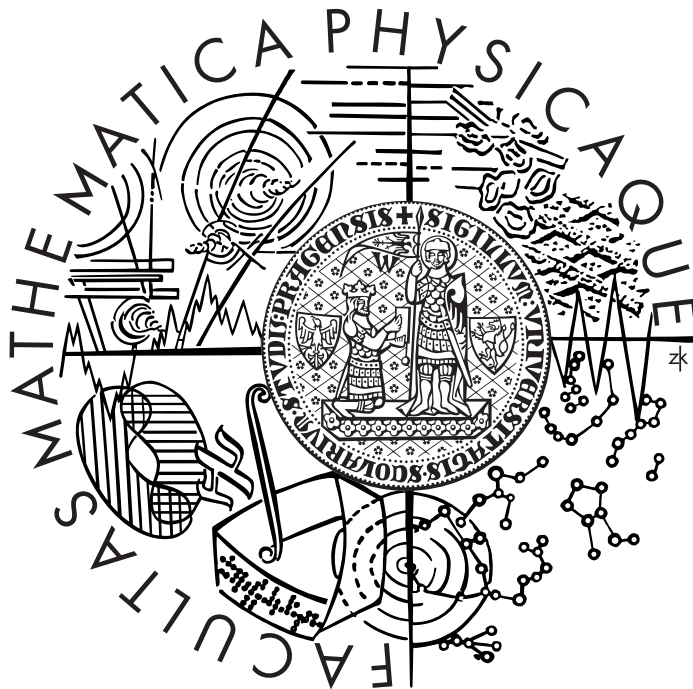


Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Jiří Marchalín

Řízení robotů s vizuálním vnímáním

Kabinet software a výuky informatiky
Vedoucí diplomové práce: RNDr. František Mráz, CSc.
Studijní program: Informatika, teoretická informatika

2008

Rád bych zde poděkoval mému vedoucímu RNDr. Františku Mrázovi, CSc. za cenné rady, konzultace a vedení. Také mé přítelkyni Hance a rodině za trpělivost a podporu, kterou projevovali po celou dobu psaní této práce. Děkuji.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze, 17. dubna 2008

Obsah

1	Úvod	5
2	Teoretické základy	7
2.1	Principy genetického algoritmu	7
2.1.1	Jedinec a populace	8
2.1.2	Reprezentace jedince	8
2.1.3	Ohodnocení jedince	8
2.1.4	Výběr populace (selekce)	8
2.1.5	Genetické operátory	10
2.1.6	Podmínka ukončení	11
2.1.7	Princip genetického algoritmu	11
2.2	Barvy a barevné prostory	12
2.2.1	Transformace barevných prostorů	14
2.3	Digitální obraz	15
2.3.1	Reprezentace digitálního obrazu	15
2.3.2	Barevná hloubka	16
2.4	Zpracování digitálního obrazu	16
2.4.1	Předzpracování obrazu	16
2.4.2	Analýza obrazu	17
2.4.3	Obrazové operátory	17
2.5	Robot	19
2.5.1	Prostředky robota pro vnímání prostředí	19
2.5.2	Prostředky robota pro pohyb	21
2.5.3	Diferenciální platforma	21
2.5.4	Robot e-puck	21
2.5.5	Řízení robota	22
3	Návrh řešení	25
3.1	Obrazové operátory	27
3.2	Napojení na řízení robota	28
3.3	Kolony	29

3.4	Genetický algoritmus	30
3.5	Simulátor	31
3.5.1	Player/Stage/Gazebo	31
3.5.2	Pyrobot	31
3.5.3	Microsoft Robotics Studio	32
3.5.4	Webots	32
4	Implementace	33
4.1	Moduly	34
4.1.1	Morpheus	34
4.1.2	Nyx	35
4.1.3	Testovací prostředí	35
4.1.4	Ovladač prostředí	35
4.2	Reprezentace obrazu	36
4.3	Obrazový operátor	36
4.4	Kolony	37
4.5	Genetický algoritmus	37
4.5.1	Reprezentace jedince	38
4.5.2	Výběr	38
4.5.3	Druhy	39
4.5.4	Elitismus	39
4.5.5	Stárnutí	40
4.5.6	Protekce	40
4.5.7	Genetické operátory	40
4.5.8	Operátory pracující s parametry	41
4.5.9	Operátory pracující se strukturou	42
4.5.10	Omezení velikosti kolony	43
4.5.11	Ohodnocení jedince	43
4.6	Navržené ovladače	44
4.6.1	Emulace kamerového vstupu	44
4.6.2	Simulátor Webots	44
4.7	Statistické charakteristiky populace	46
4.7.1	Hustota ohodnocení	46
4.7.2	Rozložení ohodnocení	46
4.7.3	Histogramy obrazových operátorů	47
4.8	Řízení robota	47
5	Experimenty	48
5.1	Emulace kamerového vstupu	48
5.1.1	Experimenty s emulací kamerového vstupu	49
5.2	Simulátor Webots	60

5.2.1	Experiment první	63
5.2.2	Experiment druhý	77
6	Závěr	80
A	Použité operátory	82
A.1	Filtry	82
A.2	Klasifikátory	101
B	Komunikační protokol	108
C	Formáty souborů	110
D	Návod na přidání nového obrazového operátoru	114
E	Konstrukce ovladače	118
F	Obsah přiloženého CD	120
G	Nastavení genetického algoritmu v experimentech	121
H	Obrazová příloha	128

Název práce: Vývoj řízení robotů s vizuálním vnímáním

Autor: Jiří Marchalín

Katedra (ústav): Kabinet software a výuky informatiky

Vedoucí diplomové práce: RNDr. František Mráz, CSc.

e-mail vedoucího: mraz@ksvi.mff.cuni.cz

Abstrakt: *Cílem této práce je návrh evolučního vývoje předřazeného filtrování pro kamerový vstup mobilního robota. Řízení mobilního robota s využitím kamerového vstupu je obecně obtížná úloha neboť informací přicházejících z tohoto vstupu je příliš mnoho a je nepraktické pracovat s nimi přímo. Toto množství informací je možné zredukovat za použití metod zpracování obrazu. Avšak pro různé úlohy není jednoduché sekvence filtrů předřazeného filtrování sestavovat ručně. Tato práce se zabývá automatickou konstrukcí takového předřazeného filtrování, které postupně zjednoduší a zmenší množství informace, které musí robot zpracovat, a dále pak rozhodne o obsahu obrazu. V rámci této práce byla navržena a implementována aplikace, která za pomoci genetického algoritmu a vhodně reprezentovaných filtrů navrhne sestavu těchto filtrů. Součástí práce jsou také experimenty na vybraných úlohách.*

Klíčová slova: vývoj řízení, vizuální vnímání, robotika, genetický algoritmus, zpracování obrazu

Title: Controlling Robots With Vision

Author: Jiří Marchalín

Department: Department of Software and Computer Science Education

Supervisor: RNDr. František Mráz, CSc.

