Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Robofotbal: Modul počítačového vidění

Plzeň 2012 Vojtěch Frič

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 26. dubna 2012

Vojtěch Frič

Poděkování

Tímto bych chtěl poděkovat panu Ing. Kamilu Ekšteinovi Ph.D. za čas a pomoc, které mi věnoval při tvorbě této práce a také za umožnění účasti na projektu robotického fotbalu na Západočeské Univerzitě v Plzni.

Abstract

The main goal of this bachelor thesis is to design and implement a module for image recognition and classification for the robotic soccer. The main aspects of the development are speed and robustness. The system has to provide precise and reliable data under varying conditions. The extracted features are position and rotation of each robot on the playground and position of the ball. Identification of features is based upon a color segmentation and a further refinement of extracted features. In order to achieve sufficient efficiency the system utilizes OpenCV library intended for image processing applications. The C++ programming language, which is essential for OpenCV, is used as well.

Obsah

1 Úvod

Projekt robotického fotbalu pokračuje na Západočeské Univerzitě již druhým rokem. Cílem celého projektu je vytvořit konkurenceschopný soutěžní tým na poli jedinečné soutěžní disciplíny. Na vývoji se podílejí studenti a pedagogové z několika kateder, kdy každá ze skupin pokrývá svou vlastní oblast práce v závislosti na zaměření. Skupina z katedry informatiky a výpočetní techniky se zaměřuje na softwarovou stránku celého projektu. Cílem naší skupiny je vytvořit „mozek“ celého projektu, systém, který bude řídit jednotlivé roboty během zápasu.

Člověk přijímá přibližně 80% vjemů pomocí zraku[1]. Oproti tomu počítače pracují s informací v binární podobě, kdy řetězec nul a jedniček může znamenat cokoliv. Výsledkem mojí práce je aplikace, která plní funkci prostředníka, kdy na jedné straně stojí fyzická scéna a na straně druhé software, který pro svou funkci vyžaduje přesný popis dané scény. Možností, jak tento popis scény získat, je více, tím nejzřejmějším je měření a ruční zadávání výsledků do systému. Tento způsob je jistě spolehlivý a přesný, ani zdaleka však neprobíhá v reálném čase.

Moje práce si klade za cíl vytvořit aplikaci, která bude na základě obrazových dat poskytovat údaje o jednotlivých robotech na hřišti a tyto údaje předávat dalším modulům řídicího systému. Data pro aplikaci jsou získávána z kamery umístěné nad hřištěm. Během vývoje a návrhu byl kladen důraz především na rychlost a přesnost zpracování, aplikace musí dále být dostatečně robustní, protože během zápasu se velmi náhle mění podmínky snímání, dále je také nutné počítat se šumem a deformací objektů způsobenou optickou soustavou.

2 Teoretická část

2.1 Robofotbal

Robotický fotbal je prestižní soutěží na poli pokročilé robotiky. Nabízí prostor mladé generaci výzkumníků z oboru robotiky, umělé inteligence, zpracování obrazu, ale také mechaniky a elektrotechniky. Oficiální turnaje jsou pořádány pod záštitou organizace FIRA – *Federation of International Robot-soccer Association*, která byla založena v roce 1997. Přesto se první oficiální turnaj konal již v roce 1995 v Korei [2]. V rámci turnaje FIRA Cup existuje několik nezávislých kategorií [3]. Jmenovitě to jsou:

- Micro-Robot Soccer Tournament (MiroSot) – roboti malých rozměrů, řízeni centrálním počítačem,
- Simulated Robot Soccer Tournament (SimuroSot) – simulované zápasy probíhající ve virtuálním prostředí,
- Humanoid Robot Soccer Tournament (HuroSot) – roboti pohybující se na dvou nohách.

2.1.1 MiroSot

Tým Západočeské Univerzity se zaměřil na kategorii MiroSot. V rámci této kategorie vyvstává jisté množství technických omezení, která budou zmíněna později. Tato kategorie si klade za cíl rozvíjet dovednosti v oblasti umělé inteligence a řízení nezávislých spolupracujících jednotek. Dále se zaměřuje na strojírenské a elektrotechnické dovednosti, kdy je nutné vytvořit robustní a efektivní konstrukci robota v rámci omezených rozměrů. Důraz je také kladen na vývoj a výzkum algoritmů počítačového vidění a zpracování obrazu, jež je hlavním tématem této práce.

2.2 Pravidla hry

Kompletní pravidla pro danou kategorii jsou velmi komplexní a není cílem této práce je rozebírat. Úplné znění je možné dohledat na oficiálním webu soutěže [4]. V následující části se proto zaměřím pouze na ty aspekty pravidel, která se přímo dotýkají počítačového vidění a zpracování obrazu. Kategorie

MiroSot se dále dělí na střední a vyšší ligu, náš tým se soustředí na střední ligu. Dále uvedená data se tak týkají střední ligy.

2.2.1 Roboti a míček

Zápas hrají dva týmy robotů v počtu 5 členů. Rozměry robota nesmí v žádném směru přesáhnout 75 mm. Jedinou výjimkou je anténa pro rádiovou komunikaci, pro kterou omezení rozměrů neplatí. Z důvodu umožnění infračerveného snímání by měly mít stěny robota světlou barvu s výjimkou oblastí nutných pro správnou funkci robota – senzory, kola. Každý robot by měl být vybaven uniformou, která plní pouze ochranou a identifikační funkci. Rozměry uniformy jsou omezeny na 80 mm v každém směru. Na vrchní straně robota je připevněn barevný identifikační štítek. Štítek musí obsahovat plochu o minimálních rozměrech 35×35 mm v barvě týmu. Barva týmu je žlutá nebo modrá a mezi jednotlivými zápasy se může měnit. Barvu týmu přiřazuje pořadatel před začátkem každého zápasu. Barevný štítek dále nesmí obsahovat barvy stejné nebo podobné jako má štítek protihráče.

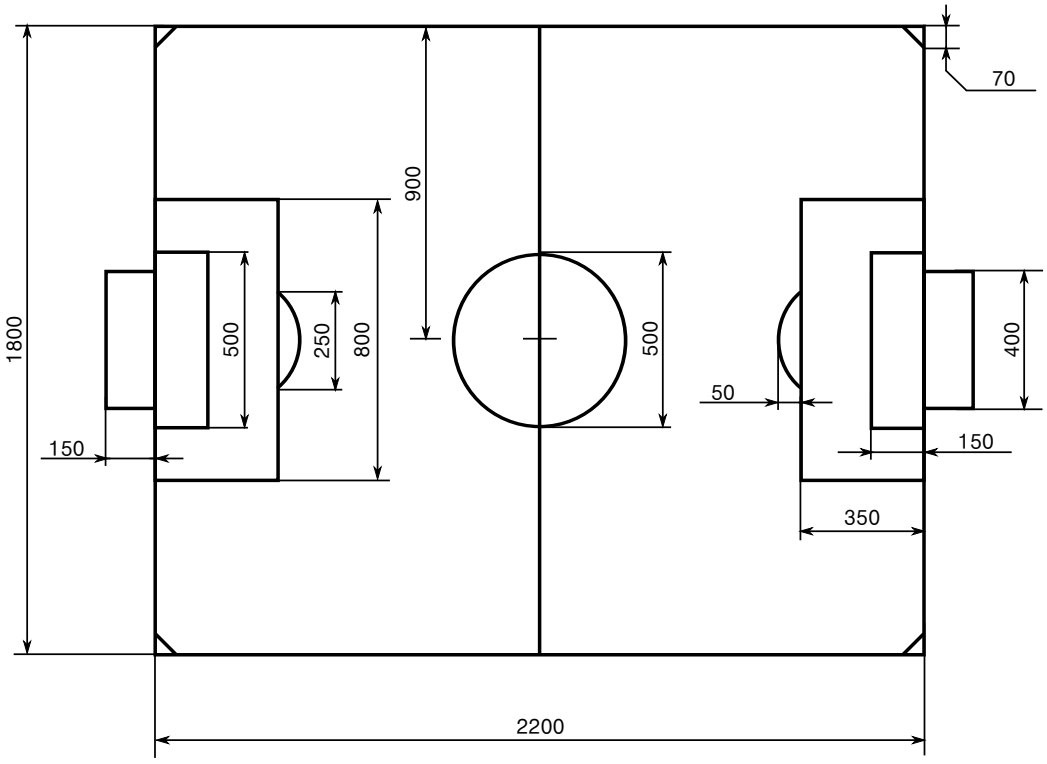
Míček je standardní golfový míček o průměru 42,7 mm oranžové barvy.

2.2.2 Hřiště

Hřiště tvoří matná černá obdélníková deska o rozměrech 1800×2200 mm. Boční stěny jsou 50 mm vysoké a 25 mm široké. Stěny mají horní stranu černou a boční strany bílé. V rozích jsou umístěny rovnoramenné trojúhelníky o straně 70 mm. Veškeré značky a čáry na hřišti jsou bílé barvy, silné 3 mm.

Kamera a osvětlení

Intenzita osvětlení musí být na celé ploše hřiště vyšší než 500 luxů. Světlo by mělo být difuzní a rovnoměrně rozptýlené po celé ploše hřiště. Je doporučeno použít neblikající zdroj světla. V opačném případě by mohlo docházet k narušení snímání při vyšších snímacích rychlostech kamery. Tým smí použít pouze jedinou kameru. Kamera musí být umístěna nad vlastní polovinou hřiště, pokud týmy chtějí umístit svoje kamery nad střed hřiště, jsou tyto umístěny vedle sebe, ve stejné vzdálenosti od středu. Kamera je umístěna ve výšce 2,5 metru nad hrací plochou.

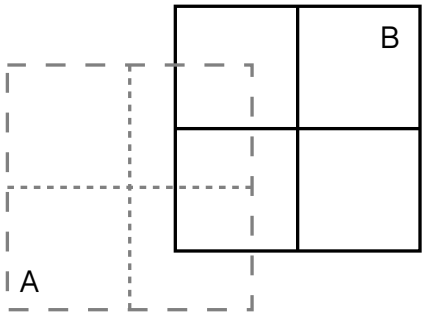


Obrázek 2.1: Schéma hřiště

2.3 Cíl práce

Výše uvedená fakta ustanovují prvotní technická omezení, která přímo ovlivňují funkci modulu pro rozpoznávání obrazu. Cílem této bakalářské práce je vytvořit systém, který dokáže na základě předaných obrazových dat zjistit pozici a natočení jednotlivých robotů a míčku na herní ploše. Dále musí být schopen jednoznačně rozlišit roboty vlastní a protihráče. Tento systém dále musí být dostatečně rychlý, aby neomezoval výpočet herní strategie a řízení robotů. Systém dále musí být robustní a dostatečně všestranný. V neposlední řadě je důležité, aby systém bylo možno jednoduše upravovat změnou parametrů. Výše uvedené požadavky plynou vesměs z technických a technologických omezení, která jsou dána pravidly. Další omezení vyvstávají až podružně na základě předpokladů ustanovených při vývoji a návrhu celého řídicího systému a také vlastních robotů. Mezi tato druhotná omezení patří především požadavek na rychlost.

Při návrhu robotů byl stanoven předpoklad, že roboti se budou pohybovat rychlostí až 2 $m \cdot s^{-1}$. Z tohoto důvodu jsme se rozhodli pro kameru snímající



Obrázek 2.2: Lokalita robota

rychlostí 50 FPS[1]. Při této frekvenci snímání je interval mezi dvěma snímky 20 ms, během kterých ujede robot maximální rychlostí 40 mm. Vzhledem k velikosti robota to znamená, že nová pozice robota bude stále v rozmezí pozice předchozí. Tento jev zobrazuje obrázek 2.2, kde A označuje pozici v čase T a B označuje pozici v čase T+20 ms.

Z celkového intervalu 20 ms mezi dvěma snímky bylo pro zpracování obrazu předběžně vyhrazeno 5 ms. Celý systém počítačového vidění tak byl navrhován s ohledem na tuto hranici.

2.4 Možnosti zpracování obrazu

Vlastnímu vývoji modulu počítačového vidění předcházel výběr vhodných softwarových prostředků. První testy probíhaly za využití prostředků .NET Framework[2], v programovacím jazyce C#. Velmi záhy se ukázalo, že C# ve své podstatě není vůbec vhodný pro implementaci, především proto, že prostředky poskytnuté pro práci s rastrovým obrazem jsou naprosto nedostačující. Pouhé zjištění barvy jednoho pixelu rastru trvá řádově desítky milisekund. V případě využití účinnějšího přístupu tento čas klesá mírně pod jednu milisekundu. Manipulace s pixely je základní operací v oblasti zpracování obrazu, .NET Framework proto není vhodným nástrojem pro daný úkol.

Pro aplikace počítačového vidění je v naprosté většině případů stěžejní rychlost programu. Velké množství aplikací tak vzniká v nízkoúrovňových programovacích jazycích, především C a C++. Tyto jazyky jsou nepřekonatelné v rychlosti, protože nabízí přímý přístup do paměti. Jejich hlavní

[1]FPS – počet snímků za sekundu

[2]`http://msdn.microsoft.com/en-us/netframework`

nevýhodou ovšem je paradoxně jejich jednoduchost. V oblasti počítačového vidění je nejdůležitějším prvkem vlastní obraz. Vzhledem k množství existujících formátů a způsobů reprezentace obrazu v počítači se pouhé samotné načtení obrazu do programu stává při využití těchto jazyků citelnou překážkou.

2.4.1 Dostupné programy

Další software využívaný především při vývoji nových algoritmů počítačového vidění je MATLAB[3] a Mathematica[4]. Tyto programy zaměřené primárně na matematické výpočty se pro náš projekt také nehodí. Hlavním problémem by bylo propojení stávajícího řídicího systému a finanční stránka věci. Využití obou programů je zpoplatněno. Dále tyto programy nejsou vhodné, protože jsou velmi pomalé.

