDist`. Jako chybový výstup slouží proměnná `CORBA_error orte.camera_result.error`.

4.5.5 Pomocné aplikace

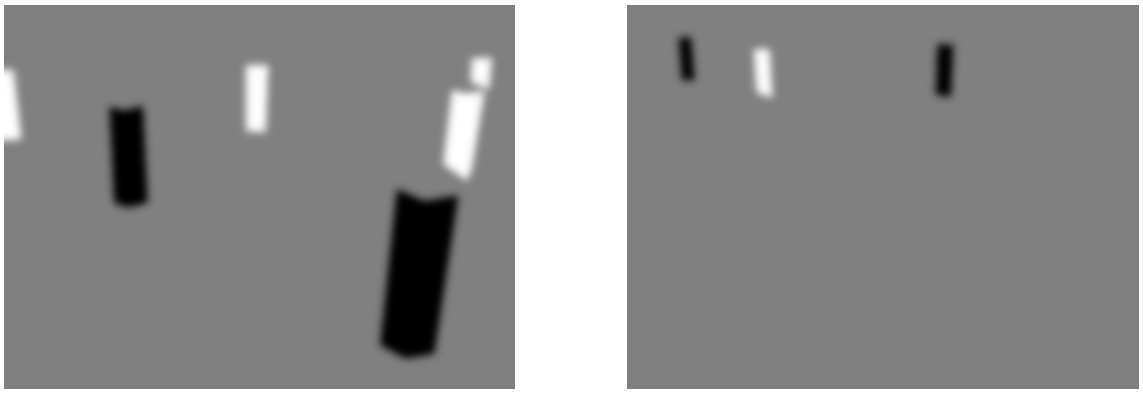
Pro usnadnění tvorby masek byla vytvořena pomocná aplikace nazvaná *maskgen*. Její účel je jednoduchý – generovat všechny masky z jedné šablony. Ve skutečnosti je třeba použít šablony dvě, neboť pohledy z modrého a žlutého startoviště jsou odlišné (a ne zcela symetrické). Šablona je vlastně obyčejný obrázek, na jehož černém pozadí jsou vyneseny přibližné tvary kukuřic, každý jiným (přesně specifikovaným) odstínem šedi, jak je patrné z obrázku 4.9. V aplikaci maskgen je definováno, jaký odstín odpovídá jaké kukuřici, což umožňuje provést barevnou náhradu tak, aby výstupem byla maska požadované konfigurace.



Obrázek 4.9: Maska masek pro modré startoviště. Zachycuje hrubé tvary viditelných kukuřic, každý jiným odstínem šedi.

Spouštěcí sekvence programu zní `./maskgen inputfile [-b value] [-y]`, povinný argument `inputfile` je název vstupní masky. Nepovinný argument `-b` umožňuje zadat šířku okna pro finální rozostření (není-li zadáno, zůstává výstup ostrý). Použitím nepovinného argumentu `-y` se říká, že chceme generovat masky pro žluté

startovní pole (v opačném případě pro modré) – tato volba má vliv pouze na pojmenování výstupních souborů. Spuštěním příkazem `./maskgen bmasks_mask.png -b 30` (`bmasks_mask.png` je název masky z obrázku 4.9) dostaneme 13 výstupních obrázků (9 pro boční a 4 pro středovou konfiguraci). Dva z nich ukazuje obrázek 4.10.



Obrázek 4.10: Boční (vlevo) a středová maska vygenerované z masky 4.9 s použitým rozostřením (velikost okna 30).

4.5.6 Zdrojové soubory

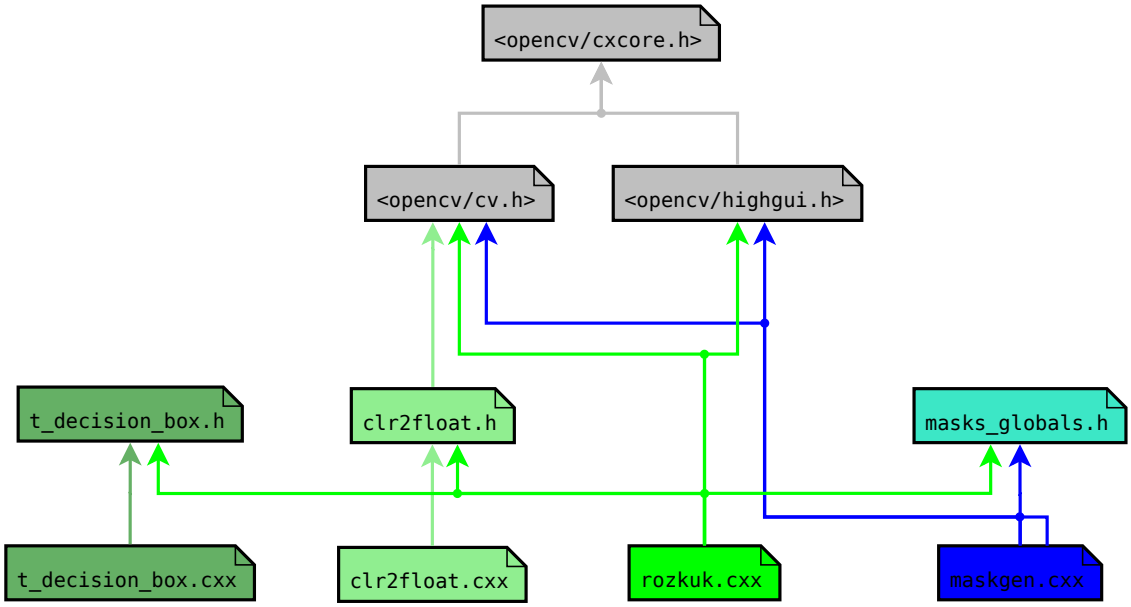
Zdrojový kód je rozdělen podle funkce do několika souborů, jak ukazuje obrázek 4.11. Společný základ většiny z nich tvoří (vedle standardních knihoven) soubory knihovny OpenCV. Ta již byla popsána v části 4.5.1.

Jádrem aplikace *rozkuk* je soubor *rozkuk.cxx*, obsahující většinu zdrojového kódu aplikace. Zajišťuje inicializaci i uvolnění zdrojů aplikace, komunikaci s ORTE, načítání masek ze souboru, implementuje všechny stavy (viz 4.5.3), přechody mezi nimi i uživatelské rozhraní.