Supervisor's e-mail address: mraz@ksvi.mff.cuni.cz

Abstract: *The goal of this thesis is the concept of the evolutionary development of the front-end camera input filtering for the mobile robot. Controlling mobile robot with the camera input is a generally hard task because it offers too much information and is impractical to process them directly. It is possible to reduce the amount of the information with use of front-end filtering using the methods of the image processing. However it is not easy to construct it manually for different tasks. This thesis proposes an automatic construction of such front-end filtering system that will reduce the amount of the information gathered from the camera input and also decides about the content of the image scene. It has also been implemented an application that hires genetic algorithm for the construction of the filtering sequences. The part of this thesis is experiments with selected tasks.*

Keywords: control development, visual perception, robotics, genetic algorithm, image processing

Kapitola 1

Úvod

V dnešní době se robotické systémy stávají stále dokonalejšími. Robot vnímá svoje okolí prostřednictvím senzorů, které dávají jen neúplnou informaci o scéně před robotem. Nejčastěji jsou používány například dotekové senzory a silové senzory, nebo detektory jasů. Jejich použití a vyhodnocení je rychlé, ale v mnoha případech není možné na jejich základě sestavit úplnou představu o situaci v níž se robot nachází. Pokud chceme získat velké množství dodatečných informací o okolním prostředí nabízí se možnost doplnit robota o kamerový systém, který by rozšiřoval jeho sensorické schopnosti. Problémem je značná velikost vstupních dat. Zatímco dotykové nebo silové senzory vracejí jen malé množství údajů, vizuální vjem poskytuje značné množství informací jejichž zpracování je již při malé velikosti obrazu značně výpočetně náročné. O těchto informacích není předem známo, kolik a které budou potřeba.

Jedním z přístupů jak zmenšit velikost obrazové informace je její filtrace pomocí obrazových operátorů. Na vstupní obraz aplikujeme v postupném sledu několik obrazových operátorů, které jsou schopny obrazovou informaci zmenšit (filtry), případně o ní podat informaci (detektory). Takto vznikne sekvence výsledků, které jsou ve srovnání s původní obrazovou informací zanedbatelně malé a jejich zpracování bude výpočetně jednodušší než přímé použití vstupního obrazu.

Vyvstává problém sestavení obrazových operátorů do sekvence a nastavení jejich parametrů. Je možné je sestavit pro konkrétní situaci ručně, ale v případě i malé změny scény je nutné, opět ručně, sekvenci překonfigurovat. Naším cílem je automatizovat návrh sekvence obrazových operátorů. Tento problém je složitý a jako jedna z možností jeho řešení se nabízí genetický algoritmus.

Genetický algoritmus je, za předpokladu dobré ohodnocovací funkce a vhodné reprezentace obrazových operátorů, schopný sestavit zadané sekvence namísto ručního sestavení. Genetický algoritmus pracuje s celou populací jedinců (robotů), pro které vyhodnocuje jejich kvalitu. Pro tento výpočet je tedy nutný běh více robotů a není rozumné použít roboty reálné. Můžeme pro vytváření vstupů pro řízení přejít k robotům modelovaným v simulátoru, například ve Webots [5].

Pro účely testování se jeví jako vhodná platforma robot e-puck [4], který byl zkonstruován pro výzkumné nasazení a obsahuje několik vhodných senzorů včetně kamery. Simulátor Webots zároveň obsahuje dobře propracovaný model tohoto robota a poskytuje tak vhodné prostředí pro simulaci. Při vhodně modelovaném prostředí je teoreticky možné přenést simulované řízení na robota reálného, tímto se však práce nezabývá.

Cílem této práce je vytvoření vhodného modelu prostředí, volba vhodné reprezentace obrazových operátorů a implementace genetického algoritmu, který je schopen sestavit potřebnou sadu obrazových operátorů k pevně danému, avšak strukturou neznámému, modelu řízení spolu se zadanou hodnotící funkcí. Navržené prostředí je otestované na vybraných úlohách.

Struktura práce je rozdělena do šesti kapitol. První z nich se věnuje úvodu k této práci. Druhá kapitola poskytuje teoretické základy z oblastí genetických algoritmů, pojmy z oblasti zpracování obrazu a její poslední část popisuje některé vybrané pojmy z oblasti robotiky a více rozebírá vlastnosti robota e-puck. Třetí kapitola popisuje navržené řešení.

Čtvrtá kapitola popisuje konkrétní implementované řešení. Pátá kapitola pak popisuje navržené experimenty, implementovaná prostředí a samotné výsledky těchto experimentů. Šestá kapitola shrnuje výsledky práce a obsahuje diskusi nad dosaženými výsledky. Součástí práce je autorem implementovaná aplikace použitá k experimentům v této práci, kterou je možno nalézt na přiloženém CD.

Kapitola 2

Teoretické základy

Tato kapitola obsahuje stručný úvod k oblastem, které jsou základem této práce. První část se věnuje genetickým algoritmům, následuje zpracování obrazu a poslední částí je krátký úvod do robotiky.

2.1 Principy genetického algoritmu

Genetický algoritmus je optimalizační technika, která využívá metody inspirované přírodní genetikou a přírodní druhovou selekcí [7]. Evoluce v přírodě je založena na přirozeném výběru jedinců z populace, kteří jsou schopni obstát v daném prostředí, a tito jedinci pak mají větší šanci se rozmnožit. Preferencí lepších jedinců a postupnou reprodukcí s jejich větším zastoupením v populaci dojde k postupnému zlepšování celé populace. V konečném důsledku bude, v ideálním případě, populace obsahovat alespoň jednoho jedince, který je schopen vyřešit problém (například problém nalezení potravy).

Podobné principy je možné využít pro aproximaci řešení úloh. Genetický algoritmus patří do třídy pravděpodobnostních algoritmů, ovšem oproti ostatním pravděpodobnostním algoritmům kombinuje prvky přímého a pravděpodobnostního prohledávání [9]. Po celou dobu výpočtu si udržuje více řešení daného problému, přičemž obvykle je počáteční skupina řešení vybrána náhodně. Genetický algoritmus pak rozvíjí řešení v postupných krocích, tzv. evolučních krocích. Na začátku každého kroku jsou všechna řešení ohodnocena a poté jsou vybrána ta s lepším ohodnocením, které reprezentuje míru úspěšnosti při řešení úlohy. Na tuto vybranou skupinu aplikuje genetické operátory, v základním provedení je to operátor křížení a mutace. Postupnou aplikací těchto operátorů se algoritmus snaží dosáhnout zlepšení na základě

předpokladu, že postupné křížení kvalitních počátečních řešení povede k vytvoření lepšího řešení.