Vzhledem k výše uvedeným předpokladům jsem se rozhodl při svojí práci využít projekt OpenCV. Jedná se o soubor knihoven určených pro počítačové vidění a zpracovnání obrazu. Projekt OpenCV je šířen pod licencí BSD [5] a má otevřený zdrojový kód. Jeho využití není zpoplatněno ani při komerčním, ani při akademickém nasazení.

2.5 OpenCV

2.5.1 Stručná historie

Projekt OpenCV vzniknul v roce 1999 [6] pod hlavičkou Intel Research. Jeho cílem bylo pokročit na poli výpočetně náročných aplikací. Zpočátku se na projektu podíleli především experti z Intel Russia a Intel Performance Library Team. Mezi jejich hlavní cíle patřilo poskytnout ucelenou infrastrukturu pro počítačové vidění, která bude nejen optimalizovaná, ale také otevřená.

První alfa verze byla přestavena v roce 2000 na konferenci IEEE Conference on Computer Vision and Pattern Recognition. První finální verze 1.0 byla vydána v roce 2006. Další verze 2.0 vyšla v říjnu roku 2009 [6]. V této verzi doznal celý systém významných změn. Mezi hlavní patří především přepsání celého systému do jazyka C++, což umožnilo snazší a bezpečnější použití. Také se zlepšila podpora vícejádrových procesorů.

[3] `http://www.mathworks.com/products/matlab/`
[4] `http://www.wolfram.com/mathematica/`

V září 2010 přibyla podpora pro výpočty na grafické kartě díky systému CUDA[5] společnosti NVIDIA.

2.5.2 Použité softwarové prostředky

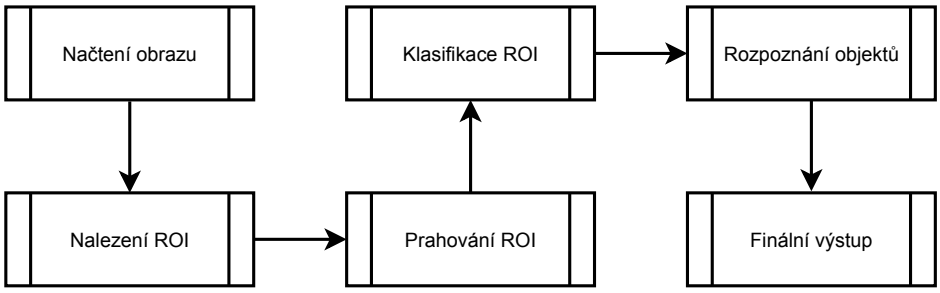
Při svojí práci jsem využíval poslední verzi OpenCV 2.3. Celý program byl napsán v jazyce C++ s využitím prostředí Microsoft Visual Studio. V době práce na bakalářské práci ještě nebyla k dispozici kamera, z toho důvodu jsem při práci jako referenční data využíval výstup simulace [7]. Díky prostředkům, které OpenCV nabízí, jsem měl velmi usnadněnou práci se správou dat, protože OpenCV obsahuje rozhraní pro načítání a ukládání velkého počtu více či méně obvyklých grafických formátů. Vývoj v C++ dále umožnil využít STL[6] pro snadnější správu dat uvnitř paměti a rychlejší zpracování v průběhu jednotlivých úseků aplikace.

[5] `http://developer.nvidia.com/what-cuda`

[6] Standard Template Library – standartní knihovna šablon

3 Vývoj

V následující kapitole popíšu průběh vývoje systému počítačového vidění. V rámci kapitoly budou postupně prezentovány jednotlivé testované postupy včetně teoretického základu jednotlivých metod. Stručný přehled jednotlivých fází zpracování obrazu ukazuje diagram 3.1.



Obrázek 3.1: Diagram zpracování obrazu

3.1 Načítání obrazu

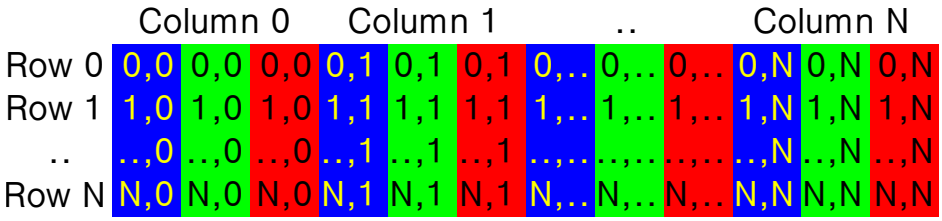
Současná verze se při načítání obrazu omezuje pouze na načítání obrazu pomocí funkcí OpenCV. V době tvorby této práce nebyla k dispozici kamera, a tudíž chyběla reálná data. V budoucnu však bude třeba tento problém vyřešit a proto nastíním několik technických detailů ohledně OpenCV.

3.1.1 Kontejner `cv::Mat`

V původní verzi využívala knihovna OpenCV rozhraní dostupné v programovacím jazyku C. To sebou neslo velký problém v podobě ruční správy paměti. S přechodem na C++ byl tento problém víceméně eliminován díky systému automatické správy paměti. Hlavním přínosem nového systému správy paměti je také úspora paměti. Veškeré operace probíhající nad daty jsou svázané s jedním blokem dat. Při předávání matice funkcím se předává pouze hlavička matice, která obsahuje základní informace o matici (rozměry, uspořádání, pointer na data).

OpenCV podporuje širokou škálu barevných systémů. RGB v několika bitových hloubkách, CIELAB, YCrCb, HSV, HSL, u všech těchto systémů, je rozdílné uspořádání dat v paměti. V kontextu mojí práce je ovšem významné

pouze načítání RGB dat. OpenCV ukládá barevné obrázky ve formátu naznačeném na obrázku (obr. 3.2). Podstatným detailem je obrácené pořadí barevných složek.



Obrázek 3.2: Uspořádání dat v paměti

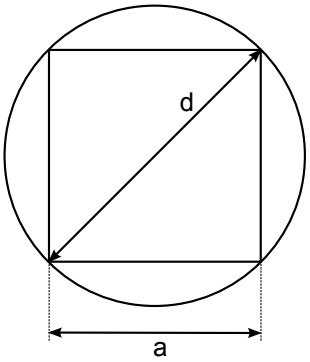
3.2 Hledání ROI

Při pohledu na hrací plochu skrz objektiv kamery je zřejmé, že převážnou většinu záběru zabírá pozadí. Pozadí jako takové ovšem není pro identifikaci jednotlivých robotů důležité. Vzhledem k tomu, že i lineární algoritmy zpracování obrazu dosahují ve své vnitřní podstatě kvadratické složitosti (N řádků, M sloupců), je velikost zpracovávaného obrazu faktorem, který má největší vliv na rychlost běhu algoritmu. Z těchto důvodů je nutné nejprve v celkovém záběru vyhledat jednotlivé oblasti obsahující data vhodná pro další zpracování. Tyto oblasti se v terminologii OpenCV označují jako ROI, z anglického *region of interest* (doslova oblast zájmu).

3.2.1 Volba velikosti kroku

Důležitým parametrem pro prvotní vyhledání ROI je velikost kroku prohledávání. Tato hodnota musí být co největší, aby se omezil počet kroků a tím zvýšila rychlost hledání, na druhou stranu musí být tato hodnota dostatečně malá, aby se nemohlo stát, že vyhledávací algoritmus mine hledaný objekt. Velikost kroku je tak přímo závislá na velikosti nejmenšího hledaného prvku, v našem případě míčku. Během prohledávání vytvářejí zkoumané body pravidelnou čtvercovou mřížku, z toho vyplývá, že velikost kroku je rovna délce strany největšího čtverce, který lze vepsat do obvodu míčku (obr. 3.3):

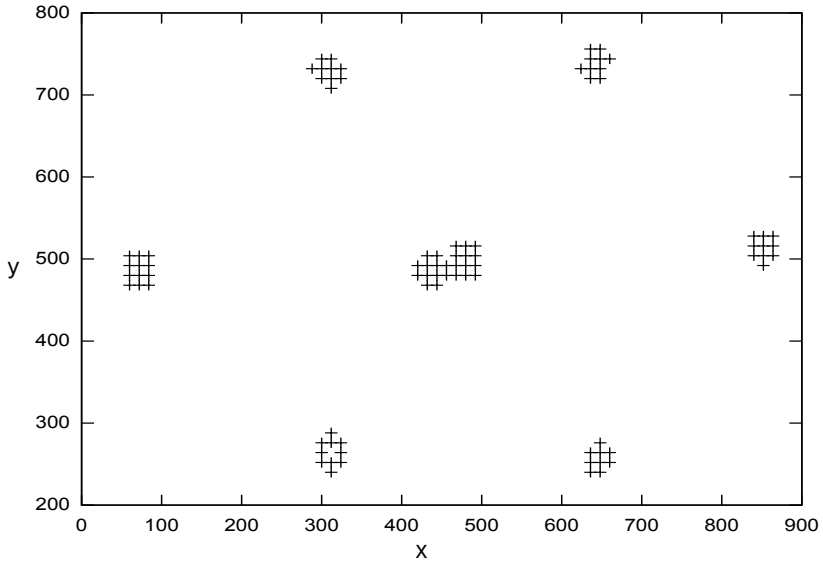
$$a = \frac{d}{\sqrt{2}} \tag{3.1}$$



Obrázek 3.3: Výpočet velikosti kroku

3.2.2 Shlukování

V každém kroku získáme souřadnice bodu, v tomto bodě poté zjistíme barvu odpovídajícího pixelu a na základě toho určíme, jestli je bod součástí pozadí nebo popředí. Body popředí ukládáme pro pozdější zpracování. Tyto body samy o sobě ovšem nenesou dostatečně užitečnou informaci, proto je nutné je dále zpracovat (obr. 3.4).



Obrázek 3.4: Data získaná při vyhledávání

Jedna z možných definic shlukování zní *„Proces slučování objektů do skupin na základě jejich podobných vlastností.“* [8]. Jedná se o hledání struktury

v množině neoznačených dat. Zmíněnými objekty jsou konkrétní body popředí a sledovanou vlastností je příslušnost k obrazu určitého robota. Vzhledem k povaze dat není možné určit pouze na základě znalosti pozice a barvy pixelu, ke kterému robotu nalezené body náleží. Z toho důvodu jsem při shlukování jako společnou vlastnost využil vzájemnou vzdálenost bodů. Konkrétní postup popisuje část 5.1.2. Výstupem mnou aplikovaného algoritmu je množina obdélníkových částí obrazu, které obsahují ty části, kde se nachází roboti a míček.

3.3 Prahování

V předchozí části jsem nastínil, jakým způsobem lze najít jednotlivé ROI. Prvním krokem zpracování je převod z barevného obrazu na obraz binární. Binární obraz obsahuje pouze dvě barvy, obvykle je bílá barva reprezentována jedničkou a barva černá nulou. Tímto převodem ztratí obraz velkou část informace, kterou poskytuje barva, ovšem přínosem je výrazné urychlení práce dalších algoritmů. Tyto algoritmy totiž pouze rozlišují dvě binární hodnoty v porovnání s 256 odstíny šedi, případně 16777216 barevnými odstíny[1].

Při vývoji jsem nejprve využíval funkce OpenCV – `inRange()`. Tato funkce nastaví hodnotu výstupního pixelu podle následujícího vzorce:

$$P = \begin{cases} 255 & l_B \leq S_B \leq u_B \wedge l_G \leq S_G \leq u_G \wedge l_R \leq S_R \leq u_R \\ 0 & jindy, \end{cases} \tag{3.2}$$

kde $S_i, i = B, G, R$ je zdrojový pixel, P je výstupní pixel, $l_i, u_i, i = B, G, R$ označují dolní a horní limit. Během vývoje se ukázalo, že tato funkce neposkytuje dostatečně kvalitní výstup. Hlavním problémem bylo správné nastavení parametrů funkce. Z toho důvodu jsem ve finální verzi využil vlastní prahovací funkci, která je popsána v části 5.2.1.

3.3.1 Eroze

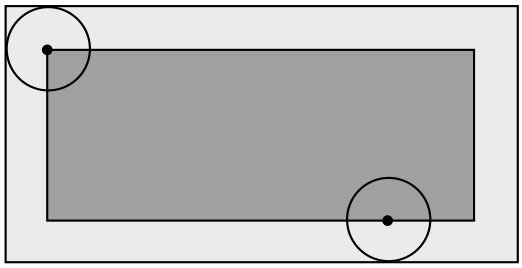
Eroze je matematická morfologická operace, která slouží k filtrování obrazu. Morfologické operace jsou operace, které slouží k získávání geometrických informací z binárních a šedotónových obrazů. Maska použití při zpracování obrazu se nazývá strukturní element (SE). Výsledek morfologické operace zřejmě závisí na tvaru velikosti použitého SE. Matematická morfologie je postavena na teorii množin. Pokud je $w = (x, y)$ (souřadnice bodu obrazu)

[1] Uvažujeme-li 8 bitů na složku.

prvkem množiny A, píšeme $w \in A$, v opačném případě $w \notin A$. Množina souřadnic B, které splňují určitou podmínku, se značí $B = \{w : podmínka\}$. Mezi základní množinové operace patří sjednocení (3.3a), průnik (3.3b), doplněk (3.3c) a rozdíl (3.3d) množin. Dalšími operacemi jsou reflexe (3.3e) a translace (3.3f).

$$C = A \cup B \tag{3.3a}$$
$$C = A \cap B \tag{3.3b}$$
$$A^{'} = \{w : \ w \notin A\} \tag{3.3c}$$
$$C = B - A \tag{3.3d}$$
$$\hat{A} = \{w : \ w = -a, pro\ a \in A\} \tag{3.3e}$$
$$(A)_z = \{c : \ c = a + z, pro\ a \in A\} \tag{3.3f}$$

Eroze je operace, která slouží ke smrštění či zúžení obrazu. Erozi je možné popsat tak, že v každém bodě obrazu přiložíme SE a zjišťujeme, zda SE protíná pozadí nebo celý leží uvnitř obrazu. Tento efekt je vidět na obrázku 3.5, kde tmavší obdélník je výsledkem eroze světlejšího obdélníku zobrazenou kruhovou maskou. Eroze je matematicky definována takto:



Obrázek 3.5: Eroze

$$A \ominus B = \{z : \ (B)_z \cap A^{'} \neq \emptyset\} \tag{3.4}$$

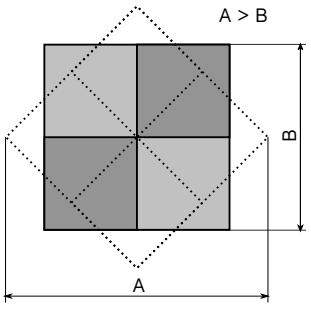
Erozi jsem přidal jako poslední krok při prahování, z důvodu odstranění šumu z binárního obrazu. Protože eroze způsobuje zmenšení objektu, je nutné s tímto zmenšením počítat při dalším zpracování. Při erozi diskem o průměru 3 pixely, tvoří zmenšení v průměru jeden pixel v každém směru. Nepříjemnou vlastností eroze je zvětšení děr v obrazu. Tento efekt je velmi problematický, především při využití distanční transformace (část 4.2.3). Ovšem vzhledem k použitým postupům (viz dále) není nutné tento jev odstranit.