Pro převod snímků kamery z RGB reprezentace do matice hodnot typu `float` využívá funkci `int clr2float(...)` definovanou v souboru *clr2float.cxx*. Tatáž funkce je využita i pro převod masek do matice stejného typu, ačkoliv výchozí formát není RGB, ale stupně šedi.

Soubor `t_decision_box.cxx` implementuje jednoduchou třídu `TDecisionBox`, jejíž objekty ukládají požadovaný počet posledních výsledků rozpoznávání a realizují „vážený modus" pro výběr vítězného výsledku, jak je vysvětleno v závěru kapitoly 4.5.2.

Poslední závislostí potřebnou pro sestavení aplikace *rozkuk* je hlavičkový soubor *masks_ globals.h*, který definuje hodnoty konstant sdílených s aplikací *maskgen*.



Obrázek 4.11: Schéma závislostí zdrojových souborů – *rozkuk* v odstínech zelené, *maskgen* v odstínech modré. Knihovní hlavičkové soubory šedě.

Příkladem takových konstant může být formát názvů souborů masek nebo celkový počet masek.

Tento soubor je tedy vkládán (direktivou `#include`) i do souboru *maskgen.cxx*, implementujícího veškerou funkcionalitu aplikace *maskgen*, tedy inicializaci i uvolnění alokovaných zdrojů, zpracování argumentů programu (tj. interakci s uživatelem), načtení masky masek a generování a uložení souborů masek.

4.6 Testování

Testování navrženého klasifikátoru probíhalo již v průběhu vývoje, aby bylo dosaženo kvalitních výsledků – tímto způsobem byl například navržen způsob výpočtu prahu, jak je popsán v úvodu kapitoly 4.5.2. Vysvětlením, proč byl použit právě takovýto výpočetní postup, je tedy skutečnost, že vedl k dobrým výsledkům.

Obdobně použití „váženého modu" při výběru výsledku klasifikátoru nebylo v původním algoritmu zamýšleno a bylo důsledkem snahy zvýšit stabilitu klasifikátoru a jeho robustnost vůči okamžitým výchylkám, pozorovaným v průběhu testování (např. při změně osvětlení).

V této části bude popsáno několik experimentů a finální výsledky ze světového kola soutěže.

4.6.1 Bezchybová datová sada

V kapitole 4.3 o bayesovském klasifikátoru byla pro testování použita datová sada čítající 5 snímků pro každou konfiguraci i barvu. Jak tam bylo zmíněno, pořízen byl aplikací *rozkuk* pro různé výchylky od ideální polohy robota tak, aby byly všechny snímky *rozkukem* klasifikovány správně. Již toto dává v kontextu s výsledky popsaného bayesovského klasifikátoru určitou představu o kvalitě klasifikátoru zkoumaného.

Tabulka 4.2 zachycuje kvality výsledků naměřených aplikací *rozkuk*, tj. vzdálenosti nejlepších (správných) ohodnocení od ohodnocení druhých nejlepších. Otázkou je, co lze z tabulky vyčíst.

	barva = modrá			*barva = žlutá*		
	MIN	*AVG*	*MAX*	*MIN*	*AVG*	*MAX*
S1	731,07	795,24	820,50	795,69	925,96	1012,75
S2	706,84	754,27	779,39	1114,64	1146,64	1199,92
S3	456,27	650,36	805,20	1062,97	1175,57	1262,95
S4	444,41	1023,83	1247,66	1065,36	1290,65	1557,82
S5	1547,96	1739,82	1872,87	1641,89	2193,26	2541,33
S6	1313,18	1522,80	1710,12	1252,73	1711,86	2123,58
S7	681,39	1206,39	1478,92	895,09	1294,56	1442,53
S8	1038,37	1366,01	1549,52	1195,64	1380,09	1485,05
S9	95,96	581,14	1163,65	1171,27	1220,63	1265,70
C1	117,17	183,97	292,43	70,06	125,12	142,31
C2	124,39	227,63	309,61	84,61	122,26	147,68
C3	123,87	288,35	372,33	161,24	236,84	291,23
C4	241,63	320,25	357,18	229,52	269,16	305,91

Tabulka 4.2: Výsledky klasifikátoru na bezchybové datové sadě. Řádky tabulky odpovídají různým konfiguracím (*S?* pro boční, *C?* pro středové), sloupce zachycují (pro obě barvy) nejnižší, průměrnou a maximální vzdálenost klasifikovaného (správného) výsledku od druhého nejlepšího ohodnocení.

To závisí značně na tom, jakým způsobem byly snímky datové sady pořizovány. Měření neprobíhalo systematicky, ale náhodně tak, aby jeden snímek přibližně odpovídal ideální poloze robota a ostatní byly pořízeny při odchylkách rámcově odpovídajících očekávatelným odchylkám v praxi (kritériem zároveň byl již zmíněný požadavek na bezchybovost datové sady; ve většině případů ovšem toto kritérium bylo splněno bez dodatečných zásahů). Takový postup se zdá být nekonvenční, v zásadě by ale mohl být smysluplnější než systematické testování výsledků pro přesně dané úhly. Důvodem k tomu je fakt, že testované odchylky přibližně odpovídaly

skutečným odchylkám v praxi, takže toto měření může podávat přesnější výsledky, nežli měření uměle vykonstruované.

Z tohoto pohledu tedy lze výsledky tabulky 4.2 považovat za mírně vychýlený odhad skutečných výsledků klasifikátoru. Relevanci nemají v oblasti úspěšnosti klasifikátoru, ale mohou pomoci vytvořit představu o tom, jaké hodnoty lze v praxi většinou očekávat. Lze například sledovat rozdíly mezi jednotlivými konfiguracemi – pozorujeme třeba významný rozdíl mezi prvními třemi bočními konfiguracemi oproti těm ostatním (bočním). Tyto konfigurace se shodují v nastavení středovější skupiny kukuřic, což je zřejmě důvodem této skutečnosti. Také lze říci, že pro žlutou barvu dosahuje klasifikátor v porovnání s barvou modrou lepších výsledků u bočních konfigurací a naopak horších u konfigurací středových. To bude zřejmě důsledkem odlišných masek pro obě barvy. Dále vidíme, že hodnoty pro boční konfigurace jsou podstatně vyšší, než ty pro konfigurace středové, což lze očekávat vzhledem k počtu kukuřic v obou skupinách i jejich vzdálenosti od kamery.