2.1.1 Jedinec a populace

Jedinec reprezentuje jedno potenciální řešení úlohy. Shromažďuje veškeré informace o řešení, které genetický algoritmus potřebuje. Skupina N jedinců se nazývá populace. Tato populace je na počátku vybrána obvykle náhodně z množiny možných řešení [2].

2.1.2 Reprezentace jedince

V základní verzi implementace genetického algoritmu je reprezentace jedince bitový řetězec x_1, x_2, \dots, x_n . Reprezentace může být přímá, kdy řešením je samotný řetězec. Například při aproximaci hodnoty proměnné, pro kterou funkce nabývá maxima. Jiný možný způsob je reprezentace nepřímá, kdy řetězec reprezentuje návod na konstrukci řešení. Například reprezentuje čísla metod a jejich parametry. Tato reprezentace se někdy též nazývá genotyp [7].

2.1.3 Ohodnocení jedince

Ohodnocení jedince udává schopnost řešit danou úlohu. Čím lepší ohodnocení jedinec obdržel, tím lepší řešení reprezentuje. Samotný genetický algoritmus nemusí mít žádnou znalost o problému který řeší a ohodnocení může být ponecháno na prostředí. Ohodnocení (fitness) zajišťuje ohodnocovací funkce Φ .

$$\Phi : x_1, x_2, \dots, x_n \rightarrow \mathbb{R} \quad (2.1)$$

Z implementačních důvodů můžeme předpokládat, že výsledné ohodnocení je vždy nezáporné.

2.1.4 Výběr populace (selekce)

Předpokladem správného fungování genetického algoritmu je vhodný výběr (selekce) jedinců, kteří se budou podílet na tvorbě dalších generacích. Někdy se celý proces selekce nazývá reprodukce. Při výběru jedinců by mělo platit, že jedinec s lepším ohodnocením má větší pravděpodobnost, že bude vybrán.

Cílem výběru je vybrat co nejlepší jedince pro tvorbu další populace, přičemž jedinec může být vybrán několikrát [7].

Příliš agresivní proces výběru, výrazně preferující vysoko ohodnocené jedince může vést genetický algoritmus k příliš brzké konvergenci k lokálnímu maximu, naproti tomu slabý důraz na preferenci výborných jedinců může vést k divergenci [10].

Selekce ruletou

Mechanismus selekce by měl být postaven na principu, že čím je větší ohodnocení jedince, tím větší je pravděpodobnost jeho výběru. Jednou ze strategií, kterou můžeme pro selekci použít je princip rulety, kde pravděpodobnost přežití jedince i je dána rovnicí 2.2.

$$P_i = \frac{\Phi(x_1^{(i)} \dots x_n^{(i)})}{\sum_{j=1}^N \Phi(x_1^{(j)} \dots x_n^{(j)})} \quad (2.2)$$

Algoritmus je analogií ruletového kola s různě velkými výsečemi. Každá výseč rulety odpovídá jednomu jedinci a velikost výseče jeho kvalitě, tedy získanému ohodnocení. Roztočíme-li takovouto ruletu pak jedinci s většími úseky mají větší pravděpodobnost, že kulička spadne právě do jejich úseku, čímž je tento jedinec vybrán.

Vybereme náhodné číslo $c \in \langle 0, 1 \rangle$ a do nové populace vybereme jedince i takového, že platí (2.3) [9].

$$\sum_{j=1}^{i-1} P_j < c \leq \sum_{j=1}^i P_j \quad (2.3)$$

Značnou nevýhodou selekce ruletovým kolem je závislost na výsledném ohodnocení. Na počátku evolučního cyklu má většina jedinců malé ohodnocení, může se však vyskytnout skupina jedinců, kteří mají výrazně vyšší ohodnocení. Algoritmus rulety pak tyto jedince preferuje a ti brzy v populaci převládnu. Ovšem toto řešení může být pouze lokální maximum.

Selekce turnajem

Další možností výběru je turnaj mezi jedinci, při kterém algoritmus náhodně vybere k jedinců, z nichž vybere nejlepšího. Kritéria výběru mezi těmito k jedinci mohou být různá, lze například vybrat z této k -tice jedince s nejvyšším ohodnocením.

Jiným kritériem je výběr mezi dvěma jedinci na základě souboje. Na počátku algoritmu zvolíme konstantu $T \in (0, 1)$ a poté pro každou dvojici vybraných jedinců vygenerujeme náhodné číslo $t \in (0, 1)$. Pokud $t \leq T$ pak postoupí jedinec s lepším ohodnocením, v případě $t > T$ jedinec s horším ohodnocením. Abychom zachovali preferenci lepších jedinců mělo by platit, že $T > 0.5$ [9].

2.1.5 Genetické operátory

Genetické operátory jsou prostředkem genetického algoritmu ke změně jedince. Základními operátory jsou křížení a mutace.

Křížení

Při křížení genetický algoritmus vybere náhodně dvojice jedinců mezi kterými s pravděpodobností p_c dojde ke křížení. Pokud má ke křížení dojít, je vybrán náhodný bod křížení $k \in \{1, 2, \dots, n\}$.

$$\begin{aligned} \text{jedinec}_i &= x_1^{(i)} \dots x_k^{(i)} x_{k+1}^{(i)} \dots x_n^{(i)} \\ \text{jedinec}_j &= x_1^{(j)} \dots x_k^{(j)} x_{k+1}^{(j)} \dots x_n^{(j)} \end{aligned}$$

Bloky určené počátkem a bodem křížení jsou vzájemně vyměněny a vzniknou tak dva noví jedinci.

$$\begin{aligned} \text{jedinec}'_i &= x_1^{(j)} \dots x_k^{(j)} x_{k+1}^{(i)} \dots x_n^{(i)} \\ \text{jedinec}'_j &= x_1^{(i)} \dots x_k^{(i)} x_{k+1}^{(j)} \dots x_n^{(j)} \end{aligned}$$

Mutace

Při mutaci každý bit jedince může s pravděpodobností p_m změnit svou hodnotu na opačnou. Mutace prvoplánově slouží k obnově diverzity populace a neměla by být hlavním nástrojem genetického algoritmu ke změně jedince [8].