3.4 Klasifikace

Vlastnímu rozpoznávání předchází ještě klasifikace jednotlivých ROI. Tento krok je významný pro další urychlení celého procesu, protože pro rozpoznávání jednotlivých robotů je využit rychlejší postup než pro rozpoznávání skupin. V prvotní verzi programu probíhala klasifikace na základě rozměrů jednotlivých ROI. Tento postup je dostačující, byť nedosahuje stoprocentní úspěšnosti. V malém počtu případů docházelo k chybné identifikaci, kdy bylo ROI s jedním robotem klasifikováno jako množina.

Hlavním důvodem, proč docházelo k chybným klasifikacím, byla skutečnost, že vzhledem ke způsobu, jakým jsou ROI získávána vzniká kolem každého objektu oblast, která je pozadím. Rozměry této oblasti nejsou pevné a nelze je určit předem. Tento problém jsem vyřešil ořezáváním jednotlivých ROI na minimální možnou velikost. Tyto minimální rozměry jsou poté rozšířeny, aby kolem každého objektu vznikl minimální rámeček, který má vliv na další zpracování. Během testů jsem zjistil, že ani takto upravené kritérium není stále dostačující.

Jelikož při konverzi obrázku z plně barevného na binární je nutné zpracovat každý pixel bez výjimky, zavedl jsem do algoritmu výpočet plochy. Za každý pixel popředí (bílá barva) je navýšen čítač, který na konci zpracování obsahuje velikost plochy popředí. Tento parametr je poté využitý pro rozlišení jednotlivých případů. Díky zavedení kontroly plochy je dále klasifikace invariantní vůči rotaci objektu. Tento efekt je vidět na obrázku 3.6.



Obrázek 3.6: Vliv rotace na rozměry ROI

4 Rozpoznávání

V této části popíšu jednotlivé postupy, které jsem vyzkoušel při řešení vlastního rozpoznávání. Celá část je rozdělená do dvou samostatných oddílů. První oddíl se týká rozpoznání samostatných objektů, což je ve své podstatě velmi jednoduchý problém. Druhá, obsáhlejší část, je zaměřená na rozpoznávání skupin robotů.

4.1 Samostatný robot

Proces rozpoznávání jednoho robota v ROI je velmi jednoduchý. V prvním kroku je prohledáno celé ROI a jsou nalezeny hranice objektu. Příslušné body na hranicích jsou uloženy a je nad nimi vypočten obdélník s minimální plochou obsahující všechny dané body. Pro hledání minimálního obdélníka využívám funkci OpenCV. Výsledkem je obdélník, který má těžiště shodné s těžištěm obrazu robota. Úhel natočení obdélníka odpovídá natočení robota.

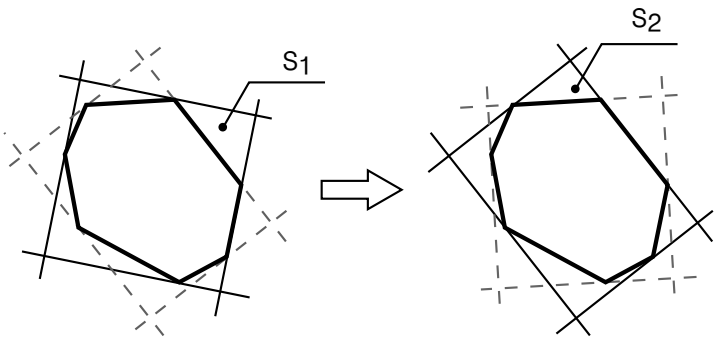
4.1.1 Rotating Calipers

Funkce pro hledání minimálního obdélníka je klíčovou pro tuto práci, proto zde nastíním, jak funguje. Funkce implementuje algoritmus *Rotating Calipers*[1] [9]. Freeman a Shapira [10] dokázali, že minimální obdélník musí mít minimálně jednu hranu shodnou s hranou polygonu. Pro nalezení minimálního obdélníka jsou použity čtyři přímky, které jsou navzájem kolmé. Tyto přímky jsou poté přiloženy na každou hranu polygonu a po otestování všech hran se vybere ta pozice s nejmenší plochou. Tento efekt je lépe patrný z obrázku 4.1. Složitost celého algoritmu je lineární. V obrázku označuje plná čára stav v iteraci `I` a přerušovaná stav v iteraci `I+1`; S_1 a S_2 označují příslušné obsahy rotovaných obdélníků.

4.2 Množina robotů

V této části uvedu vyzkoušené postupy pro detekci jednotlivých robotů, pokud se v jednom ROI nenachází pouze jeden robot. U jednotlivých postupů uvedu jejich princip a dále zhodnocení, nakolik jsou vhodné pro účely systému počítačového vidění.

[1] česky rotující „šuplera“



Obrázek 4.1: Rotating Calipers

4.2.1 Hashování obrazu

Jedno z prvních uvažovaných řešení bylo využití obrazového hashování. Jedná se o postup, který úspěšně využívají mnohé webové služby k vyhledávání na základě obrazových dat. Celý postup je postaven na porovnání číselných otisků obrázků. Pro výpočet takového otisku existuje spousta algoritmů. Nejjednodušším je hashování podle průměru (Average Hashing) [11]. Zpracovaný obrázek je nejprve zmenšen na velikost 8×8 pixelů. Tento krok redukuje detaily v obraze, ovšem celková struktura obrazu se výrazně nemění. V dalším kroku je zmenšený obraz převeden na šedotónový obraz. Opět se jedná o krok, který slouží k redukci zpracovávaných dat a neovlivní přenášenou informaci o struktuře obrazu. Šedotónový obraz je dále převeden na obraz binární, jako prahová hodnota se použije průměrná hodnota celého obrazu. Hodnoty vyšší než práh představují 1, ostatní hodnoty 0. Na základě těchto binárních hodnot je sestaven výsledný otisk jako 64-bitové číslo. Samotné porovnávání je realizováno jako výpočet Hammingovy vzdálenosti dvou otisků. Menší vzdálenost znamená vyšší pravděpodobnost shody.

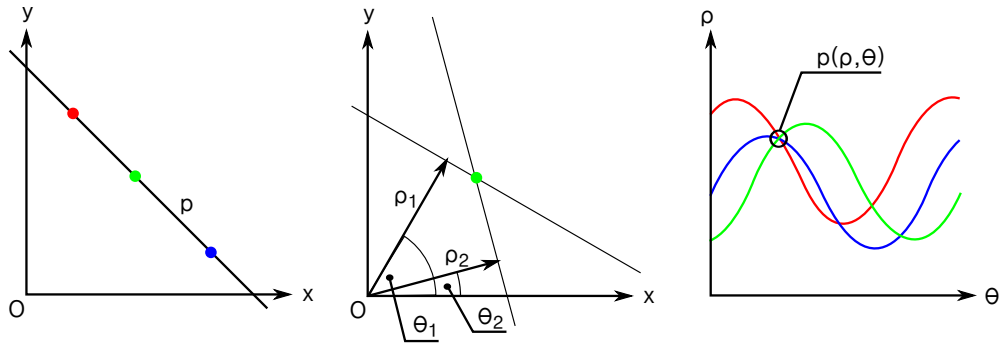
Tato metoda je nesporně velmi rychlá. Ovšem je také velmi nepřesná, už ze své podstaty nerozlišuje jemné detaily, které jsou ovšem pro moji práci stěžejní. Jedná se především o to, že informace o rotaci se během škálování obrazu ztratí. Rozdíly v řádech jednotek stupňů nijak neovlivní výsledný otisk. Další mnohem významnější problém je potřeba předpočítané databáze referenčních vzorů. Ovšem vytvoření této databáze je vzhledem k povaze řešeného problému nemyslitelné. Tato databáze by musela obsahovat veškeré možné herní situace, které mohou nastat.

4.2.2 Houghova transformace

Houghova transformace umožňuje vyhledávání parametrických křivek v binárním obraze, který je prahovaným výstupem hranového operátoru [12]. Hlavní výhodou je, že Houghova transformace vykazuje dobré výsledky i pro obrazy silně zašuměné či poškozené. Pro získání dobrých výsledků nemusí být hrany v binárním obraze spojité, což je další výhoda. Houghova transformace pro funkci $A(x, y)$ je definována následovně:

$$H(\theta, \rho) \quad = \quad \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} A(x, y)\delta(\rho - x \cdot cos\theta - y \cdot sin\theta)\mathrm{d}x\mathrm{d}y \qquad (4.1)$$

Tuto rovnici lze názorně vysvětlit tak, že každým bodem obrazu A proložíme přímku (obr. 4.2b), kde θ je úhel, který svírá přímka s kladnou osou x a ρ je kolmá vzdálenost přímky od počátku. Každou z těchto přímek je možné reprezentovat jako bod v tzv. *Houghově prostoru* (obr. 4.2c). V praxi se využívá pro zaznamenávání akumulátor tvořený dvourozměrným polem, kde jedna souřadnice odpovídá úhlu a druhá vzdálenosti. Pro každou proloženou přímku se zvýší čítač na odpovídající pozici v akumulátoru. Pokud dva body v obraze leží na stejné přímce (obr. 4.2a), tak vytvoří stejnou odezvu (dvojici ρ a θ), která se promítne v akumulátoru. Po vypočtení parametrů všech přímek pro každý bod jsou vyhledána lokální maxima v akumulátoru. Pokud nalezené lokální maximum převyšuje předem stanovený práh, je zpětně vypočtena přímka, která těmto bodům odpovídá.



(a) Body v obraze (b) Generované přímky (c) Houghův prostor

Obrázek 4.2: Houghova transformace

Houghova transformace poskytuje velmi dobré výsledky. Má ovšem velmi významné nedostatky, které způsobily, že jsem tento postup zavrhnul. Hlavním nedostatkem je rychlost. Houghova transformace je z principu velmi

pomalá, je nutné pro každý bod vygenerovat sadu přímek a pro každou z přímek vypočítat její vzdálenost od počátku. Pokud budeme požadovat rozlišení 1 stupeň, dostaneme se na hodnotu 180 výpočtů pro každý bod. Dále je nutné najít v obraze hrany před vlastní transformací. Tato operace také není triviální. Další problém je volba vhodné hodnoty prahu pro hledání parametrů v akumulátoru. V neposlední řadě je překážkou samotný výstup Houghovy transformace. I za předpokladu ideálních podmínek je výstupem množina přímek, které odpovídají pozicí a úhlem úsečkám v obraze, ovšem nejsou nijak omezeny. Pro účinné rozpoznání objektů ve scéně je tak nutné najít správné průsečíky těchto přímek, které odpovídají rohům objektů a na základě těchto průsečíků zpětně rekonstruovat tvar hledaného objektu.

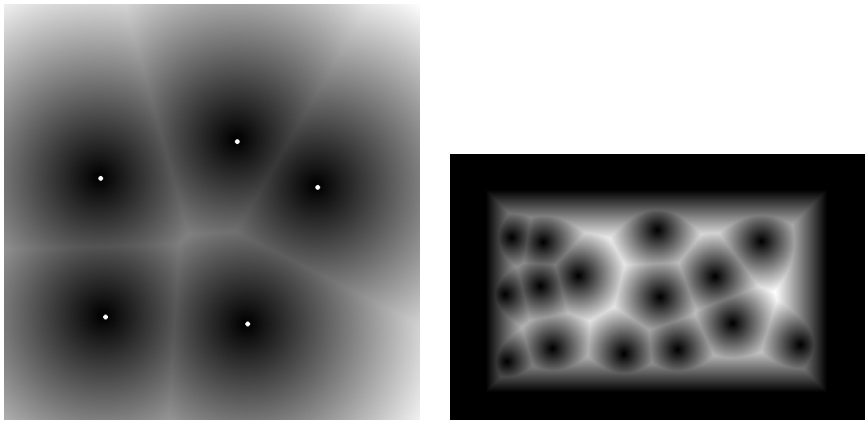
4.2.3 Transformace vzdálenosti

Při práci na rozpoznávacím systému jsem zjistil, že řešení podobného problému je velmi časté v aplikacích pro mikrobiologii [13] a agronomii [14, 15]. V těchto případech je také cílem rozpoznat jednotlivé části shluků částic, buněk, případně semen. Pro řešení tohoto zadání se velmi úspěšně využívá transformace vzdálenosti (distance transform).