4.6.2 Experimenty s prahem

Pro otestování robustnosti klasifikátoru bylo provedeno několik experimentů. Cílem prvního z nich bylo zjistit, jak je rozhodovací mechanizmus náchylný na změnu prahu. K tomu byl využit stav `IMAGE` aplikace rozkuk, popsaný v sekcích 4.5.3 a 4.5.4. Experiment byl proveden na jednom snímku pro každou konfiguraci obou barev (celkem tedy 26 snímků) v rozsahu posunů prahu od -75 do $+25$. Výsledky ukazuje tabulka 4.3.

V ní lze pozorovat několik věcí. Především, vezmeme-li maximum ze všech minim a minimum ze všech maxim, dostaneme množinu $\{-53; \ldots; -21\}$ hodnot, které můžeme použít jako aditivní konstantu prahu. Přičtením jakékoliv z těchto hodnot k automaticky určenému základu prahu zajistíme správnost výsledku klasifikace. Pochopitelně je třeba si uvědomit, že test byl (z důvodu pracnosti) pro každou konfiguraci proveden pouze pro jeden snímek – pro reprezentatnivnější počet snímků tedy lze očekávat zúžení povoleného rozsahu. Na druhou stranu je také třeba říci, že testovací snímky byly pořizovány z různých poloh (myšleno různých odchylek od ideální polohy), takže výsledky experimentu nejsou platné jen pro jednu konstantní, ideální polohu, čímž jejich věrohodnost stoupá.

Porovnáním výsledků pro modrou a žlutou hrací barvu si lze všimnout, že hranice správnosti i optimální práh pro konkrétní konfiguraci se u obou barev (většinou) dosti podobají. To potvrzuje ne zcela samozřejmou skutečnost, že snímky jedné

	barva = modrá			*barva = žlutá*		
	MIN	*OPT*	*MAX*	*MIN*	*OPT*	*MAX*
S1	−75	−15 (944,96)	+3	−62	+6 (2107,32)	+19
S2	−70	−12 (934,62)	+17	−68	+25 (3285,59)	+25
S3	−75	−23 (810,92)	−13	−68	−20 (1367,93)	+6
S4	−54	−34 (677,50)	−21	−72	−44 (1473,61)	−2
S5	−69	+25 (4898,13)	+25	−60	+24 (6757,37)	+25
S6	−66	−6 (2132,11)	+12	−67	+1 (2451,04)	+16
S7	−75	−34 (1545,76)	+4	−74	−35 (1105,40)	+7
S8	−75	−29 (1523,25)	+10	−67	−55 (1848,60)	+24
S9	−75	−36 (1690,41)	−8	−75	−75 (1978,04)	+2
C1	−75	−69 (425,02)	−13	−75	−75 (445,94)	−9
C2	−75	−75 (569,60)	+7	−75	−75 (455,49)	−11
C3	−63	+25 (1129,72)	+25	−54	+23 (672,85)	+25
C4	−62	+25 (1112,17)	+25	−53	+25 (910,33)	+25
	−54		**−21**	**−53**		**−11**

Tabulka 4.3: Výsledky experimentu s prahem. Řádky tabulky odpovídají různým konfiguracím (*S?* pro boční, *C?* pro středové), sloupce zachycují (pro obě barvy) nejnižší, optimální a nejvyšší možnou aditivní konstantu prahu k jeho automaticky počítanému základu tak, aby klasifikátor rozpoznal konfiguraci správně (rozmezí je ohraničeno hodnotami −75 a +25). Optimální hodnota je ta, při které je dosaženo nejvyšší vzdálenosti (v závorce) od druhého nejlepšího výsledku.

konfigurace mají pro obě barvy podobný celkový jas i vhodný práh. Z tohoto důvodu tedy můžeme páry považovat za dvojice snímků stejné konfigurace bez ohledu na barvu, a tudíž vlastně na použitou sadu lze pohlížet jako na sadu dvou snímků pro každou konfiguraci. Výsledkem této myšlenkové úvahy je zvýšení věrohodnosti výsledků tohoto experimentu, protože při rozšíření pokusné sady můžeme očekávat obdobné chování, jako u snímků použitých.

4.6.3 Experimenty s redundancí

Dalším robustnostním testem bylo pozorování významu redundance ve snímcích. Použita byla opět „bezchybová" datová sada, tzn. 130 kamerových snímků hřiště (pět pro každou barvu a konfiguraci). Pro zjištění vlivu jednotlivých kukuřic na chybovost klasifikátoru byly z masek postupně vypouštěny jednotlivé plochy odpovídající kukuřicím (zkrátka jako by vždy jedna z kukuřic na stole vůbec nebyla). V maskách je takových ploch devět, důsledkem tedy bylo 1170 měření, jejichž výsledky zachycuje tabulka 4.4. Předmětem měření byl počet správně klasifikovaných snímků.