Elitismus

Vývoj populace v genetickém algoritmu je založen na výběru dobrých jedinců, jejich křížení a mutaci. V průběhu tohoto vývoje může dojít ke vzniku velmi dobrého jedince s vysokým ohodnocením. Avšak protože výběr jedinců je založen na pravděpodobnosti, může se stát, že tento jedinec nebude vybrán. Může se také stát, že v důsledku použití operátoru křížení nebo mutace může dojít ke ztrátě tohoto řešení, neboť jeho parametry budou změněny.

Metodou, která tomuto může zabránit je elitismus. Po každém ohodnocení populace je e jedinců s nejlepším ohodnocením uschováno a beze změny přejdou do další populace, neúčastní se tak výběru, ani aplikace genetických operátorů [7].

2.1.6 Podmínka ukončení

Ukončení běhu genetického algoritmu je obecně problematické. V průběhu algoritmu můžeme sledovat určité statistické veličiny, například průměrné ohodnocení jedinců v populaci, nárůst ohodnocení v průběhu času, a na základě těchto veličin běh algoritmu ukončit. Genetický algoritmus můžeme také ukončit po předem zadaném počtu evolučních kroků.

2.1.7 Princip genetického algoritmu

Základní schéma popsání genetického algoritmu je shrnuto v algoritmu 1.

Algoritmus 1 Schéma genetického algoritmu

- 1: Vytvoření populace
 - 2: Výpočet ohodnocení jedinců
 - 3: **repeat**
 - 4: Výběr nové populace ze staré
 - 5: Aplikace genetických operátorů
 - 6: Výpočet ohodnocení jedinců
 - 7: Nahrazení staré populace novou
 - 8: **until** Podmínka ukončení
-

2.2 Barvy a barevné prostory

Obraz můžeme chápat jako spojitou funkci dvou proměnných x a y , která je složením osvětlení scény l a odrazivosti objektů ve scéně r , viz (2.4). Získaná $f(x, y)$ je obvykle vektorem hodnot, který tvoří například jednotlivé složky základních barev v některém z barevných prostorů [13].

$$f(x, y) = l(x, y) \times r(x, y) \quad (2.4)$$

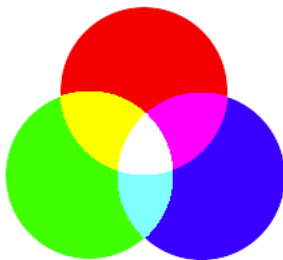
Lidské oko je schopno vnímat barvu díky fotopigmentu, který se rozděluje na tři základní složky: modrý, zelený a červený. Názvy těchto fotopigmentů vyjadřují jejich vnímavost vůči maximální vlnové délce spektra, výsledná vnímaná barva je pak důsledkem rekombinací hodnot R, G a B (červená, zelená a modrá) [13].

V přímé souvislosti s vnímáním barev jsou barevné prostory, které zachycují reprezentaci příslušných barev. Barevných prostorů existuje velké množství. V následujících odstavcích se zaměříme na dva, RGB a HSV, které jsou používány v této práci. Příklady dalších barevných prostorů je možno najít v [13] případně v [12].

RGB

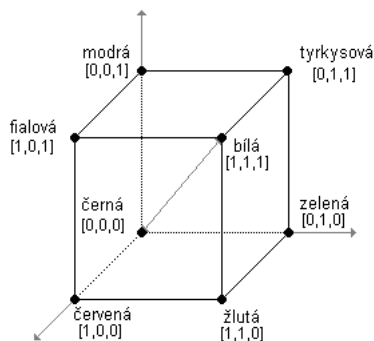
Barevný prostor RGB vychází z přirozeného vnímání lidského oka. Uchovává jednotlivé barvy jako poměr tří barevných složek, tzv. primárních barev. Tyto barvy jsou červená (**R**ed), zelená (**G**reen) a modrá (**B**lue), neboť na tyto barvy je lidské oko nejcitlivější. Barva je pak definována jako trojice (r, g, b) , kde $r, g, b \in \langle 0, 1 \rangle$. Hodnota 1 znamená maximální zastoupení jedné složky barvy a 0 pak nejmenší. Například hodnota $(1, 0, 0)$ reprezentuje červenou barvu bez jakéhokoliv zastoupení ostatních dvou barevných složek.

Barevný prostor RGB využívá efektu skládání barev, kdy využíváme pouze tři barvy ke generování všech ostatních. Princip je založen na vlastnostech vnímání barev lidským okem. Elektromagnetické vlnění o délce 580 nm vnímá lidské oko jako žlutou barvu, avšak stejným způsobem vnímá kombinaci červené a zelené [13]. Smícháním všech tří barev získáme například barvu bílou. Tento efekt je zobrazen na obrázku 2.1.



Obrázek 2.1: Aditivní skládání barev

Geometricky si tento prostor lze také představit jako krychli, kde $(0, 0, 0)$ vyjadřuje černou barvu a $(1, 1, 1)$ barvu bílou, viz náčrt 2.2. Prostor RGB je poměrně obtížně zpracovatelný, protože není zcela intuitivně zřejmé, jaká barva vznikne smícháním daných tří složek.



Obrázek 2.2: Geometrické zobrazení RGB

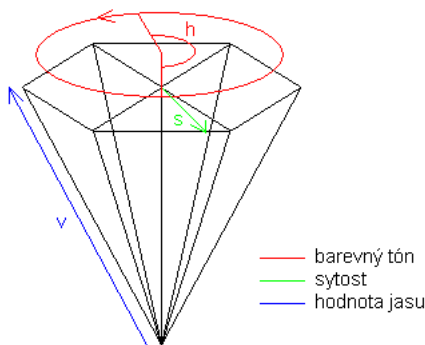
HSV

Barevný prostor HSV stejně jako RGB reprezentuje barvu jako tříložkový vektor hodnot. V případě HSV se však jedná o barevný tón (**H**ue), sytost (**S**aturation) a hodnotu jasu (**V**alue). Barevný tón vyjadřuje barvu spektra, která převládá, sytost vyjadřuje zastoupení ostatních barev ze spektra a jas vyjadřuje množství bílého světla [13].

V geometrickém vyjádření HSV trojice (h, s, v) , kde $h \in \langle 0, 360 \rangle$ a $s, v \in \langle 0, 1 \rangle$ určuje bod v šestibokém jehlanu. Výška jehlanu určuje hodnotu jasu, tedy bílá barva je zcela uprostřed reprezentovaná bodem $(0, 0, 1)$ a černá v bodě $(0, 0, 0)$. Úhel h určuje barevný tón a vzdálenost od středu určuje

sytost. Geometrické znázornění je na obrázku 2.3.