Princip transformace vzdálenosti

Máme danou množinou diskrétních bodů $\mathbf{P}_i$, $i = 0, 1, \ldots, n$ (nazývané význačné body), ve dvourozměrném prostoru $\mathcal{R}^2$. Nejbližší soused bodu $\mathbf{A}$, označovaný jako *NN(*$\mathbf{A}$*)*, je bod $\mathbf{P}_k$ takový, že pro něj platí $d(\mathbf{A}, \mathbf{P}_k) \leq d(\mathbf{A}, \mathbf{P}_i)$ pro všechna $i = 0, 1, \ldots, n$, kde $d(*, *)$ je Euklidovská vzdálenost dvou bodů. Potom můžeme pro danou množinu bodů v $\mathcal{R}^2$ definovat funkční hodnotu v každém bodě $\mathbf{A}$ jako Euklidovskou vzdálenost mezi bodem $\mathbf{A}$ a jeho nejbližším sousedem *NN(*$\mathbf{A}$*)* [12]. Výsledkem je transformace vzdálenosti množiny význačných bodů. Na obrázku 4.3a jsou význačné body pixely s hodnotou 0. V článku [13] je představen postup pro detekci částic v mikroskopických vzorcích. Postup spočívá v nalezení hran v obraze, kde jsou body hran využity jako význačné body pro transformaci vzdálenosti. Vzhledem k implementaci algoritmu hledání distanční transformace v OpenCV je možné tento krok vynechat. Ve výsledné mapě vzdálenosti jsou poté vyhledána lokální maxima, která odpovídají středům jednotlivých objektů. Po nalezení středů jsou do těchto středů umístěny šablony hledaných částic a na základě dalšího zpracování jsou nalezeny pozice a rotace konkrétních objektů.

V článku [16] je prezentován postup, kdy je transformace vzdálenosti vyu-



(a) množina význačných bodů (b) obdélník se šumem

Obrázek 4.3: Distanční transformace

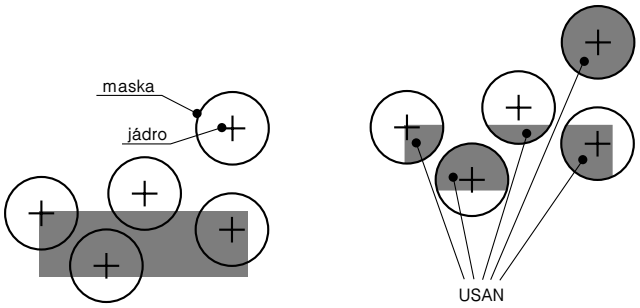
žita pro odhadování úhlu natočení skenovaných dokumentů. Uvedený postup je založen na výpočtu průměrného gradientu mapy transformace vzdálenosti. Využívá toho faktu, že mezery mezi řádky textu vytváří dobře rozpoznatelné oblasti, které sklonem odpovídají natočení dokumentu.

Oba výše uvedené postupy jsem implementoval a během testů jsem oba zamítnul. Hlavním nedostatkem v obou případech je rychlost výpočtu transformace vzdálenosti. Dalším problémem je citlivost transformace vzdálenosti na šum, tento efekt je vidět na obrázku 4.3b, kdy i drobné nepřesnosti znehodnotí celý výpočet. V případě postupu z článku [13] se dále projevila jako zásadní problém malá přesnost. V případě, že byly zpracovávány obrázky malých rozměrů (< 60 px), nebylo možné dostatečně přesně určit lokální maximum a tím i výchozí bod pro další zpracování. V případě druhém se překážkou stala nutnost vypočítat hodnotu gradientu pro každý pixel obrazu. Ukázalo se, že běžné metody aproximace gradientu dosahují příliš velké chyby, než aby se získaná data dala využít pro další zpracování. Tato chyba je způsobena diskrétní povahou aproximace gradientu a kroky, které by vedly k lepším výsledkům by dále zvýšily čas nutný ke zpracování obrázku, což není žádoucí.

4.2.4 Detektor SUSAN

Jako další postup jsem zvolil metodu zaměřenou na vyhledávání rohů v obrázku. Pro hledání rohů existuje větší množství algoritmů [17]. Já jsem se rozhodl implementovat detektor SUSAN [18], který na rozdíl od jiných

detektorů (Shi-Thomas, Harris) není součástí OpenCV. Detektor SUSAN je založen na jednodušším principu než dříve uvedené detektory. Díky tomu je snadnější jeho implementace. Detektor SUSAN využívá při hledání význač-



Obrázek 4.4: Maska a USAN

ných bodů kruhovou masku. Pro každý bod vstupního obrázku je vypočtena velikost plochy, která má podobnou intenzitu jako jádro masky (nucleus). Za jádro masky je považován střed masky, z toho důvodu je v praxi průměr masky liché číslo. Dopad tohoto předpokladu je zobrazen na obrázku 4.4. Oblast s podobnou intenzitou je nazývána USAN – Univalue Segment Assimilating Nucleus[2]. Velikost plochy USAN tedy přímo popisuje strukturu obrázku v daném místě. V praxi to znamená, že menší hodnota znamená výraznější prvek v obraze, rohy jsou navíc mnohem výraznější než hrany. Díky tomu, že SUSAN nevyužívá žádné derivace obrazu, je velmi odolný vůči šumu.

Původní řešení

Algoritmus vykonává následující kroky pro každý pixel vstupního obrazu. Nejprve je do obrazu umístěna kruhová maska a je vypočtena odezva dané oblasti na základě rovnice (4.3), kde $\mathbf{r_0}$ je střed masky $\mathbf{M}$, $\mathbf{r}$ jsou jednotlivé pixely masky.

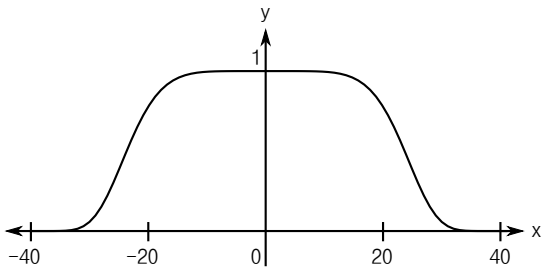
$$c(\mathbf{r}, \mathbf{r_0}) = e^{-\left(\frac{I(\mathbf{r})-I(\mathbf{r_0})}{t}\right)^6} \tag{4.2}$$

$$n\left(\mathbf{r_0}\right) = \sum_{\mathbf{r}\in\mathbf{M}} c(\mathbf{r}, \mathbf{r_0}) \tag{4.3}$$

Funkce $c(\mathbf{r}, \mathbf{r_0})$ slouží k určení podobnosti intenzity pixelů[3], její průběh je na obrázku 4.5. Osa x označuje vzájemný rozdíl intenzity pixelů, osa y je odezva detektoru. Parametry $\mathbf{r_0}$ a $\mathbf{r}$ odpovídají pozici jádra a jednotlivým bodům

[2]segment podobný jádru

[3]Detektor je určen pro šedotónové obrázky.



Obrázek 4.5: Funkce pro porovnání pixelů

masky. Funkce $I(\mathbf{r})$ označuje intenzitu pixelu v daném místě. Parametr t je vstupní parametr algoritmu a určuje, zda je měřený pixel dostatečně podobný jádru. Výstupem algoritmu je obraz R definovaný následujícím předpisem:

$$R(\mathbf{r_0}) = \begin{cases} g - n(\mathbf{r_0}) & n(\mathbf{r_0}) < g \\ 0 & jindy, \end{cases} \tag{4.4}$$

kde g je dalším parametrem algoritmu. Jedná se o geometrický práh, který určuje, zda daný bod je potenciální roh, jeho hodnota je rovna polovině hodnoty n_{max}, což je maximální hodnota, které může parametr n nabývat. V obraze R jsou poté hledána lokální minima, která odpovídají pozicím potenciálních rohů. Detailnější popis je možno najít v původním článku [18].

Zavedené úpravy

Po prostudování materiálů k algoritmu SUSAN jsem se rozhodl algoritmus zjednodušit. Zjednodušení spočívá v tom, že na rozdíl od autorů původního algoritmu ve své práci zpracovávám pouze binární obraz. Také je předem známo, že pixely pozadí mají hodnotu 0 a pixely popředí 1 (viz 3.3). Rovnice 4.3 se tak zjednoduší následujícím způsobem:

$$n\left(\mathbf{r_0}\right) = \sum_{\mathbf{r} \in \mathbf{M}} I(\mathbf{r}), \tag{4.5}$$

zbytek výpočtu už probíhá totožně s původní verzí.

V obou případech se ukázalo, že detekce rohů s využitím detektoru SUSAN není dostatečně rychlá. Kompenzace zmenšením použité masky vedla k výraznému zhoršení přesnosti. Dalším problémem bylo, že se mi nepodařilo dosáhnout přesnosti, která by se blížila přesnosti publikované v článku [18]. To bylo způsobeno především nevhodnou implementací následného zpracování výstupního obrazu R. Pokud by se mi podařilo implementovat funkční

detektor SUSAN, stále by bylo nutné nalezené rohy dále zpracovat: Nejprve zamítnout ty rohy, které nemají pravý úhel a poté ze získaných bodů rekonstruovat finální tvar robota. Jelikož ovšem detektory poskytují pouze informaci o pozici, ale ne o spojitosti, nebyla by tato rekonstrukce triviálním úkolem. Úspěšná rekonstrukce jednotlivých robotů by stále nebyla dostačující, protože by stále bylo nutné lokalizovat míček, pokud by se v obraze nacházel.

4.3 Hledání obrysů

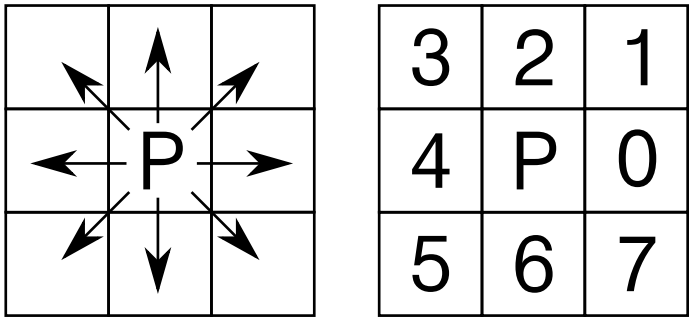
Během řešení daného úkolu jsem zjistil, že stěžejním úkolem bude nalezení významných bodů v obraze, konkrétně rohů jednotlivých robotů. Již jsem popsal některé metody, které vedly k nalezení rohů, ovšem vzhledem k časovému limitu nebylo vhodné je využít. Dalším argumentem pro hledání jiného postupu bylo, že uvedené metody poskytují pouze informaci o pozici, ale ne informaci o propojení jednotlivých bodů.

4.3.1 Freemanův řetězový kód

Řetězové kódy jsou určeny k účinné bezztrátové kompresi plošných křivek[4]. Princip spočívá v tom, že pro danou křivku není nutné ukládat souřadnice všech bodů, ale ukládá se pouze informace o změně průběhu křivky v daném kroku. V případě reálných[5] křivek tedy dochází k jisté ztrátě přesnosti z důvodu vzorkování. Touto křivkou ovšem může být i obrys oblasti v dvourozměrném rastru. Pokud budeme uvažovat rastr v celočíselných souřadnicích, je problém přesnosti bezpředmětný. Jediným omezením je jednoznačné rozlišení jednotlivých komponent obrazu a pozadí. V článku [19] popisuje Freeman postup pro kódování rastrových dat. V rámci jeho práce je křivka spojitá, pokud každý bod křivky sousedí s dalším bodem. Sousednost je definována v rámci tzv. 8-okolí bodu (obr. 4.6a). Pro reprezentaci křivky se nejprve určí počáteční bod a zaznamenají se jeho souřadnice. U každého dalšího bodu se zaznamenává pouze směr, jakým se do tohoto bodu lze dostat z bodu předchozího (obr. 4.6b). S pomocí této reprezentace lze každý bod (kromě prvního) zakódovat pomocí pouhých tří bitů.

[4]Existují aplikace pro vyšší dimenze.

[5]souřadnice z oboru reálných čísel

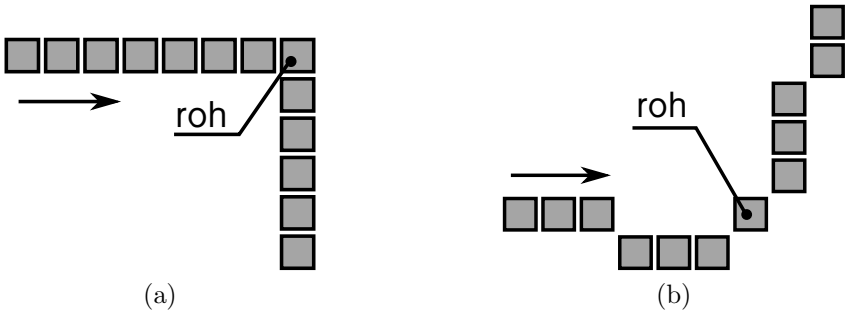


(a) 8-okolí (b) Kódování

Obrázek 4.6: Freemanův kód

4.3.2 Zpracování řetězového kódu

V řetězovém kódu je nutné nalézt význačné body. Řešení tohoto problému není triviální. Hlavním problémem při hledání význačných bodů (rohů) je vlastní definování rohu. Obecně je roh definován jako náhlá změna směru kontury. Ovšem lokalizace těchto náhlých změn je komplikována rastrovou povahou dat. Tento efekt je vidět na obrázku 4.7a, kde je obrys zakódován následující sekvencí *000000066666*. Jak je vidět, roh je jasně definován změnou kódu z *0* na *6*. To je příklad ideálního rohu, v praxi se ovšem setkáváme především s rohy, které mají mnohem obecnější tvar (obr. 4.7b). Zde je obrys zakódován sekvencí *00700112212*. Jak je vidět, roh není jasně definován jako v předchozím případě. Články [20] a [21] prezentují řešení, které umožňuje



(a) (b)

Obrázek 4.7: Roh

vyhladit celý kód, díky čemuž rohy vyniknou. Během testů se ukázalo, že tyto metody nejsou vhodné, protože neposkytují žádnou kontrolu nad parametry nalezených rohů. Častým jevem se ukázala být chybná identifikace v případě

lokálních změn kontury. Dále byly jako rohy označeny některé části míčku, který může vykazovat lokální změny směru, ovšem v globálním měřítku nemá rohy žádné, protože první derivace obrysu je spojitá. Další komplikací je, že OpenCV implementuje funkce pro hledání obrysů v obraze, ale výstupem je posloupnost bodů, nikoliv řetězový kód, proto je nutné nejprve z těchto bodů vypočítat řetězový kód. Při výpočtu kódu se postupně prochází posloupnost bodů, pro každé dva po sobě jdoucí body se určí jejich kód podle schématu na obrázku 4.6b.