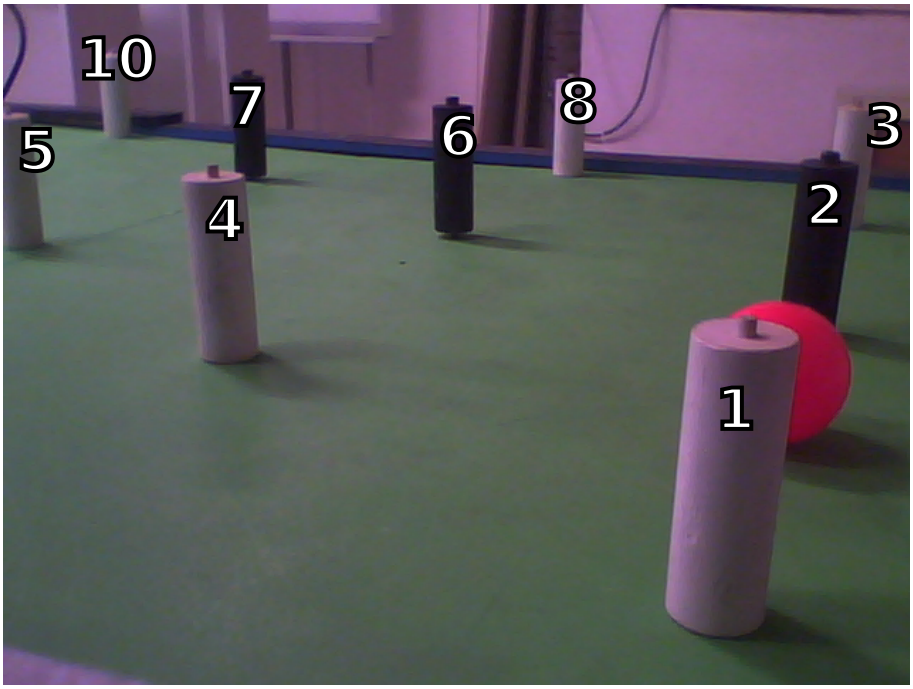
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*10*
S1	**10**	**10**	**10**	**10**	**10**	**10**	10	10	10
S2	**10**	**10**	**10**	**10**	**10**	**10**	10	10	10
S3	**10**	**10**	**10**	**10**	**10**	**10**	10	10	10
S4	**9**	**9**	**10**	**10**	**10**	**10**	10	10	10
S5	**10**	**10**	**10**	**10**	**10**	**10**	10	10	10
S6	**10**	**10**	**10**	**10**	**10**	**10**	10	10	10
S7	**10**	**9**	**10**	**6**	**10**	**6**	10	10	10
S8	**10**	**10**	**10**	**6**	**10**	**6**	10	10	10
S9	**10**	**10**	**7**	**5**	**10**	**5**	10	10	10
C1	10	10	10	10	10	10	**10**	**10**	**10**
C2	10	10	10	10	10	10	**10**	**10**	**10**
C3	10	10	10	10	10	10	**10**	**10**	**0**
C4	10	10	10	10	10	10	**10**	**10**	**0**

Tabulka 4.4: Výsledky experimentu s redundancí. Řádky tabulky odpovídají různým konfiguracím (*S?* pro boční, *C?* pro středové), sloupce různým plochám (odpovídajícím kukuřicím) vynechaným z masek. Plochy jsou číslovány stejně, jako kukuřice na obrázku 2.4. Buňky obsahují počet správně klasifikovaných snímků – červeně jsou zvýrazněny ty, kde došlo k nějakým chybám.

Nejprve si všimněme grafického zpracování tabulky. Sloupce jsou rozděleny do skupin (všimněme si dvojitých čar) tak, aby každá skupina odpovídala jedné oblasti výlučnosti, jak byly vysvětleny v části 2.2.1 a znázorněny na obrázku 2.4, který zároveň zobrazuje i čísla kukuřic, odpovídající označením sloupců tabulky. Vidíme, že poslední skupina obsahuje pouze jediný sloupec (10), protože kukuřice 9 není v zorném úhlu kamery (pro lepší představu viz obrázek 4.12).

Nyní v tabulce pozorujeme, pro které upravené masky při jakých konfiguracích klasifikátor chyboval, resp. nechyboval. Význačný smysl mají buňky, které **nejsou** zvýrazněny tučným řezem, tj. buňky centrálních konfigurací ve sloupcích 1 až 6 a buňky konfigurací bočních ve sloupcích 7, 8 a 10. Ty by totiž měly být, a jak vidíme, tak i jsou, zcela bezchybné. Důvod k tomu je ten, že chybějící plochy bočních kukuřic nemohou mít vliv na výsledek klasifikace v oblastech středových a naopak.

Tato skutečnost platí i více do hloubky, v rámci samotných oblastí výlučnosti. Z tabulky 4.4 to sice nejde poznat, ale hlubším prozkoumáním výpisu A.2 (viz přílohy), ze kterého byla tabulka vyextrahována, zjistíme, že všechny chyby klasifikátoru se týkají pouze těch oblastí výlučnosti, ve kterých chybí příslušná plocha. Například v případě chybějící plochy 1, kdy klasifikátor chyboval pouze jednou, a to u boční konfigurace *S4* (černé kukuřice 3 a 4), došlo ke klasifikaci do třídy *S5* (černé



Obrázek 4.12: Kamerový snímek hřiště (pro modrou barvu při konfiguracích *S1* a *C2*) odpovídající snímkům ze soutěže. Kukuřice jsou očíslované souhlasně s obrázkem 2.4.

kukuřice 1 a 4). Vidíme tedy, že oblast kukuřic 4 až 6 byla klasifikována správně.

Dále vidíme, že kukuřice č. 4 a 6 hrají při určování boční třídy vysokou roli – v situaci, kdy v maskách chyběly, byla totiž odpovídající oblast často klasifikována nesprávně. Naopak přítomnost kukuřice č. 5 výsledek zřejmě příliš neovlivňuje (tj. její přítomnost pouze zvyšuje redundanci, tedy jistotu správného výsledku).

Nečekaný výsledek podala skupina středových kukuřic 7-8. Vzhledem k tomu, že se tato skupina skládá pouze ze 2 kukuřic, dalo by se očekávat, že redundance zde nebude vysoká (resp. bude jen pokrývat vysoké množství šumů dané vysokou vzdáleností kukuřic od objektivu kamery), a tudíž nepřítomnost jedné z nich povede k chybám v této skupině. Překvapivě ale stačí libovolná z těchto kukuřic k bezpečnému určení správné konfigurace.

Co bylo naopak očekáváno, je výsledek experimentu při chybějící kukuřici 10. Vzhledem k tomu, že v záběru kamery chybí k ní párová kukuřice č. 9, není v této oblasti výlučnosti žádná redundance. Nepřítomnost kukuřice 10 tedy vede k „náhodné" klasifikaci v oblasti 9-10, a tudíž dochází k 50% chybě. Však také ve výpisu A.2 vidíme, že vzdálenost od druhého nejlepšího řešení je 0. V takovém případě je vybíráno první vyhovující řešení, proto jsou konfigurace *C1* i *C2* vždy správné a

naopak *C3* a *C4* vždy špatné (klasifikace tedy není náhodná, chyba jí ale odpovídá).

Jak jsme ukázali, nepřítomnost některé z ploch masky vede v případě chyby k nesprávné klasifikaci pouze v odpovídající oblasti výlučnosti, což odpovídá chybnému určení barev na čtyřech kukuřicích na celém hřišti u skupin 1-3, 4-6 a 7-8; v případě skupiny 9-10 jsou chybně určeny kukuřice dvě. Při celkovém počtu 18 kukuřic na hřišti a 1170 měření tedy bylo při tomto experimentu chybně určeno pouhých 208 z 21060 kukuřic, tedy necelé 1 %! To dokazuje, že i po snížení redundance je zachována vysoká kvalita rozhodování klasifikátoru.