Prostor HSV je pro člověka intuitivní, neboť lze snadno odhadnout, která barva bude výsledkem kombinace složek h , s a v . Díky těmto svým vlastnostem se prostor HSV velmi dobře hodí k detekci různých barev v obraze.



Obrázek 2.3: Geometrické zobrazení HSV

2.2.1 Transformace barevných prostorů

Barvy lze reprezentovat v různých barevných prostorech a tyto reprezentace je možné na sebe vzájemně převádět. Toto může být nutné například v případě, kdy záznamové zařízení podává obraz v barevném prostoru RGB a filtr, který chceme použít, pracuje s barevným prostorem HSV.

RGB na HSV

Transformace z barevného prostoru RGB do HSV vypočte pro barvu (r, g, b) , $r, g, b \in \langle 0, 1 \rangle$ příslušnou hodnotu v prostoru HSV (h, s, v) , $h \in \langle 0, 360 \rangle$, $s, v \in \langle 0, 1 \rangle$ podle následujících podmínek [1].

$$h = \begin{cases} 0, & \text{jestliže } \max = \min \\ 60 \times \frac{g-b}{\max-\min} + 0, & \text{jestliže } \max = r \text{ a } g \geq b \\ 60 \times \frac{g-b}{\max-\min} + 360, & \text{jestliže } \max = r \text{ a } g < b \\ 60 \times \frac{b-r}{\max-\min} + 120, & \text{jestliže } \max = g \\ 60 \times \frac{r-g}{\max-\min} + 240, & \text{jestliže } \max = b \end{cases}$$

$$s = \begin{cases} 0, & \text{jestliže } \max = \min \\ \frac{\max - \min}{\max}, & \text{jinak} \end{cases}$$

$$v = \max$$

Hodnoty max a min jsou maximální a minimální hodnotou ze zpracovávané trojice (r, g, b) .

HSV na RGB

Transformace z barevného prostoru HSV do prostoru RGB probíhá podobným způsobem podle následujících podmínek [1].

$$\begin{aligned} h_i &= \left\lfloor \frac{h}{60} \right\rfloor \bmod 6 & p &= v \times (1 - s) \\ f &= \frac{h}{60} - h_i & q &= v \times (1 - fs) \\ & & t &= v \times (1 - (1 - f)s) \end{aligned}$$

$$(r, g, b) = \begin{cases} (v, t, p), & \text{jestliže } h_i = 0 \\ (q, v, p), & \text{jestliže } h_i = 1 \\ (p, v, t), & \text{jestliže } h_i = 2 \\ (p, q, v), & \text{jestliže } h_i = 3 \\ (t, p, v), & \text{jestliže } h_i = 4 \\ (v, p, q), & \text{jestliže } h_i = 5 \end{cases}$$

2.3 Digitální obraz

Obraz, který získáme ze záznamového zařízení, je spojitá funkce, avšak k počítačovému zpracování je třeba z takto získaného obrazu vytvořit obraz diskrétní. Při snímání obrazu digitální kamerou je tato transformace určena vlastnostmi snímacího čipu, především počtem obrazových bodů a rozsahem barev, které jsou k dispozici. Transformace obrazu ze spojitě reprezentace do diskrétní podoby se nazývá digitalizace [13].

2.3.1 Reprezentace digitálního obrazu

Obvyklou formou reprezentace digitálního obrazu je matice o rozměrech $m \times n$, jejíž každý bod reprezentuje hodnotu obrazového bodu, tzv. pixelu (*picture element*). Další varianty datových struktur pro obrazovou analýzu,

například *Integral image*, *Co-occurrence matrix* a *Quadtrees* jsou uvedeny v knize [16].

Schéma (2.5) znázorňuje reprezentaci digitálního obrazu pomocí matice, přičemž levý horní bod snímaného obrazu reprezentuje levý horní bod matice.

$$\begin{bmatrix} I(0,0) & I(0,1) & \cdots & I(0,m-1) \\ I(1,0) & I(1,1) & \cdots & I(1,m-1) \\ \vdots & \vdots & & \vdots \\ I(n-1,0) & I(n-1,1) & \cdots & I(n-1,m-1) \end{bmatrix} \quad (2.5)$$

2.3.2 Barevná hloubka

Barevná hloubka určuje počet barev, které jsme schopni reprezentovat. Tento počet barev je dán počtem bitů, kterými je reprezentován jeden obrazový bod. Obvyklá hodnota je 1, 8 nebo 24 bitů ačkoliv v literatuře (například v [13]) se objevují i další možné hodnoty. V této práci používáme reprezentaci jednoho obrazového bodu pomocí 24 bitů, ve kterém používáme 8 bitů na každou barevnou složku. Tato reprezentace velmi dobře odpovídá barevnému prostoru RGB.

2.4 Zpracování digitálního obrazu

Obrazová data, která získáme ze snímače mohou být pro pozorovatele obtížně čitelná, například mohou obsahovat vysoké procento digitálního šumu, který vzniká nepřesným sejmutím hodnoty snímače a také ztráty informace vzniklé digitalizací. Mohou být nevýrazná a k jejich správné interpretaci je nutné některé partie obrazových dat zvýraznit a jiné naopak potlačit. Pro člověka je poměrně snadné daná obrazová data správně interpretovat, ale pro počítač je to obtížná úloha.

V této části můžeme použít metody zpracování digitálního obrazu, které můžeme rozdělit na dvě hlavní části, předzpracování obrazu a následnou analýzu obrazu.

2.4.1 Předzpracování obrazu

Předzpracování obrazu (*image enhancement* nebo *image pre-processing*) zahrnuje operace s obrazem na úrovni obrazových dat bez jakékoliv interpretace

obrazu. Předzpracování informací obsaženou v obrazových datech zmenšuje. Avšak může hrát významnou roli pro pozdější interpretaci obrazových dat [16].

2.4.2 Analýza obrazu

Cílem analýzy obrazu je poskytnout srozumitelnou informaci o obsahu a poskytnout informaci o objektech nasnímaných v obrazových datech. Obrazová data, která prošla předzpracováním poskytují samy o sobě stále příliš mnoho informací, kterým je těžké porozumět. Metody analýzy obrazu umožňují stanovit charakteristiky obrazu a na jejich základě poskytnout informaci o objektech ve scéně, jejich tvaru, vzdálenosti, barevnosti nebo třeba o rozložení objektů v prostoru. Jedním ze způsobů takovéto interpretace je zařadit daný obraz do nějaké třídy, která sdružuje typy obrazů se stejnými vlastnostmi.