4.3.3 Identifikace rohů v posloupnosti bodů

V nalezené posloupnosti bodů, které tvoří obrysy objektu, se vyskytuje větší množství rohů. Pro nalezení robotů jsou podstatné pouze rohy, které mají pravý úhel, tj. body, které odpovídají rohům jednotlivých robotů.

Při hledání pravoúhlých rohů jsem z důvodu rychlosti zamítl metodu založenou na goniometrických výpočtech. Rozhodl jsem se využít vlastnosti pravoúhlých trojúhelníků, vyplývající z Pythagorovy věty

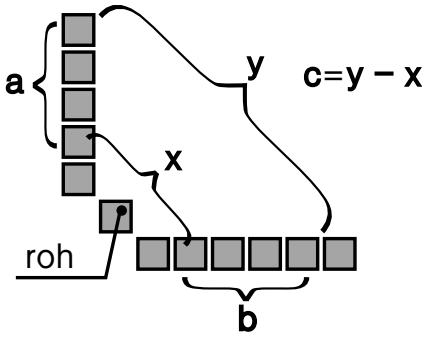
$$|\mathbf{a}|^2 + |\mathbf{b}|^2 - |\mathbf{c}|^2 = 0, \tag{4.6}$$

kde $|\mathbf{a}|$ a $|\mathbf{b}|$ jsou délky odvěsen a $|\mathbf{c}|$ je délka přepony pravoúhlého trojúhelníka. Díky znalosti pozic jednotlivých bodů je možné využít vektorového počtu. Jednotlivé vektory vypočteme podle rovnice

$$\mathbf{a} = \mathbf{P_n} - \mathbf{P_{n+\delta}}, \qquad \mathbf{b} = \mathbf{P_n} - \mathbf{P_{n-\delta}}, \qquad \mathbf{c} = \mathbf{P_{n+\delta}} - \mathbf{P_{n-\delta}}, \tag{4.7}$$

kde $\mathbf{P_n}$ je pozice bodu, pro který ověřujeme, zda se jedná o pravoúhlý roh a $\mathbf{P_{n\pm\delta}}$ jsou body vzdálené na křivce o δ indexů od bodu $\mathbf{P_n}$. Při testování se ukázalo, že ne všechny pravoúhlé rohy jsou přesně vymezené (obr. 4.8). Z toho důvodu jsem se rozhodl výpočet upravit tak, že započítám body, které jsou posunuté vůči bodu, pro který počítám pravý úhel. Z důvodu posunutí bylo nutné upravit výpočet, aby stále odpovídaly poměry druhých mocnin. Ostatní výpočty zůstaly beze změny.

Ukázalo se, že tato metoda není vůbec spolehlivá, především nebylo možné lokalizovat roh přesně. Jako druhý problém se ukázal velký počet nesprávně potvrzených rohů.



Obrázek 4.8: Neostrý roh

4.3.4 Aproximace obrysu

Jak jsem naznačil v předchozích odstavcích, metody zpracovávající celý řetězový kód, nejsou dostatečně přesné. Proto jsem se rozhodl využít metodu pro aproximaci obrysů, kdy vstupem metody je polygon a výstupem je opět polygon, který je složen z menšího počtu vrcholů, ale vzdálenost původního a nového polygonu je menší než zvolená přesnost. Jinými slovy, získáme nový polygon, který je téměř totožný s původním, ale je složen pouze z minimálního počtu vrcholů. Tato metoda má tu vlastnost, že téměř lineární úseky nahradí úsečkou, díky tomu je snadné lokalizovat význačné body na celé křivce.

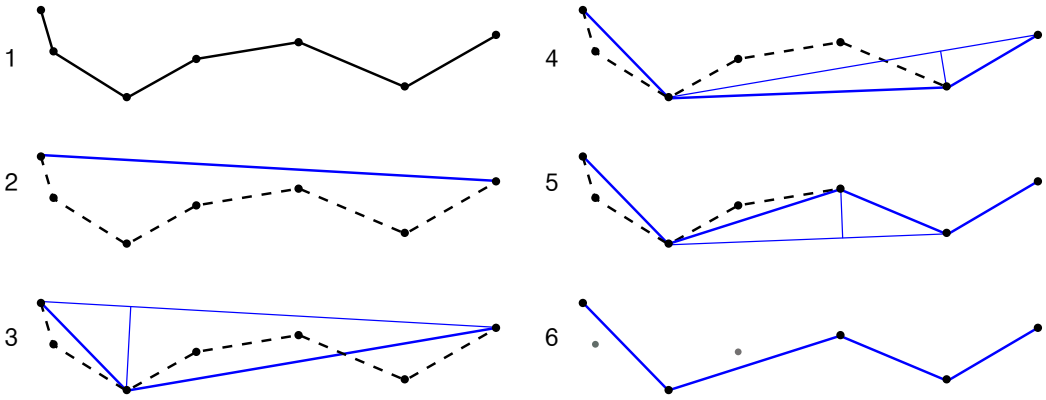
Algoritmus Ramer–Douglas–Peucker

OpenCV pro aproximaci obrysů využívá rekurzivní algoritmus RDP[6]. Algoritmus nejprve označí dva body křivky, které jsou od sebe nejdále, za konečné. Tyto body jsou označeny jako počáteční a konečný bod nové křivky. V dalším kroku je nalezen bod, který je od této nově vzniklé úsečky nejvzdálenější, vzdálenost se měří na kolmici. Pokud je vzdálenost nalezeného bodu od přímky menší než zvolené ε tak všechny body, které leží mezi koncovými body úsečky, je možné vynechat, včetně právě nalezeného. Nově vzniklá úsečka se tak od původní neliší více než o dané ε.

Pokud je nalezený bod vzdálenější od původní úsečky dále než je zvolené ε, je tento bod označen za konečný a původní úsečka je nahrazena dvěma úsečkami, které spojují krajní body úsečky původní a nově nalezený bod. Algoritmus poté rekurzivně otestuje obě nově vzniklé úsečky. Rekurze po-

[6]Ramer–Douglas–Peucker

kračuje, dokud se některý z úseků liší od původní přímky více než o ε.



Obrázek 4.9: RDP algoritmus

4.3.5 Rekonstrukce objektů

Po provedení aproximace vyhledám na křivce body, ve kterých je úhel rohu větší než 180°. Tyto body jsou význačné tím, že označují místa, ve kterých se jednotlivé objekty dotýkají. Z jednotlivých částí obrysů objektu je potom možné s pomocí dříve popsaných postupů rekonstruovat jednotlivé objekty.

5 Realizace

V následující části práce představím celý systém počítačového vidění z hlediska implementace a použitých programátorských postupů. První část je věnována prvotní verzi systému, který byl vytvořen na základě dat, která se reálným datům pouze blíží. Druhá část popisuje vývoj prozatím finální verze, která vznikala za využití dat získaných ze simulace reálného zápasu [7, 22, 23, 24].

5.1 První verze

První verze programu dokázala pouze lokalizovat osamocené roboty. Vzhledem k povaze vstupních dat ovšem dobře posloužila k vyladění postupů pro práci s barvou a identifikaci pozadí.

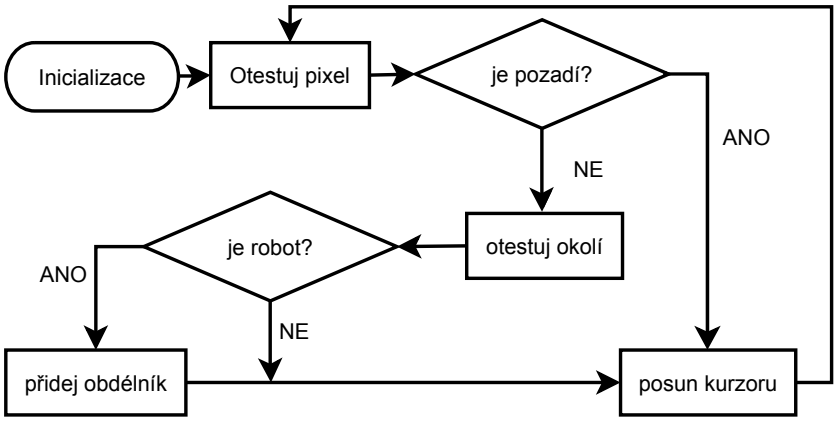
5.1.1 Data

Jako pracovní data pro první verzi systému posloužily fotografie modelů robotů pořízené pomocí fotoaparátu. Modely robotů jsou vyrobeny z pevného kartonu, který je z jedné strany označen pomocí barevných štítků. Rozměry robotů odpovídají rozměrům uvedeným v pravidlech, tj. 75×75 mm. Barevné štítky tvoří šachovnici, barvy také odpovídají pravidlům. Namísto hřiště posloužila školní tabule, která svými rozměry přibližně odpovídá rozměrům hřiště.

Získaná data odpovídají reálným datům jen velmi zhruba. Především u herní plochy se dá předpokládat mnohem rovnoměrnější rozložení barvy, kterého nebylo možné u školní tabule dosáhnout z důvodu poškození. Povrch školní tabule není rovnoměrný, obsahuje jisté množství povrchových narušení (rýhy, škrábance, odloupaná barva), která narušují jednolitost plochy. Barva tabule neodpovídá černé matné barvě, která je popsaná v pravidlech. Z důvodu špatných světelných podmínek byly snímky pořízeny s bleskem, což vedlo k nerovnoměrnému osvětlení celé plochy, proto střed obrazu vykazuje mnohem vyšší hodnoty jasu, než oblasti při krajích. Jelikož jsou roboti reprezentování pouze pomocí štítků, neprojeví se na výsledných snímcích zkreslení způsobené výškou robota.

5.1.2 Hledání ROI

Postup hledání ROI je zobrazen na diagramu 5.1. Prvním krokem je otestování barvy. Pro tento účel jsem navrhnul vlastní metodu, která využívá vlastností barevného systému RGB. Do budoucna uvažuji o přepsání metody, což umožní využít vlastností jiných barevných systémů, které jsou vhodnější pro porovnávání jednotlivých barev.



Obrázek 5.1: Hledání ROI

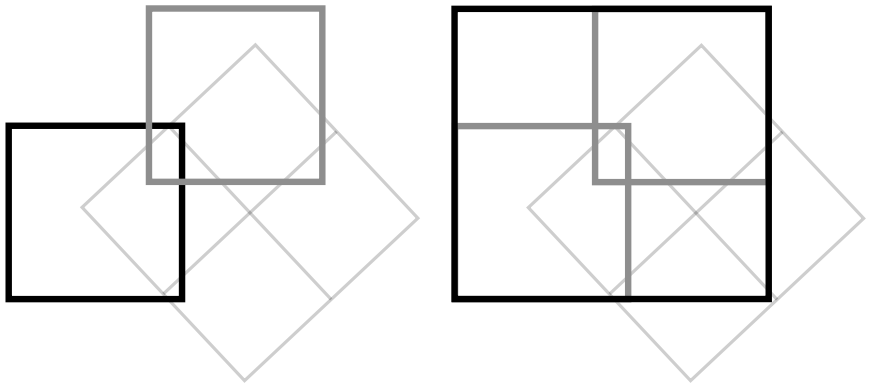
Funkce `dist_from_gray` slouží k určení vzdálenosti konkrétní barvy od odstínu šedé. V této funkci jsem využil reprezentace barevného systému RGB pomocí barevné krychle. Při této realizaci leží všechny odstíny šedé, počínaje černou barvou (0,0,0) a konče bílou barvou (255,255,255), na tělesové úhlopříčce této krychle. Odstíny šedi mají všechny tři barevné složky totožné. Podobnost barvy se šedou poté určuji jako největší absolutní hodnota rozdílů dvojic jednotlivých složek (5.1).

$$dist = max\{|R - G|, |B - G|, |R - B|\} \tag{5.1}$$

Výsledná hodnota je porovnána s prahem, který byl zvolen na základě experimentů a závisí na kvalitě zpracovávaných snímků.