4.6.4 Výsledky v soutěži

Na závěr uveďme ještě výsledky klasifikátoru v průběhu soutěže. Ta se skládala z národního, a pro tři nejúspěšnější týmy také mezinárodního kola.

V národním kole, pořádaném v Praze 1.5.2010, nebyl ještě klasifikátor použit, protože se tým potýkal se zásadnějšími problémy především v oblasti HW. Ty vedly dokonce k neúspěchu ve vyřazovací části turnaje. Díky vysokému potenciálu robota ovšem tým Flamingos získal cenu poroty, díky které mu bylo umožněno zúčastnit se mezinárodního kola společně s vítězným a druhým nejlepším týmem turnaje.

Mezinárodní kolo se konalo 26.-30.5.2010 ve švýcarském Rapperswil-Jona. Tým Flamingos se zde zúčastnil celkem šesti zápasů, přičemž v posledních třech byl využit i tento klasifikátor (do té doby nebyly jeho výsledky používány ve strategii robota). Výsledky z nich zobrazuje tabulka 4.5.

	boční		středová	
	skutečná	predikce	skutečná	predikce
4.	**3**	**3** (575,66)	**1**	**3** (185,79)
5.	**2**	**2** (1116,19)	**1**	**1** (54,32)
6.	**2**	**2** (1043,41)	**1**	**1** (67,77)

Tabulka 4.5: Výsledky v soutěži. Řádky odpovídají různým zápasům, ve sloupcích jsou zaznamenány skutečné a predikované konfigurace. Chyby jsou zvýrazněny červenou barvou.

Pozorujeme v ní jednu chybu, a to kdy při čtvrtém zápase byla středová konfigurace *C1* klasifikována jako *C3*. To odpovídá chybě ve skupině 9-10, tedy dvěma špatně klasifikovaným kukuřicím na celém hřišti. Při celkem třech měření tedy byly chybně klasifikovány 2 kukuřice z celkového počtu 54, naměřená chyba tedy byla přibližně 3,7 %.

5 Závěr

Cílem této práce bylo vytvořit klasifikátor, který by na základě snímků z kamery byl schopen s dostatečně vysokou spolehlivostí určit rozestavění černých válečků (tzv. falešných kukuřic) na stole při soutěži Eurobot. Byla vytvořena aplikace *rozkuk*, která tuto úlohu plní a navíc při spuštění z PC umožňuje dobře analyzovat a snadno ladit chod klasifikátoru. Pro klasifikaci byla použita vlastní, výpočetně přijatelně náročná metoda, pracující na principu porovnávání kamerových snímků s připravenými snímky ideálních stavů hřiště, tzv. maskami. Příprava masek zvyšuje časovou náročnost přípravy klasifikátoru, což bylo důvodem k vytvoření pomocné aplikace *maskgen*, generující tyto masky hromadně.

Experimenty ukázaly, že klasifikátor je velmi odolný vůči šumu, což zajišťuje vysoká redundance dat v obraze, a především, že chyba v jedné části hřiště neovlivňuje kvalitu rozhodování v jiné části hřiště. Díky této vlastnosti i při chybném rozpoznání konfigurace bývají většinou chybně určeny nejvýše čtyři kukuřice. Výsledky experimentů jsou podtrženy výsledky posledních třech zápasů z finále soutěže Eurobot, kam tým Flamingos postoupil přes národní kolo, a kde se umístil nejlépe z českých týmů na 9.-16. místě. Klasifikátor zde dosáhl velmi vysoké úspěšnosti 96,3 % (po přepočtu na počet chybně určených kukuřic). Ostatní výsledky, jež nebyly z různých důvodů zaznamenány, by přitom úspěšnost zřejmě ještě o něco zvýšily.

Užitečné měřítko bychom získali také regulérním otestováním odolnosti klasifikátoru vůči nepřesné poloze a orientaci robota. Toto měření však již bohužel v době potřeby nebylo možné uskutečnit – testovací stůl byl totiž mezitím rozebrán. Jak ovšem výsledky z finále potvrzují, tolerance je dostatečně vysoká na to, aby bylo možné robota na startovací pole umisťovat jednoduchou manipulací obsluhy.

Pro porovnání byl v této práci ještě otestován velmi jednoduchý bayesovský klasifikátor, který dosahoval testovací chyby podstatně vyšší. Problém byl ale zřejmě ve volbě měřených veličin, případně v jejich počtu – vhodnějším výběrem by jistě bylo možné kvalitu klasifikátoru podstatně zvýšit. Záměrem tvorby tohoto klasifikátoru ovšem bylo pouze poznat jeho chování na daném problému. Jeho nevýhodou je především potřeba vytvořit dostatečně vysoký počet snímků jako trénovacích dat, což u použité metody zapotřebí není – stačí pouze ručně nakreslit dva obrázky (masky masek), z nichž se vygenerují všechny masky.

Při testování i v praxi se vytvořený klasifikátor ukázal jako velmi použitelný, a věřím proto, že bude i v některých dalších ročnících soutěže v nějaké podobě s úspěchem použit.

6 LITERATURA

[1] Eurobot. *Contest themes archive* [online]. [cit. 2011-01-02]. Dostupné z WWW: <http://www.eurobot.org/eng/archives.php>.

[2] Eurobot. *Feed the World : Rules 2010* [online]. Document version 24.09.2009 15:33 [cit. 2011-01-02]. Dostupné z WWW: <http://www.eurobot.org/commonfiles/docs/2010/E2010_rules_and_drawing_EN.pdf>

[3] Univerzita Karlova v Praze, Matematicko-fyzikální fakulta, Katedra softwarového inženýrství. *Nakrmte svět : Pravidla 2010* [online]. Release 1. 2009 [cit. 2011-01-02]. Dostupné z WWW: <http://www.eurobot.cz/2010/E2010_Rules-CZ-v1.pdf>.

[4] *Flamingos : An Eurobot team from Czech Technical University* [online]. 2010 [cit. 2011-05-22]. Eurobot 2010. Dostupné z WWW: <http://flamingos.felk.cvut.cz/?page_id=18>.

[5] BENDA, Jan. *CANOpen komunikace pro mobilního robota* [online]. Praha, 2008. xvi, 68 s., XIX. Diplomová práce. České vysoké učení technické v Praze, Fakulta elektrotechnická. Dostupné z WWW: <http://rtime.felk.cvut.cz/dragons/repos/papers/benda-2008.pdf>.