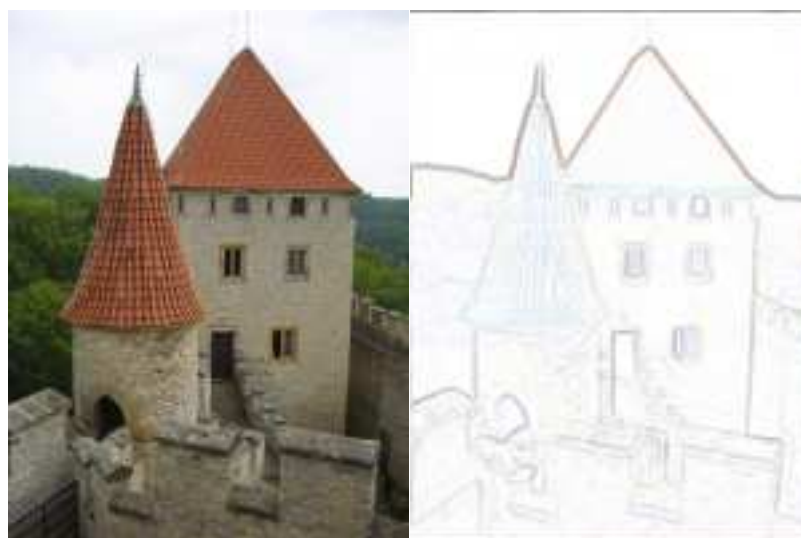
Obě uvedené části můžeme realizovat jako tzv. obrazové operátory pracující s obrazem, tyto si dále popíšeme.

2.4.3 Obrazové operátory

Z předchozího vyplývá, že obrazový operátor je jakákoliv operace prováděná s obrazovými daty. Speciálním případem obrazového operátoru je filtr, který můžeme vyjádřit jako transformaci obrazových dat (2.6), kde $m \times n$ jsou výchozí rozměry matice reprezentující digitální obraz ($m, n \in \mathbb{Z}^+$) a $k \times l$ jsou rozměry cílové matice po aplikaci obrazového operátoru ($k, l \in \mathbb{Z}^+$ a $k \leq m, l \leq n$).

Příkladem může být například konvoluční operátor Sobelova filtru, který zvýrazní hrany v obraze. Příklad práce tohoto filtru je na obrázku 2.4 Více o filtrech viz příloha A o použitých filtrech.

$$\mathcal{O} : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{k \times l} \quad (2.6)$$



(a) Původní obrázek

(b) Zvýraznění hran

Obrázek 2.4: Příklad práce Sobelova operátoru

Obrazové operátory použijeme i v případě analýzy obrazu. V tom případě nahradíme vztah (2.6) vztahem (2.7). Pro potřeby analýzy obrazu nazveme takto definované operátory klasifikátory.

$$\mathcal{K} : \mathbb{R}^{k \times l} \rightarrow \mathbb{R} \quad (2.7)$$

Vstupem klasifikátoru jsou tedy obrazová data, která mohla projít předzpracováním obrazu a výstupem je klasifikace obrazu vyjádřená například jedním reálným číslem.

2.5 Robot

Robot je obvykle chápán jako virtuální nebo reálný programovatelný elektromechanický stroj schopný samostatně vykonávat určitou činnost [14]. ANSI/RIA (*Robotics Industrial Association*) R15.06-1999 definuje robota jako automaticky kontrolovatelný, víceúčelový manipulátor programovatelný ve třech nebo více osách, který může být staticky připevněn na místě nebo na mobilním podvozku pro použití v průmyslových robotických aplikacích [3].

Systémů, které jsou považovány za roboty je velké množství a jejich taxonomie je obecně složitá, neboť neexistuje jedna univerzální definice. V různých oblastech a v různých společnostech jsou definice různé.

V této práci se budeme zabývat výhradně pozemními roboty s kolovým mechanismem pohybu jakým je například již zmíněný robot e-puck.

2.5.1 Prostředky robota pro vnímání prostředí

Lidské vnímání je založeno na vrozených smyslech mezi které patří mimo jiné hmat, čich, sluch a zrak. Tyto prostředky vnímání poskytují informaci pro nervové centrum člověka potřebnou k rozhodnutí o stavu okolí. Podobný systém je potřebný i pro autonomního robota, neboť na základě vnímání prostředí může být schopen rozhodování. U robota se prostředky vnímání nazývají senzory. Dále budeme uvažovat pouze senzory digitální, jejichž principem je převod některé fyzikální veličiny ve formě spojitého signálu do digitálního, toto zobrazuje schéma 2.5 [20].



Obrázek 2.5: Obecné schéma senzoru, převzato z [20]

Snímač zaznamená hodnotu aktuální měřené fyzikální veličiny, to může být například teplota okolí, výstupem snímače může být spojitý signál ve formě napětí, velikosti proudu, kapacity nebo jiné fyzikální veličiny. Převodník ze spojitého, analogového, signálu do digitálního obvykle vyžaduje na vstupu velikost napětí. Pokud snímač zaznamenává hodnoty jiné fyzikální veličiny, přichází na řadu konvertor signálu. Konvertor signálu transformuje

hodnoty měřené veličiny na napětí [20]. A/D převodník (*Analog/Digital*) převede naměřené hodnoty do digitální podoby.

Dále uvedeme principy některých vybraných senzorů, které používá již zmíněný robot e-puck.

Optické senzory

Optické senzory využívají snímače citlivé na různé části elektromagnetického spektra. Obvyklou formou jsou snímače citlivé na jas okolního světla, schopné rozlišit například tmavý a světlý objekt. S úspěchem se také využívají dálkové optické senzory, které pracují na principu vyzařování elektromagnetického záření. Vyzářené elektromagnetické záření, nejčastěji infračervené záření, se odrazí od objektů před senzorem, který pak počítá dobu od vyslání záření do přijetí odrazu. Ze znalosti rychlosti pohybu elektromagnetického záření c , doby t od vyzáření k odrazu je možné spočítat vzdálenost s senzoru od objektu dle vzorce (2.8), pohyb robota mezi dobou záření a odrazu zanedbáme.

$$s = \frac{ct}{2} \quad (2.8)$$

Kamera

Kamera je záznamové zařízení, které snímá okolní prostředí a poskytuje robotovi komplexní pohled na scénu před ním. Je možné použít kameru analogovou, avšak pro účely zpracování obrazu je vhodnější použít kameru digitální. Digitální kamera používá k snímání obrazu speciální čip, obvykle CCD (*Charge-coupled device*). Čip CCD využívá k záznamu obrazu fotoelektrický jev, kdy částice světla (foton) dokáže dostat elektron do excitovaného stavu.