Pokud je testovaný bod označen jako bod popředí, je proveden test okolí. Tento test je nutný k zamezení chybných identifikací. V okolí bodu o daných rozměrech v každém směru je s menší hodnotou kroku proveden test barvy pro odpovídající pixely. Za každý bod popředí je navýšen čítač. Když hodnota čítače přesáhne stanovenou mez, je prohledávání ukončeno a oblast je prohlášena za součást ROI. Tato oblast je v praxi většinou pouze

zlomkem velikosti výsledného ROI. Proto jsou tyto jednotlivé části ukládány do spojového seznamu a při nalezení nové části je proveden test, zda tato nová část nekoliduje s částí již nalezenou. Pokud tomu tak je, jsou tyto dvě části nahrazeny novou částí, která svými rozměry obsáhne části již nalezené (obr. 5.2). Tento postup je nezbytný pro správnou lokalizaci ROI, nese sebou ovšem drobný problém, kdy dva samostatní roboti jsou zařazeni do stejného ROI, pokud jsou dostatečně blízko sebe. Díky využití spojového seznamu



(a) situace před sloučením (b) po sloučení

Obrázek 5.2: Části ROI

je spojování a rušení oblastí velmi urychleno v porovnání s prvotní implementací, která využívala pole. Hlavním důvodem pro nevyužití pole byla komplikovaná správa dat, především plýtvání pamětí, kdy pole muselo mít dostatečnou rezervu, protože nebyl dopředu znám počet ukládaných prvků. V případě využití spojového seznamu tento problém odpadá, protože se jedná o dynamickou datovou strukturu.

5.1.3 Rozpoznání robota

Vlastní rozpoznání samostatných robotů probíhá podle postupu popsaného dříve (část 4.1). Hledání hranic objektu v binárním obraze probíhá postupným prohledáváním každého n-tého řádku, kde n se navyšuje v každém kroku o danou hodnotu, která je parametrem funkce. Každý zpracovávaný řádek ROI je kontrolován pixel po pixelu, zleva doprava. Při nalezení prvního bílého pixelu, který značí objekt, jsou uloženy jeho souřadnice. Kurzor se poté přesune na konec řádku a prohledávání probíhá v opačném směru, tj. zprava doleva. Tento postup slouží k urychlení vyhledávání, v rámci ROI je větší počet pixelů popředí (bílých), než pozadí (černých). Po nalezení dvou

bodů se kurzor přesune na další řádek obrazu.

Výsledná množina bodů je poté předána do funkce `minAreaRect`, která je součástí OpenCV a rekonstruuje příslušný obdélník na základě předaných bodů (část 4.1.1).

5.2 Finální verze

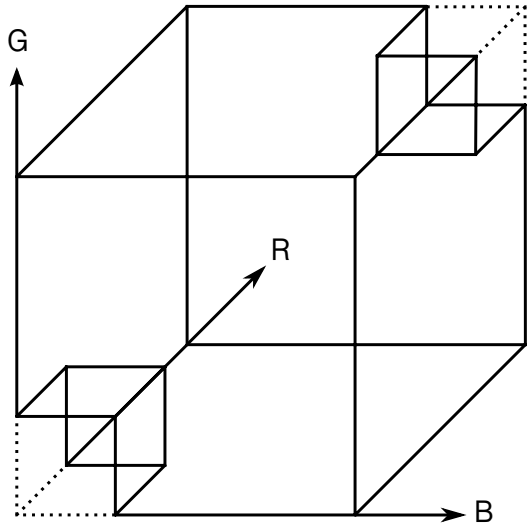
Ve verzi, která je v rámci této práce finální, jsou postupy pro lokalizaci ROI a samostatných robotů převzaty z verze předchozí. Jedinou změnou byly drobné úpravy s cílem sjednocení celého systému.

Při práci na konečné verzi jsem již využíval data získaná ze simulace zápasu. Simulaci vytvořili ostatní členové týmu v rámci svých bakalářských prací [7, 22, 23, 24]. Data ze simulace odpovídají předpokládaným datům z kamery. V době psaní této práce bohužel není kamera k dispozici, proto se jedná pouze o odhad. Předem je však jisté, že finální data se budou výrazně lišit v barevnosti. To bude způsobeno tím, že kamera má pouze 15 bitovou barevnou hloubku. Dále se dají předpokládat větší výkyvy v osvětlení, což bude způsobeno pořizováním snímků s bleskem, což je během turnajů běžné. Zatím neznámým aspektem je zašuměnost obrazu, ta výrazně ovlivní výsledky poskytované systémem.

5.2.1 Prahování

Prvním krokem po nalezení všech ROI je prahování. Při prahování (funkce `my_range_crop`) používám metodu se dvěma prahy, kdy se filtrují barvy, které leží v daném intervalu, nikoliv pouze barvy, které mají nižší hodnotu než daný práh. Díky tomuto přístupu mám mnohem vyšší kontrolu nad výsledným obrazem. Vlastní rozhodování, zda pixel odpovídá popředí nebo pozadí probíhá ve dvou krocích. Nejprve ověřím, že všechny hodnoty složek barvy leží v rozmezí daného intervalu. Pokud pixel neprojde tímto testem, je prohlášen za popředí. Pokud pixel testem projde, je proveden druhý test, za pomoci funkce `dist_from_gray`, která byla zmíněna již dříve. Opět se ověřuje, zda je pixel dostatečně daleko od příčky RGB krychle. Oba testy jsou v tomto pořadí z důvodu optimalizace; pokud pixel leží v daném intervalu, můžeme ho bezpečně označit jako popředí, protože jeho barva není dostatečně podobná bílé ani černé. V rámci tohoto intervalu leží i oblast odstínů šedi, tato výjimka byla zavedena z toho důvodu, že roboti mohou mít barevné štítky do jisté míry blízké odstínům šedi. Barvy, které jsou považovány za

popředí, jsou znázorněny na obrázku 5.3.



Obrázek 5.3: Oblast barev popředí

Po provedení prahování je aplikována eroze (část 3.3.1), ta slouží k odstranění případného šumu. V tomto kroku se zmenší plocha objektu, proto je při dalším zpracování nutné s touto změnou počítat.

V poslední fázi je celý obrázek prohledán ještě jednou. V tomto posledním běhu se počítá plocha objektu, která slouží ke klasifikaci jednotlivých ROI (část 3.4). Během výpočtu plochy objektu také dochází k oříznutí ROI na minimální rozměry. Pro každý bílý pixel popředí se porovnají souřadnice s dočasným minimem a maximem, pro obě souřadnice a případně se nahradí prozatím zjištěné hodnoty. Výpočet plochy a ořezávání probíhaly v prvotní verzi v rámci jednoho průchodu, ovšem při zavedení eroze to již nebylo možné. Teoreticky by bylo možné i erozi vypočítat v rámci jednoho průchodu, ale vzhledem ke složitosti výsledného kódu by toto řešení nevedlo k lepším výsledkům.

5.2.2 Rozpoznání skupiny robotů

Klasifikace ROI (část 3.4) a rozpoznání samostatného robota (část 5.1.3) již byly popsány, proto se budu dále věnovat implementaci rozpoznávání skupiny robotů.

Při zpracování ROI, které obsahují skupinu robotů, je prvním krokem na-

lezení obrysů v obraze. Implementace vyhledávacího algoritmu je v OpenCV natolik sofistikovaná, že nalezne všechny kontury v obraze během jednoho volání funkce. Získané kontury jsou poté zpracovávány jedna po druhé. Prvním krokem je opět výpočet plochy, tentokrát se ovšem počítá plocha uzavřená daným obrysem, díky tomu je možné dále roztřídit jednotlivé kontury, podle toho co reprezentují – samostatný robot, míček nebo shluk.

V případě, že se jedná o míček nebo samostatného robota, je nalezen minimální obdélník obsahující daný obrys a získaná data jsou připravena na výstup. Funkce pro hledání minimálního obdélníku je využitá i při určování středu míčku. Ačkoliv se to může zdát nelogické, pro účely aplikace je stěžejní znalost středu míčku. Střed výsledného obdélníka odpovídá středu míčku.

Prvním krokem při zpracování shluku je aproximace křivky jednodušším polygonem (část 4.3.4). Zjednodušená sekce je pak ve vybraných rozích rozdělena na jednotlivé segmenty. Rohy jsou vybírány na základě toho, jaký úhel svírají. K výpočetu orientovaného úhlu v daném rohu používám následující rovnice:

$$\begin{aligned} \mathbf{a} &= \mathbf{P_{n+1}} - \mathbf{P_n} \\ \mathbf{b} &= \mathbf{P_{n-1}} - \mathbf{P_n} \\ \alpha &= \operatorname{atan2}(\mathbf{a}_x \cdot \mathbf{b}_y - \mathbf{b}_x \cdot \mathbf{a}_y, \mathbf{a}_x^2 + \mathbf{a}_y^2) \bmod 2\pi, \end{aligned} \tag{5.2}$$

kde $\mathbf{P_n}$ je bod polygonu, ve kterém je měřen úhel, $\mathbf{P_{n\pm1}}$ jsou sousední body rohu. Výraz $\mathbf{a}_x \cdot \mathbf{b}_y - \mathbf{b}_x \cdot \mathbf{a}_y$ je velikost vektorového součinu vektorů a výraz $\mathbf{a}_x^2 + \mathbf{a}_y^2$ je skalární součin vektorů. Funkce *atan2* je definována následovně:

$$\operatorname{atan2}(y, x) = \begin{cases} \arctan\left(\frac{y}{x}\right) & x > 0 \\ \arctan\left(\frac{y}{x}\right) + \pi & y \geq 0, x < 0 \\ \arctan\left(\frac{y}{x}\right) - \pi & y < 0, x < 0 \\ +\frac{\pi}{2} & y > 0, x = 0 \\ -\frac{\pi}{2} & y < 0, x = 0 \\ \text{nedefinováno} & y = 0, x = 0. \end{cases} \tag{5.3}$$

V případě, že je úhel α větší než π, je možné křivku v tomto bodě rozdělit [14, 25]. Vzhledem k povaze problému je velikost úhlu dostatečným ukazatelem pro určení významných rohů. Aby se zamezilo chybám, je dále zavedena podmínka, že dva body, které následují na křivce, musí mít vzájemnou vzdálenost v ploše větší než daný práh. Protože je aproximovaná křivka uzavřená, ovšem uložená lineárně, je nutné po dokončení rozdělování sloučit první a poslední segment, díky tomu je zaručená spojitost výsledných dat.

Nad každým segmentem je opět vypočten minimální obdélník. Pro obdélníky, které mají menší rozměry, než předem stanovený práh je započítán pouze střed. Tyto segmenty náleží obrysu míčku. Po zpracování všech segmentů je vypočten průměr ze všech středů obdélníků, které odpovídají míčku. Tento průměr je potom bod, který přibližně odpovídá středu míčku. Tato metoda není dokonalá, protože středy vypočtených obdélníků neodpovídají přesně těžišti segmentů, ale pro potřeby rozpoznávání je dostatečně přesná.

5.3 Dosažené výsledky

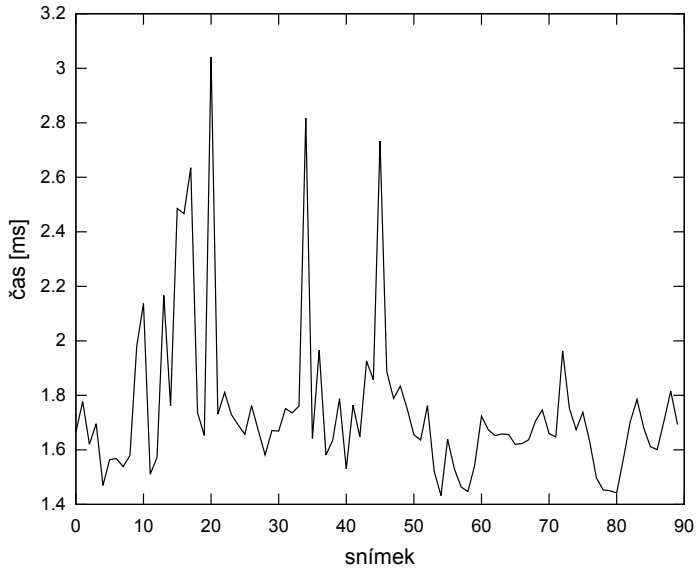
Při testování jsem využíval data získaná ze simulace. Z celkového počtu přibližně 800 snímků jsem vybral cílovou sadu 90 snímků, které dostatečně zmapovaly velkou část možných herních situací. Důraz byl kladen především na snímky obsahující shluky, protože lokalizace samostatných robotů je dobře zvládnutá.

Program při testování zpracovával snímky o rozlišení 948x990 pixelů o barevné hloubce 24 bitů. Všechny testy probíhaly na notebooku s dvoujádrovým procesorem Intel Pentium T4400 s frekvencí každého jádra 2.2 GHz, 4 GB RAM, pod operačním systémem Microsoft Windows 7. Na každém snímku jsou 4 roboti v každém týmu.

Základní rozlišení systémového časovače Windows 7 je 15.6 ms [26] s možností zpřesnění až na 1 ms. Tato rozlišení však nejsou pro měření výkonu mé aplikace dostatečně přesná. Z toho důvodu jsem zvolil funkce dostupné v OpenCV, které jsou založeny na měření počtu taktů procesoru a frekvence. Údaje o době běhu jednotlivých operací nejsou absolutně přesné, při měření takto krátkých časových úseků dochází k jistému zkreslení. Výsledky jednotlivých měření jsou uvedeny v tabulce 5.1.

proces	čas [ms]	počet prvků
lokalizace ROI	0,51	90
prahování	0,07	778
shluk robotů	0,10	284
jeden robot	0,03	494
celý snímek	1,74	90

Tabulka 5.1: Výsledky měření



Obrázek 5.4: Naměřené časy zpracování jednoho snímku

Jak je vidět z grafu 5.4, i v nejhorším případě je čas nutný pro zpracování snímku zhruba 3 ms. Dále přibližně v 90 % případů je doba zpracování kratší než 2 ms.

Při lokalizaci dosahuje program maximální přesnosti. V žádném z testovacích případů nebyl objekt minut a pouze v jednom případě byla část obrazu identifikována jako míček, který se ovšem na snímku nenalézal. Program se dále ukázal velmi robustní vůči deformacím tvarů, výstup simulace počítá s optickou deformací způsobenou optickou soustavou kamery, z toho důvodu nemají roboti tvar čtverce a jednotlivé rohy nesvírají úhel 90°. Navzdory těmto deformacím systém identifikuje středy a natočení jednotlivých robotů přesně.