[6] KUBIAS, Jiří. *Hardware robota pro soutěž Eurobot* [online]. Liberec, 2010. xi, 59, XIII. Diplomová práce. České vysoké učení technické v Praze, Fakulta elektrotechnická. Dostupné z WWW: <http://rtime.felk.cvut.cz/dragons/repos/papers/kubias-2010.pdf>.

[7] JAREŠ, Filip. *Implementace stavových automatů pro soutěž Eurobot 2009* [online]. Praha, 2010. ix, 46 s. Bakalářská práce. České vysoké učení technické v Praze, Fakulta elektrotechnická. Dostupné z WWW: <http://rtime.felk.cvut.cz/dragons/repos/papers/jares-2010.pdf>.

[8] SMOLÍK, Petr; PÍŠA, Pavel. *ORTE : The Open Real Time Ethernet* [online]. Czech Technical University in Prague, Department of Control Engineering. Dostupné z WWW: <http://smoliku.cz/orte/lib/exe/fetch.php?media=wiki:rtn08_orte.pdf>.

[9] BSD licenses. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 1.8.2001, last modified on 6.3.2011 [cit. 2011-03-06]. Dostupné z WWW: <http://en.wikipedia.org/wiki/BSD_licenses>.

[10] *OpenCV 2.0 C Reference* [online]. c2009 [cit. 2011-03-06]. Dostupné z WWW: <http://opencv.willowgarage.com/documentation/index.html>.

[11] AGAM, Gady. *Illinois Institute of Technology : Department of Computer Science* [online]. January 27, 2006, 2006-03-31 [cit. 2011-03-06]. Introduction to programming with OpenCV. Dostupné z WWW: <http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/index.html>.

[12] TAKAHASHI, Yasutake; NOWAK Walter; WISSPEINTNER Thomas. *Adaptive Recognition of Color-Coded Objects in Indoor and Outdoor Environments* [online]. [200?] [cit. 2011-05-11]. Dostupné z WWW: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.95.5988&rep=rep1&type=pdf>.

[13] Mahalanobis distance. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 1 December 2006, last modified on 11 May 2011 [cit. 2011-05-22]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Mahalanobis_distance>.

[14] TAO, Linmi; XU, Guangyou. *Color in machine vision and its application*. 2001-09-01 [cit. 2011-05-11]. Dostupné z WWW: <http://80.www.springerlink.com.dialog.cvut.cz/content/f2061x7017600h22/fulltext.pdf>.

[15] GEVERS, Theo; SMEULDERS, Arnold. *Color-based object recognition*. Received 22 December 1997; received for publication 4 February 1998 [cit. 2011-05-11]. Dostupné z WWW: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.75.2419&rep=rep1&type=pdf>.

[16] GEVERS, Theo; STOKMAN, Harro. *Robust histogram construction from color invariants for object recognition*. Volume 26, No. 1, January 2004 [cit. 2011-05-11]. Dostupné z WWW: <http://ieeexplore.ieee.org/search/srchabstract.jsp?tp=&arnumber=1261083&queryText%3DRobust+Histogram+Construction%26openedRefinements%3D*%26searchField%3DSearch+All>.

[17] HLAVÁČ, Václav; MATAS, Jiří. *Bayesian Decision Theory* [online]. [200?] [cit. 2011-02-12]. Dostupné z WWW: <http://cw.felk.cvut.cz/lib/exe/fetch.php/courses/a4b33rpz/2010.10-bayes.pdf>.

[18] URBAN, Martin. *Cvičení z předmětu Rozpoznávání* [online]. 5.10.2006, 10.03.2009 [cit. 2011-05-22]. Bayesovská úloha rozhodování. Dostupné z WWW: <http://cmp.felk.cvut.cz/cmp/courses/recognition/Labs/bayes/index.html>.

[19] ŠOCHMAN, Jan; FRANC, Vojtěch. *Cvičení z RPZ : Maximálně věrohodný odhad* [online]. 23.03.2006 [cit. 2011-02-13]. Dostupné z WWW: <http://cmp.felk.cvut.cz/cmp/courses/recognition/Labs/mlodhad/ml.pdf>.

[20] ROUBAL, Jiří. *Jirkovy stránky* [online]. 12.01.2009 [cit. 2011-05-20]. Dostupné z WWW: <http://support.dce.felk.cvut.cz/pub/roubalj/>.

A Výpisy aplikace rozkuk

Pro živé sledování výstupu aplikace *rozkuk* bylo na displeji robota vyhrazené pole zobrazující čísla rozpoznané boční a středové konfigurace (hodnoty byly propagovány skrze ORTE). Vedle toho měl ale program také výstup do konzole, který byl logován přesměrováním do souboru. Takových souborů je na přiloženém CD poměrně vysoké množství. V tomto dodatku jsou pouze 2 zkrácené výpisy – první (A.1) pouze pro vysvětlení jejich obsahu, druhý (A.2) pro jeho vysoký význam při vysvětlování výsledků experimentu v části 4.6.3.

A.1 Výpis 6. zápasu

V této části bude vysvětlen obsah výpisů aplikace rozkuk na souboru *logs/rozkuk-Ne kvě 30 10:52:19 CEST 2010.log* ze 6. zápasu v mezinárodním kole soutěže. Výpis je zde podstatně zkrácen, vynechané řádky jsou naznačeny výpustkou.

Zpráva na prvním řádku informuje o hrací barvě, vyčtené z ORTE – v tomto případě odpovídá žluté.

Druhý řádek potvrzuje úspěšný start aplikace. Jejím výchozím stavem je `WAIT` (viz schéma na obrázku 4.7), což se projevuje následujícím řádkem.

Po něm pozorujeme další zprávu od ORTE, tentokrát o změně hodnoty proměnné `CMR_CTRL_ON` na `1`, což vede k přechodu do stavu `RECOGNIZE`. Prvním krokem je načtení všech masek žluté barvy, poté začíná rozpoznávací cyklus. V každé iteraci jsou vytištěny 3 řádky – nepodstatné varování knihovny OpenCV, aktuální práh s rozpoznanými konfiguracemi (v závorce je jejich vzdálenost od druhého nejvhodnějšího řešení) pro aktuální snímek a hodnoty konfigurací publikované do ORTE (vypočtené váženým modem).