Obrázek 2.6: Kamerový výstup v simulátoru Webots

2.5.2 Prostředky robota pro pohyb

Pohyb mobilního robota zajišťují tzv. aktuátory, obvykle se jedná o servomotory. V našem případě uvažujeme kolového robota umístěného na diferenciální platformě.

2.5.3 Diferenciální platforma

Diferenciální platforma je velmi účinným typem nosné jednotky robota. Základem je dvojice nezávisle zavěšených kol, které jsou řízeny dvěma motory. Rozdílným nastavením výkonu motorů je možné robotem zatáčet, princip je stejný jako u diferenciálního pohonu, kdy se kola pohybují různými rychlostmi [6].

2.5.4 Robot e-puck

Robot e-puck byl původně vyvinut v institutu EPFL (École Polytechnique Fédérale de Lausanne) s původním záměrem vyrobit robota pro výzkumné účely. Robot e-puck je velmi malý a odolný, založený na platformě s diferenciálním řízením s několika základními senzory. Zajímavostí je, že se jedná a tzv. open hardware, jehož plány jsou volně dostupné a jde tedy o stejný princip jako v případě open source programů. Průměr platformy, na které je postaven, je 70mm a výška celého robota je 50mm. Váží pouhých 200 g. Obrázek 2.7 znázorňuje robota e-puck (převzat ze stránek Cyberbotics¹).



Obrázek 2.7: Robot e-puck

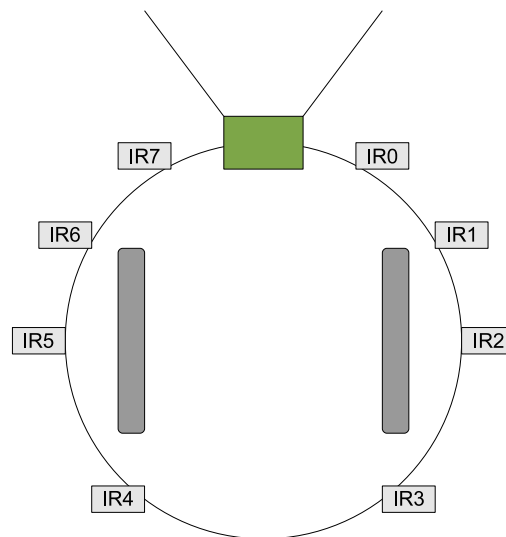
¹<http://www.cyberbotics.com/products/robots/e-puck.html>

Senzory robota e-puck

Robot e-puck má k dispozici 8 infračervených senzorů, 2 enkodéry, akcelerometr, 3 mikrofony a barevnou kameru s rozlišením 480×640 pixelů. Některé senzory, které jsou použity v této práci si dále popíšeme.

Kamera Kamera má celkové rozlišení 480×640 pixelů, avšak procesor e-pucku (dsPIC) není schopen takto velkou informaci zpracovat najednou a přenos plného rozlišení se musí realizovat v několika krocích. K získání celého obrazu v jednom kroku je nutné podstatně snížit použité rozlišení. Model e-pucku v použitém simulátoru má přednastaveno rozlišení kamery 52×39 pixelů.

Dálkové senzory Robot má 8 infračervených senzorů, které používá k detekci překážek, jejich rozmístění je znázorněno na schématu 2.8.



Obrázek 2.8: Schéma senzorů

2.5.5 Řízení robota

Předpokládejme, že mobilní robot má k dispozici $n \in \mathbb{Z}^+$ senzorů, které můžeme využívat. Definujme jejich výstupy jako množinu $S = \{s_1, s_2, \dots, s_n\}$, kde s_i je například reálné číslo nebo číslo celé. Dále předpokládejme, že se mobilní robot v každém okamžiku nachází v nějakém stavu q z množiny stavů $Q = \{q_1, q_2, \dots, q_m\}$, kde $m \in \mathbb{Z}^+$. Dále máme k dispozici množinu

akcí $A = \{a_1, a_2, \dots, a_k\}$, kde $k \in \mathbb{Z}^+$, které může robot vykonávat. Systém řízení R si pak můžeme představit jako (2.9), kde $a \in A$ a $q_i, q_j \in Q$.

$$R : (q_i, s_1, s_2, \dots, s_n) \rightarrow (q_j, a) \quad (2.9)$$

Samotný mechanismus rozhodování může mít několik podob. Mezi nejjednodušší patří podmínkové řízení.

Podmínkové řízení

Podmínkové řízení je sestava několika různě komplexních podmínek $P(Q, S)$, které vybírají akce z množiny A na základě hodnot přijatých ze vstupů. Avšak údržba takového typu řízení není jednoduchá. V algoritmu 2 je znázorněno jednoduché schéma podmínkového řízení.

Algoritmus 2 Podmínkové řízení

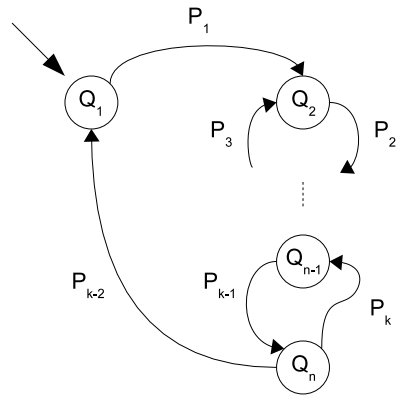
```

1: repeat
2:   Hodnota sensorů  $\leftarrow$  Načti senzory
3:   if  $P_1(Q, S)$  then
4:     ...
5:   else if  $P_2(Q, S)$  then
6:     ...  $\vdots$ 
7:   else if  $P_x(Q, S)$  then
8:     ...
9:   end if
10: until Robot vypnut

```

Řízení konečným automatem

Podmínkové řízení má řadu nevýhod, mezi ně patří nesnadné udržování komplexních podmínek, dále jakákoliv změna nebo přidání akce do množiny A může být velmi obtížná. Částečnou odpovědí na tyto problémy je implementace řízení pomocí stavového automatu. Řízení konečným automatem nám umožní snadněji provádět korekce pohybu robota na základě vstupů ze sensorů, čímž by řízení mělo být robustnější vůči testovacímu prostředí. Jedno z možných schémat implementace je znázorněno na obrázku 2.9, přičemž P_i jsou přechody mezi stavy Q_i .