6 Závěr

V rámci celého řídicího systému robotického fotbalu je modul pro počítačové vidění naprosto nezbytný. Ostatní moduly mohou pracovat samostatně, ovšem kamera a aplikace pro zpracování obrazu, kterou jsem vyvinul, představuje jediné vstupní rozhraní mezi systémem a reálným světem. Pro dosažení dokonalých výsledků je nezbytná spolupráce všech komponent celého systému, ovšem sebelepší program nemůže poskytovat správné výsledky, pokud nezíská správná vstupní data.

V duchu této myšlenky byl vyvíjen celý můj systém, proto byl mimo rychlosti kladen velký důraz i na robustnost celého systému. Systém v současné podobě splňuje veškeré požadavky na něj kladené. Doba potřebná k vyhledání všech robotů a míčku je v rámci předem stanoveného limitu a stále ponechává jistou rezervu. Při sledování výkonnosti aplikace je nutné přihlížet k výkonnosti stroje, na kterém probíhaly testy. Nasazení do praxe počítá s využitím výkonnějšího počítače, což dále urychlí vyhledávání. Přesnost vyhledávání momentálně dosahuje téměř 100 %, je však třeba přihlédnout k faktu, že systém dosud nebyl testován v reálných podmínkách. Před vlastním uvedením do praxe, bude třeba celý systém vyladit na reálné podmínky. Vzhledem k postupům, které jsem využil při vývoji aplikace, bude ladění obnášet především úpravy parametrů. Veškeré ladění by se mělo obejít bez dalších větších zásahů do kódu.

Dalším klíčovým krokem při vývoji systému bude propojení kamery a mé aplikace. To dosud nebylo možné, protože kamera v době tvorby této práce nebyla k dispozici. Je pravděpodobné, že propojení s kamerou s sebou přinese nutnost změn, především v oblasti práce s barvami. Dalším důležitým krokem bude návrh a realizace propojení mého modulu se zbytkem řídicího systému. Tento krok je také závislý na práci s kamerou, protože momentálně není známo, jakým způsobem bude kamera řízena.

Přehled zkratek

- **FIRA** – *Federation of International Robot-soccer Association* – organizace pořádající mezinárodní turnaje v robotickém fotbale.
- **MiroSot** – *Micro-Robot Soccer Tournament* – kategorie malých robotů s centrálním řízením.
- **FPS** – *frames per second*, počet snímků za sekundu.
- **RGB** – barevný systém, složky jsou červená, zelená a modrá.
- **CIELAB** – barevný systém definovaný mezinárodní komisí pro osvětlování.
- **YCrCb** – barevný systém, složky jsou jas a červená a modrá chrominanční složka.
- **HSV** – barevný systém, složky jsou odstín, sytost a hodnota.
- **HSL** – barevný systém, složky jsou odstín, sytost a světelnost.
- **ROI** – *region of interest*, oblast zájmu, část obrazu, která obsahuje data významná pro další zpracování.
- **SE** – *structure element*, maska využívaná při morfologických operacích.
- **RDP** – *Ramer-Douglas-Peucker*, iniciály autorů algoritmu pro aproximaci polygonů.

Literatura

[1] McArdle, G.: *Instructional design for action learning*. New York: American Management Association, 2010, ISBN 0-8144-1566-0, str. 65.

[2] FIRA: Greetings. [Online; citace 19. dubna 2012].
Dostupné z: `http://www.fira.net/?mid=Greetings`

[3] FIRA: FIRA Cup Overview. [Online; citace 19. dubna 2012].
Dostupné z: `http://www.fira.net/?mid=firaoverview`

[4] FIRA: Mirosot. [Online; citace 19. dubna 2012].
Dostupné z: `http://www.fira.net/?mid=mirosot`

[5] Open Source Initiative OSI – The BSD License. Říjen 2010, [Online; citace 19. dubna 2012].
Dostupné z: `http://opensource.org/licenses/bsd-license.php`

[6] Wikipedia: OpenCV – Wikipedia, The Free Encyclopedia. 2012, [Online; citace 19. dubna 2012].
Dostupné z: `http://en.wikipedia.org/w/index.php?title=OpenCV`

[7] Eckstein, R.: *Vizualizační modul pro robotický fotbal*. Bakalářská práce, Západočeská Univerzita v Plzni, Fakulta aplikovaných věd, 2011.

[8] Matteucci, M.: A Tutorial on Clustering Algorithms. [Online; citace 19. dubna 2012].
Dostupné z: `http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/`

[9] Toussaint, G.: Solving geometric problems with the rotating calipers. In *Proceedings of IEEE MELECON*, 1983.

[10] Freeman, H.; Shapira, R.: Determining the minimum-area encasing rectangle for an arbitrary closed curve. *Commun. ACM*, ročník 18, č. 7, Červenec 1975: s. 409–413, ISSN 0001-0782, doi:10.1145/360881.360919.
Dostupné z: `http://doi.acm.org/10.1145/360881.360919`

[11] Krawetz, N.: Looks Like It. Květen 2011, [Online; citace 19. dubna 2012].
Dostupné z: http://www.hackerfactor.com/blog/index.php?/archives/432-Looks-Like-It.html

[12] Huiyu Zhou, J. Z., Jiahua Wu: *Digital Image Processing – Part II*. BookBoon, 2010.
Dostupné z: http://www.e-booksdirectory.com/details.php?ebook=4284

[13] Yu, Z.; Bajaj, C.: Detecting circular and rectangular particles based on geometric feature detection in electron micrographs. *Journal of Structural Biology*, ročník 145, č. 1–2, 2004: s. 168 – 180, ISSN 1047-8477, doi:10.1016/j.jsb.2003.10.027.
Dostupné z: http://www.cs.utexas.edu/~bajaj/papers/2004/journal/sdarticle02.pdf

[14] Kiratiratanapruk, K.; Sinthupinyo, W.: Segmentation algoritm for touching round grain image. In *Electronics and Information Engineering (ICEIE), 2010 International Conference*, ročník 1, Srpen 2010, s. V1–263 –V1–266, doi:10.1109/ICEIE.2010.5559878.

[15] Yao, Q.; Zhou, Y.; Wang, J.: An automatic segmentation algorithm for touching rice grains images. In *Audio Language and Image Processing (ICALIP), 2010 International Conference*, Listopad 2010, s. 802 –805, doi:10.1109/ICALIP.2010.5685114.

[16] Bar-Yosef, I.; Hagbi, N.; Kedem, K.; aj.: Fast and Accurate Skew Estimation Based on Distance Transform. In *Document Analysis Systems, 2008. DAS '08. The Eighth IAPR International Workshop*, Září 2008, s. 402 –407, doi:10.1109/DAS.2008.65.

[17] Wikipedia: Corner detection – Wikipedia, The Free Encyclopedia. 2012, [Online; citace 19. dubna 2012].
Dostupné z: http://en.wikipedia.org/w/index.php?title=Corner_detection

[18] Smith, S. M.; Brady, J. M.: SUSAN–A New Approach to Low Level Image Processing. *International Journal of Computer Vision*, ročník 23, 1997: s. 45–78, ISSN 0920-5691.
Dostupné z: http://dx.doi.org/10.1023/A:1007963824710

[19] Freeman, H.: On the Encoding of Arbitrary Geometric Configurations. *Electronic Computers, IRE Transactions*, ročník EC-10, č. 2, Červen 1961: s. 260 –268, ISSN 0367-9950, doi:10.1109/TEC.1961.5219197.

[20] Nain, N.; Laxmi, V.; Bhadviya, B.; aj.: Corner Detection Using Difference Chain Code as Curvature. In *Signal-Image Technologies and Internet-Based System, 2007. SITIS '07. Third International IEEE Conference*, Prosinec 2007, s. 821 –825, doi:10.1109/SITIS.2007.118.

[21] Nain, N.; Laxmi, V.; Jain, A. K.; aj.: Morphological Edge Detection And Corner Detection Algorithm Using Chain-Encoding. 2006.

[22] Altman, P.: *Jádro řídicího systému a virtualizační modul pro robotický fotbal*. Bakalářská práce, Západočeská Univerzita v Plzni, Fakulta aplikovaných věd, 2011.

[23] Lepič, J.: *Multiagentní systém pro plánování herní strategie robotického fotbalu*. Bakalářská práce, Západočeská Univerzita v Plzni, Fakulta aplikovaných věd, 2011.

[24] Vališ, J.: *Modul elementární inteligence robotického fotbalu*. Bakalářská práce, Západočeská Univerzita v Plzni, Fakulta aplikovaných věd, 2011.

[25] Kong, H.; Gurcan, M.; Belkacem-Boussaid, K.: Splitting touching-cell clusters on histopathological images. In *Biomedical Imaging: From Nano to Macro, 2011 IEEE International Symposium*, Duben 2011, ISSN 1945-7928, s. 208–211, doi:10.1109/ISBI.2011.5872389.

[26] Corporation, M.: Timers, Timer Resolution, and Development of Efficient Code. Červenec 2010, [Online; citace 19. dubna 2012].
Dostupné z: `http://msdn.microsoft.com/en-us/windows/hardware/gg463266`

A Uživatelská příručka

Před vlastním použitím aplikace je nutné nainstalovat knihovny OpenCV. Typů instalace je několik v závislosti na zvolené funkčnosti a vývojovém prostředí. V následující části popíšu instalaci kompilovaných knihoven pro systém Windows 7 a Microsoft Visual Studio 2010. Celý postup je převzat z oficiální příručky, která obsahuje podrobnější návod pro instalaci dalších konfigurací.

Poznámka: Dále uvedené údaje jsou relevantní k datu tvorby práce – duben 2012. Vzhledem k vývojovému cyklu OpenCV je nutné do budoucna počítat s průběžnými aktualizacemi. Nová verze knihovny vychází v půlročních intervalech.

A.1 Instalace OpenCV

1. Ve webovém prohlížeči otevřete adresu

   `http://sourceforge.net/projects/opencvlibrary/files/opencv-win/`

2. V nabídce vybete složku s aktuální verzí (2.3.1)

3. Stáhněte soubor `OpenCV-2.3.1-win-superpack.exe`

4. Stažený soubor extrahujte do požadovaného umístění

5. V adresáři `xxyyzz\opencv\doc\`, kde `xxyyzz` je umístění extrahovaných souborů, se nachází soubor `opencv_tutorials.pdf`, který obsahuje podrobný návod k instalaci a nastavení celé knihovny.

A.2 Kompilace a spuštění

K překladu je nutné mít nainstalované Microsoft Visual Studio 2010. Pro načtení projektu stačí otevřít soubor `robovision.sln`. Jelikož systém v aktuální verzi pracuje se soubory, je nutné před kompilací nastavit správně jména souborů a adresářů. Program očekává jména souborů ve tvaru `nnnnn#C.eee`, kde `nnnnn` je jméno každého souboru, `#C` je pořadové číslo souboru a `eee`

je přípona souboru. Dále je nutné nastavit počet zpracovávaných souborů. Veškeré tyto parametry jsou označeny ve zdrojovém kódu v metodě `main`.

Překlad je možné provést výběrem volby *Build Solution* v menu *Build*, případně stiskem klávesy F6. Překlad je podmíněn správným nastavením referencí na OpenCV, předpokládá se existence systémové proměnné `%OPENCV_DIR%`, která zastupuje cestu k jednotlivým souborům OpenCV. Více informací k nastavení poskytuje oficiální příručka.

Program je možné spustit z menu *Debug* příkazem *Start Without Debugging*, případně stistkem Ctrl+F5. Po skončení běhu programu jsou výsledné zpracované soubory ve výstupním adresáři.