Kameře po jejím spuštění chvilku trvá, než se přizpůsobí okolnímu osvětlení, jak vidíme z výpisu prvních pěti snímků. Poté je již podávána informace o rozpoznávaných konfiguracích kukuřic na hřišti. Je nutné upozornit, že hodnoty konfigurací ve všech výpisech jsou **o jedničku nižší**, než jak jsou konfigurace číslovány v pravidlech i v této práci! Tyto hodnoty se mění postupně tak, jak během přípravného času rozhodčí upravují pozice kukuřic do nově vylosovaných konfigurací. Tato část výpisu je vypuštěna, je zobrazen pouze jeden takový přechod (druhý blok výpisu).

V poslední části výpisu vidíme přepnutí proměnné `CMR_CTRL_ON` do nuly, což způsobí přechod programu do stavu `WAIT` (o čemž informují následující řádky). To je provedeno v okamžiku odstartování robota. Za rozpoznanou konfiguraci pak jsou považovány poslední tisknuté hodnoty.

```
orte: robot_switches changed: color 1
rozkuk started, camera connected successfully
waiting...
orte: camera_control changed: ctrl 1
Mask ./mask00sy.pnm successfully loaded.
Mask ./mask01sy.pnm successfully loaded.
Mask ./mask02sy.pnm successfully loaded.
Mask ./mask03sy.pnm successfully loaded.
Mask ./mask04sy.pnm successfully loaded.
Mask ./mask05sy.pnm successfully loaded.
Mask ./mask06sy.pnm successfully loaded.
Mask ./mask07sy.pnm successfully loaded.
Mask ./mask08sy.pnm successfully loaded.
Mask ./mask00cy.pnm successfully loaded.
Mask ./mask01cy.pnm successfully loaded.
Mask ./mask02cy.pnm successfully loaded.
Mask ./mask03cy.pnm successfully loaded.
Invalid SOS parameters for sequential JPEG
SINGLE:   thr:   0, side: 8 (106.73), center: 0 (5.46)
ORTE:             side: 8 (106.73), center: 0 (5.46)
Invalid SOS parameters for sequential JPEG
SINGLE:   thr:   0, side: 8 (106.95), center: 0 (4.68)
ORTE:             side: 8 (106.84), center: 0 (5.07)
Invalid SOS parameters for sequential JPEG
SINGLE:   thr:   0, side: 8 (109.62), center: 0 (5.22)
ORTE:             side: 8 (107.77), center: 0 (5.12)
Invalid SOS parameters for sequential JPEG
SINGLE:   thr:   0, side: 8 (111.94), center: 0 (5.44)
ORTE:             side: 8 (109.50), center: 0 (5.11)
Invalid SOS parameters for sequential JPEG
SINGLE:   thr:   0, side: 8 (111.39), center: 0 (5.94)
ORTE:             side: 8 (110.99), center: 0 (5.53)
Invalid SOS parameters for sequential JPEG
SINGLE:   thr:  77, side: 0 (839.02), center: 3 (305.62)
ORTE:             side: 0 (279.67), center: 3 (101.87)
Invalid SOS parameters for sequential JPEG
SINGLE:   thr:  84, side: 0 (987.94), center: 3 (359.53)
ORTE:             side: 0 (608.99), center: 3 (221.72)
Invalid SOS parameters for sequential JPEG
SINGLE:   thr:  84, side: 0 (1009.31), center: 3 (353.42)
ORTE:             side: 0 (945.42), center: 3 (339.52)
Invalid SOS parameters for sequential JPEG
SINGLE:   thr:  85, side: 0 (1040.54), center: 3 (363.46)
```

```
ORTE:               side: 0 (1012.60), center: 3 (358.80)

...

SINGLE:   thr:  85, side: 1 (977.47), center: 2 (265.17)
ORTE:               side: 1 (927.12), center: 2 (260.63)
Invalid SOS parameters for sequential JPEG
SINGLE:   thr:  84, side: 1 (919.91), center: 2 (238.68)
ORTE:               side: 1 (934.33), center: 2 (250.24)
Invalid SOS parameters for sequential JPEG
SINGLE:   thr:  85, side: 1 (934.59), center: 0 (41.13)
ORTE:               side: 1 (943.99), center: 2 (167.95)
Invalid SOS parameters for sequential JPEG
SINGLE:   thr:  86, side: 1 (973.56), center: 0 (70.70)
ORTE:               side: 1 (942.69), center: 2 (79.56)
Invalid SOS parameters for sequential JPEG
SINGLE:   thr:  88, side: 1 (1036.92), center: 0 (58.69)
ORTE:               side: 1 (981.69), center: 0 (56.84)
Invalid SOS parameters for sequential JPEG
SINGLE:   thr:  87, side: 1 (998.00), center: 0 (75.30)
ORTE:               side: 1 (1002.83), center: 0 (68.23)
Invalid SOS parameters for sequential JPEG
SINGLE:   thr:  90, side: 1 (1092.41), center: 0 (51.26)
ORTE:               side: 1 (1042.44), center: 0 (61.75)

...

SINGLE:   thr:  90, side: 1 (1078.15), center: 0 (60.99)
ORTE:               side: 1 (1054.40), center: 0 (66.76)
Invalid SOS parameters for sequential JPEG
SINGLE:   thr:  89, side: 1 (1040.66), center: 0 (67.72)
ORTE:               side: 1 (1042.51), center: 0 (68.74)
Invalid SOS parameters for sequential JPEG
orte: camera_control changed: ctrl 0
SINGLE:   thr:  88, side: 1 (1011.42), center: 0 (74.59)
ORTE:               side: 1 (1043.41), center: 0 (67.77)
waiting...
waiting...
waiting...

...
```

A.2 Experiment s redundancí

Následující výpis vznikl sloučením výpisů (resp. jejich potřebných částí) chybně klasifikovaných snímků při experimentu s redundancí. Řádky v blocích jsou uvozeny informací o hrací barvě a správné konfiguraci (**snížené o jedna!**), následují informace tištěné aplikací *rozkuk*. Bloky jsou nadepsány čísly kukuřic, které byly z masek vypuštěny.

```
1:
b42_3  thr:  64 (thrChange 0), side: 4 (46,28), center: 1 (249,68)

2:
b42_3  thr:  64 (thrChange 0), side: 5 (46,21), center: 1 (249,68)
b72_4  thr:  81 (thrChange 0), side: 7 (425,12), center: 1 (62,73)

3:
b91_1  thr:  81 (thrChange 0), side: 7 (138,03), center: 0 (124,30)
b91_3  thr:  81 (thrChange 0), side: 7 (286,82), center: 0 (117,17)
b91_4  thr:  82 (thrChange 0), side: 7 (126,15), center: 0 (165,06)

4:
y74_2  thr:  85 (thrChange 0), side: 5 (16,30), center: 3 (301,95)
y74_3  thr:  84 (thrChange 0), side: 5 (19,18), center: 3 (250,41)
y74_4  thr:  84 (thrChange 0), side: 5 (19,02), center: 3 (229,52)
y74_5  thr:  84 (thrChange 0), side: 5 (11,75), center: 3 (305,91)
y84_1  thr:  83 (thrChange 0), side: 4 (10,17), center: 3 (299,83)
y84_2  thr:  81 (thrChange 0), side: 4 (26,77), center: 3 (90,64)
y84_3  thr:  80 (thrChange 0), side: 4 (24,62), center: 3 (253,30)
y84_5  thr:  84 (thrChange 0), side: 4 (18,29), center: 3 (294,21)
y94_1  thr:  76 (thrChange 0), side: 3 (15,20), center: 3 (269,62)
y94_2  thr:  76 (thrChange 0), side: 3 (16,52), center: 3 (223,48)
y94_3  thr:  76 (thrChange 0), side: 3 (10,07), center: 3 (271,07)
y94_4  thr:  76 (thrChange 0), side: 3 (14,09), center: 3 (256,18)
y94_5  thr:  76 (thrChange 0), side: 3 (15,59), center: 3 (254,90)

6:
y74_2  thr:  85 (thrChange 0), side: 0 (16,30), center: 3 (301,95)
y74_3  thr:  84 (thrChange 0), side: 0 (19,18), center: 3 (250,41)
y74_4  thr:  84 (thrChange 0), side: 0 (19,02), center: 3 (229,52)
y74_5  thr:  84 (thrChange 0), side: 0 (11,75), center: 3 (305,91)
y84_1  thr:  83 (thrChange 0), side: 1 (10,17), center: 3 (299,83)
y84_2  thr:  81 (thrChange 0), side: 1 (26,77), center: 3 (90,64)
y84_3  thr:  80 (thrChange 0), side: 1 (24,62), center: 3 (253,30)
```

```
y84_5  thr:  84 (thrChange 0), side: 1 (18,29), center: 3 (294,21)
y94_1  thr:  76 (thrChange 0), side: 2 (15,20), center: 3 (269,62)
y94_2  thr:  76 (thrChange 0), side: 2 (16,52), center: 3 (223,48)
y94_3  thr:  76 (thrChange 0), side: 2 (10,07), center: 3 (271,07)
y94_4  thr:  76 (thrChange 0), side: 2 (14,09), center: 3 (256,18)
y94_5  thr:  76 (thrChange 0), side: 2 (15,59), center: 3 (254,90)

10:
b93_1  thr:  81 (thrChange 0), side: 8 (1110,44), center: 0 (0,00)
b93_2  thr:  81 (thrChange 0), side: 8 (1290,06), center: 0 (0,00)
b93_3  thr:  80 (thrChange 0), side: 8 (1292,80), center: 0 (0,00)
b93_4  thr:  81 (thrChange 0), side: 8 (422,39), center: 0 (0,00)
b93_5  thr:  81 (thrChange 0), side: 8 (160,15), center: 0 (0,00)
y53_1  thr:  79 (thrChange 0), side: 4 (2462,51), center: 0 (0,00)
y53_2  thr:  79 (thrChange 0), side: 4 (2158,48), center: 0 (0,00)
y53_3  thr:  79 (thrChange 0), side: 4 (1641,89), center: 0 (0,00)
y53_4  thr:  81 (thrChange 0), side: 4 (2541,33), center: 0 (0,00)
y53_5  thr:  79 (thrChange 0), side: 4 (2162,12), center: 0 (0,00)
b94_1  thr:  81 (thrChange 0), side: 8 (145,92), center: 1 (0,00)
b94_2  thr:  81 (thrChange 0), side: 8 (1141,11), center: 1 (0,00)
b94_3  thr:  81 (thrChange 0), side: 8 (511,73), center: 1 (0,00)
b94_4  thr:  77 (thrChange 0), side: 8 (391,45), center: 1 (0,00)
b94_5  thr:  81 (thrChange 0), side: 8 (969,63), center: 1 (0,00)
y74_1  thr:  84 (thrChange 0), side: 6 (895,09), center: 1 (0,00)
y74_2  thr:  85 (thrChange 0), side: 6 (1364,96), center: 1 (0,00)
y74_3  thr:  84 (thrChange 0), side: 6 (1435,04), center: 1 (0,00)
y74_4  thr:  84 (thrChange 0), side: 6 (1442,53), center: 1 (0,00)
y74_5  thr:  84 (thrChange 0), side: 6 (1335,18), center: 1 (0,00)
```

B Obsah přiloženého CD

bayes/ Zdrojové soubory bayesovského klasifikátoru pro Matlab®, masky a předzpracovaná data ze snímků v adresáři *measurements* (viz *readme.txt*).

doc/ Pravidla soutěže Eurobot a další dokumenty používané při tvorbě práce.

Figures/ Počítačové grafiky – nákresy, schémata, grafy, ...

logs/ Konzolové výpisy aplikace rozkuk pořízené především ve Švýcarsku (viz *readme.txt*).

measurements/ Kamerové snímky tzv. „bezchybové datové sady" pořízené na testovacím hřišti (viz *readme.txt*).

- **experiments**/
  - **redundancy**/ Data a výsledky měření při experimentu s redundancí (kapitola 4.6.3; viz *readme.txt*).
  - **threshold**/ Výsledky měření při experimentu s prahem (kapitola 4.6.2; viz *readme.txt*).
- **outputs**/ Výpisy aplikace *rozkuk* pro snímky bezchybové datové sady při standardních podmínkách.

photos/ Fotografie robota pořízené fotoaparátem.

snapshots/ Kamerové snímky určené k prezentačním účelům.

src/ Zdrojové soubory aplikací *rozkuk* a *maskgen* (viz *readme.txt*).

Styles/ Šablony pro generování hlavy této práce.