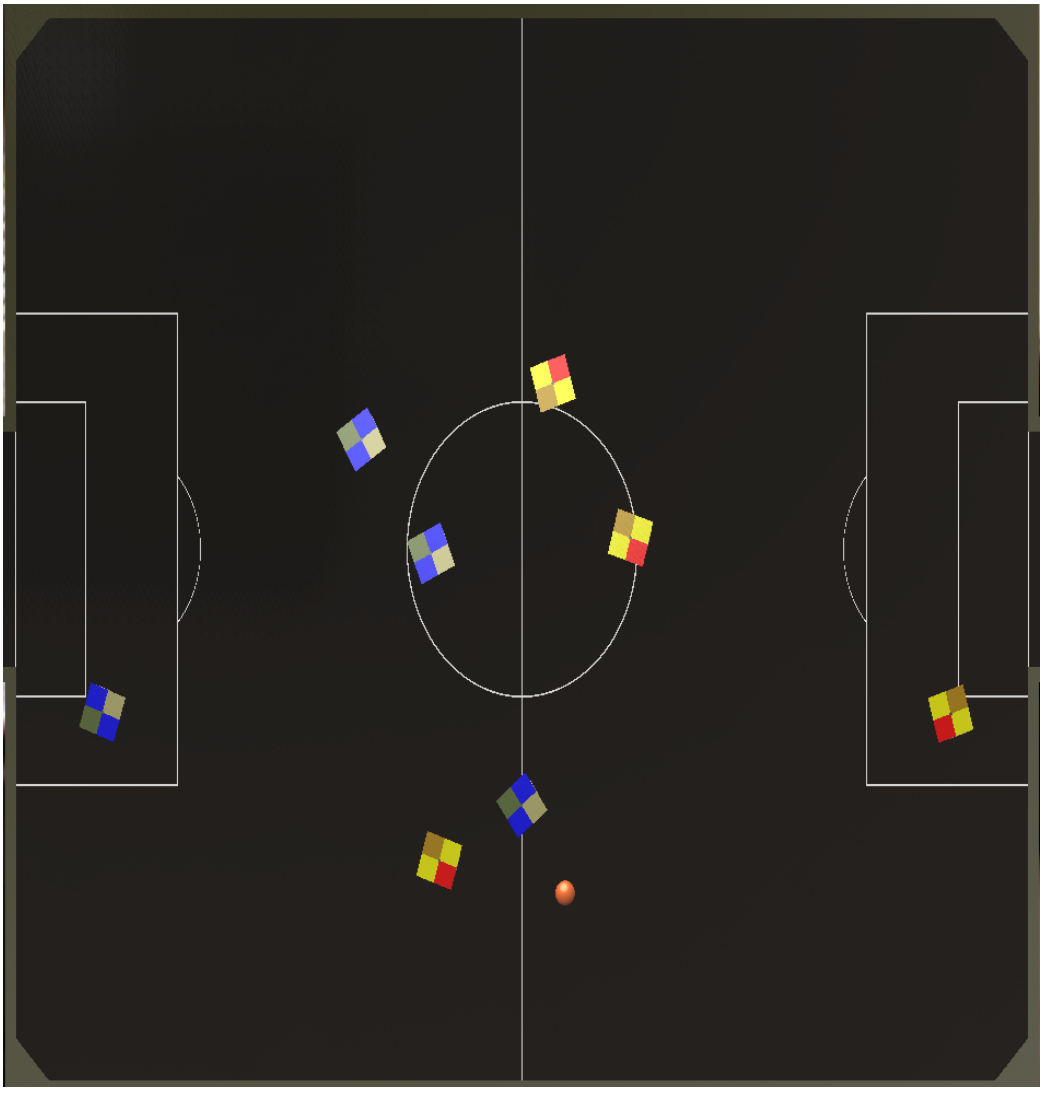
A.3 Nastavení programu

Veškeré parametry programu se nachází v samostaném souboru `params.cpp`. Následuje výčet a popis jednotlivých parametrů. Veškeré rozměry jsou uváděny v pixelech, pokud není řečeno jinak.

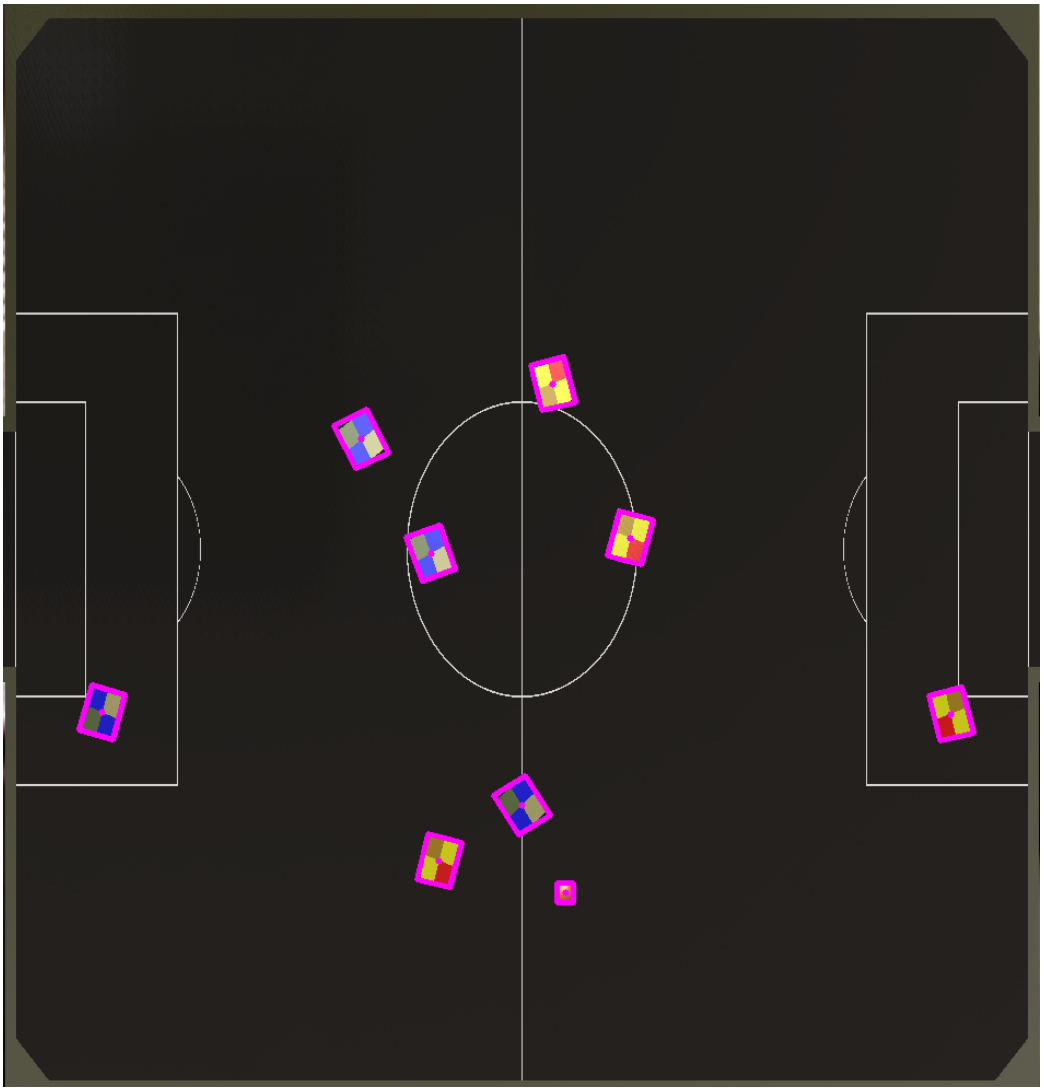
- `ROBOT_SIZE` – velikost strany robota.
- `COLOR_THRESH` – barevný práh pro segmentaci. (část 5.2.1)
- `REGION_THRES` – práh popředí v okolí podezřelého bodu. (část 5.1.2)
- `STEP_SCALE` – koeficient kroku prohledávání okolí bodu. (část 5.1.2)
- `BALL_DIAMETER` – průměr míčku.
- `AREA_TOLERANCE` – tolerance velikosti plochy robota.
- `LENGTH_TOLERANCE` – tolerance velikosti robota.
- `COLOR_L` – dolní práh pro určení vzdálenosti barvy od šedé. (část 5.2.1)
- `COLOR_U` – horní práh pro určení vzdálenosti barvy od šedé. (část 5.2.1)
- `MIN_SPAN_SQ` – druhá mocnina minimální vzdálenosti dvou rohů na obrysu. (část 5.2.2)

B Obrazová příloha

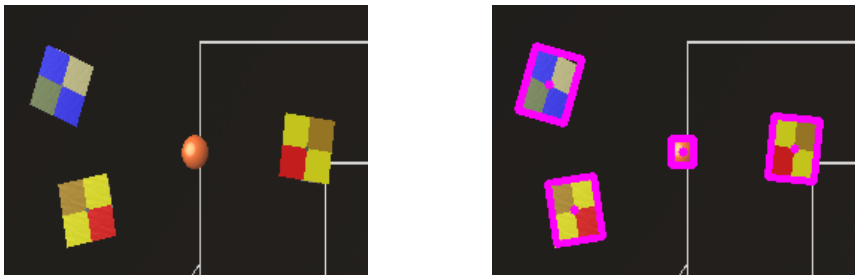
Na následujících obrázcích se nachází ukázky výstupu programu v různých situacích. Barva obrysu je závislé na postupu, který vedl k lokalizaci. Purpurová barva značí lokalizaci samostatného objektu, zelená označuje situaci kdy ROI obsahuje více samostatných robotů, červená označuje míček v situaci, kdy je v ROI více objektů. Azurová a žlutá označují míček a roboty v případě, že se jedná o shluk, kdy se obrysy slučují v jeden.



Obrázek B.1: Herní situace

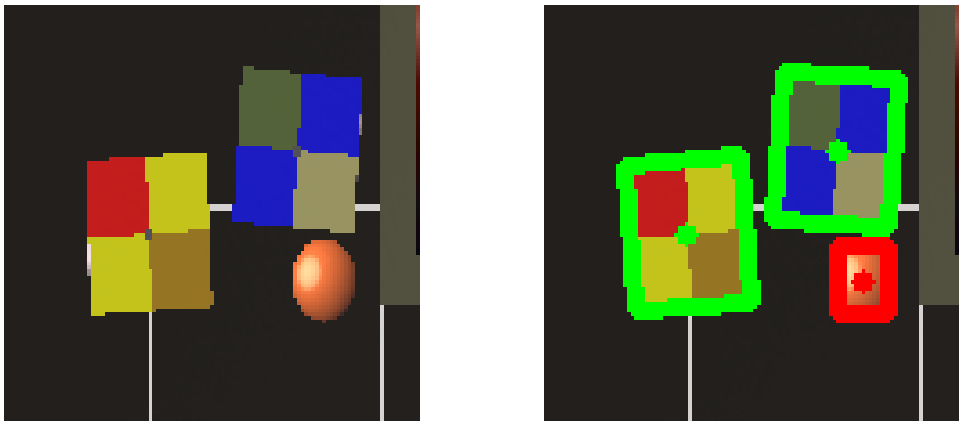


Obrázek B.2: Herní situace - výstup programu



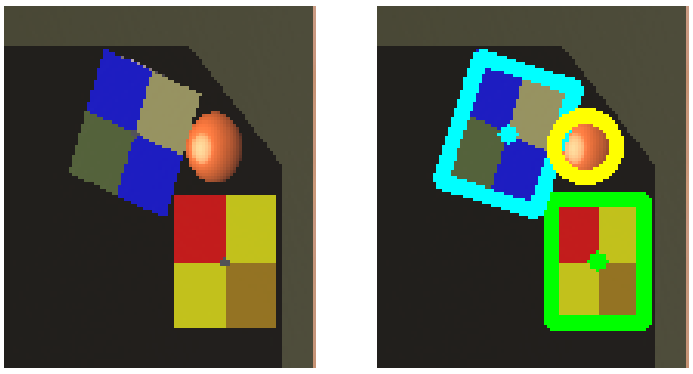
(a) vstup (b) výstup

Obrázek B.3: Skupina robotů s míčkem č.1



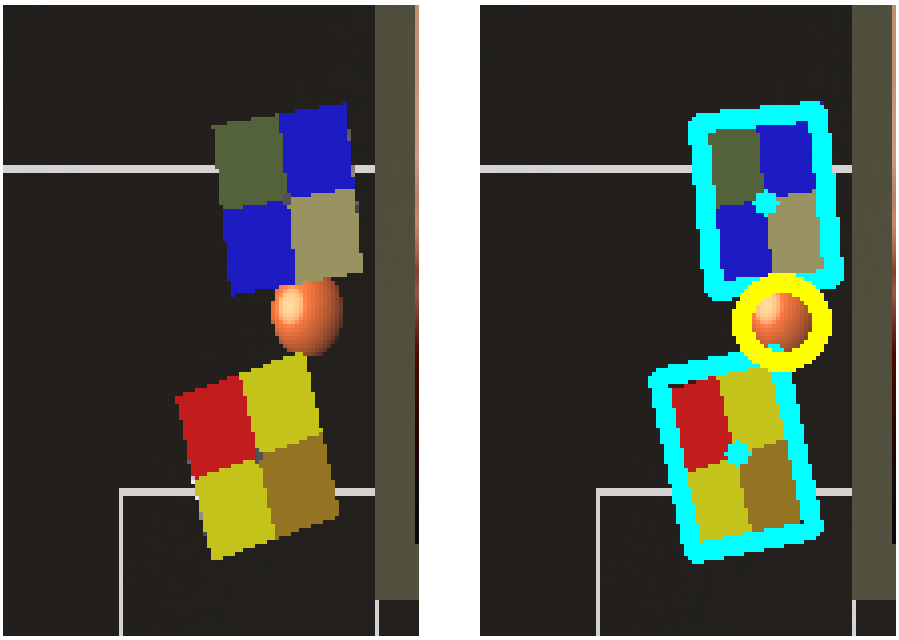
(a) vstup (b) výstup

Obrázek B.4: Skupina robotů s míčkem č.2



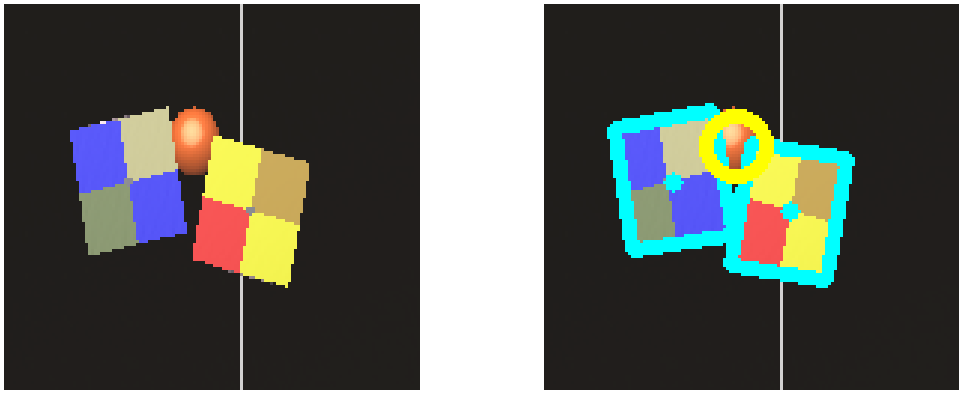
(a) vstup (b) výstup

Obrázek B.5: Skupina robotů s míčkem č.3



(a) vstup (b) výstup

Obrázek B.6: Skupina robotů s míčkem č.4



(a) vstup (b) výstup

Obrázek B.7: Skupina robotů s míčkem č.5