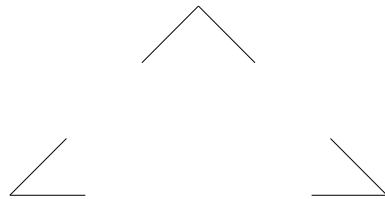
Obrázek 2.9: Schéma řízení robota konečným automatem

Kapitola 3

Návrh řešení

Připojíme-li k mobilnímu robotovi kameru získáme tak bohatou dodatečnou informaci o okolním prostředí, která je velmi dobře srozumitelná pro člověka, nikoliv však pro stroj. Zatímco výstupy z ostatních senzorů jsou v mnoha případech jednoduše interpretovatelná skupina čísel, tak v případě obrazu z kamery má robot k dispozici $m \times n$ hodnot, kde m a n je šířka a výška obrazu. Z těchto informací musí robot rozpoznat co se děje před ním.

Například na obrázku 3.1 je obraz s velmi jednoduchým obrazcem. Pro člověka je snadné poznat v něm trojúhelník, ale pro jednoduché řízení robota to je však obtížná úloha. Je těžké a nepraktické navrhovat řízení robota, které pracuje přímo s takovýmito obrazovými daty získanými z kamery.



Obrázek 3.1: Obrazec

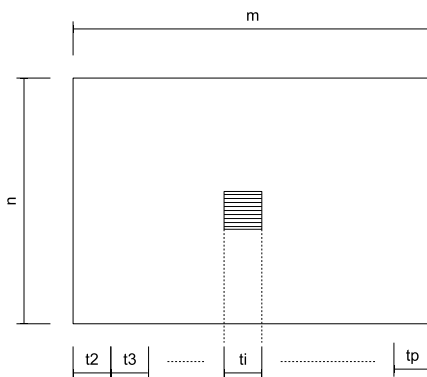
Tento problém můžeme rozdělit na dvě části. Na část, která se zabývá snížením množství informací, a na část, která se zabývá interpretací scény zachycené na obraze.

Množství informací, které obraz poskytuje, můžeme snížit některým z obrazových operátorů filtrujících obraz, které jsme popsali v kapitole o teoretických základech a které transformují obraz $I(x, y)$ o rozměrech $m \times n$ na obraz

$I'(x, y)$ o rozměrech $k \times l$, kde $k \leq m$ a $l \leq n$. Tato transformace je vždy ztrátová. Důležité je, aby zůstala zachována informace relevantní pro řešení dané úlohy. Ovšem pro různé úlohy je obtížné říci, které filtry odstraní jen tu správnou část informací, a které naopak ponechají informace nepotřebné nebo dokonce odstraní informace nutné k řešení úlohy.

Pro popis dalšího postupu dále předpokládejme, že jsme vybrali vhodné filtry, které v obraze zanechají pro danou úlohu relevantní informace. Robot má tedy již k dispozici obraz, který například obsahuje pouze červené objekty. Robot by nyní chtěl tuto informaci nějakým způsobem využít. Přesto, že informace je již značným způsobem redukována, je stále obtížné zkonstruovat řízení, které by ji využilo. Pro robota by bylo vhodné tuto informaci ještě dále interpretovat a poskytnout mu množství informací, které bude schopen zpracovat.

Řízení robota by mělo být schopné tuto interpretaci rychle vyhodnotit proto by neměla být implementována jako složitá struktura. Vhodný způsob interpretace je například jedno reálné číslo z intervalu $\langle 0, 1 \rangle$. Představme si například situaci, kdy chceme určit horizontální pozici objektu v obraze. Pro tento konkrétní případ si můžeme definovat $t_1 = 0$ (v případě, že objekt se v obraze nenachází), $t_2 = 0.01$ (v případě, že obraz je v levé krajní části obrazu) a $t_p = 1$ (v případě, že je obraz v pravé krajní části obrazu). Příklad této situace ilustruje obrázek 3.2.

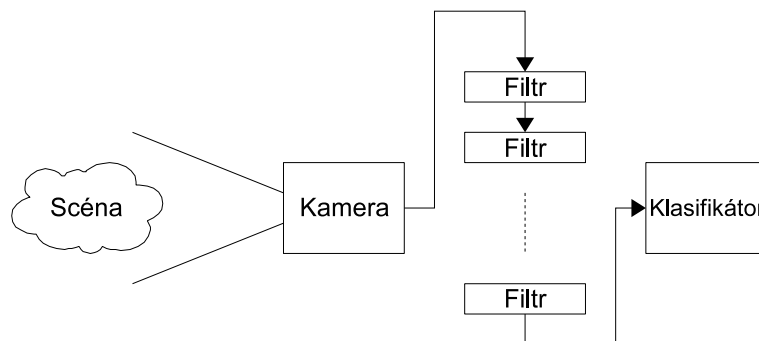


Obrázek 3.2: Interpretace obrazce

Interpretací obrazu v této situaci je prvek t_i , v konkrétní implementaci si jej můžeme představit například jako $t_i = \frac{i}{m}$. Zůstává otázka, jakým způsobem takto obraz klasifikovat.

V kapitole o teoretických základech jsme definovali skupinu obrazových operátorů, které klasifikují obraz do daných tříd. Konkrétně jsou jejich vstupem obrazová data a jejich výstupem jedno reálné číslo. Implementujeme tedy tyto obrazové operátory tak, aby jejich výstup byl v rozmezí $(0, 1)$ a nazveme je klasifikátory

Nyní máme k dispozici dva druhy obrazových operátorů, jedním jsou filtry a druhým klasifikátory, jejichž prostřednictvím jednak snížíme množství informací a zároveň interpretujeme obraz. Různé filtry mohou zachycovat a upravovat různé typy informací, je proto vhodné mít možnost jejich zřetězení. Uvedené řešení pak vede na schéma 3.3.



Obrázek 3.3: Postup filtrování obrazových dat

V další části této kapitoly si popíšeme konkrétněji obrazové operátory.

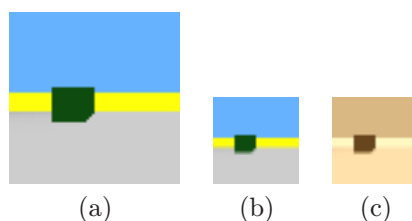
3.1 Obrazové operátory

Řídící elektronika uvažovaného robota je obvykle málo výkonná. Především se jedná o slabý procesor a malou velikost operační paměti. Při vybavení robota kamerovým vstupem jsou na tuto elektroniku kladeny vysoké nároky, neboť v krátkém časovém úseku musí zpracovat poměrně velké množství dat ze sensorů. Rychlost běžné kamery je průměrně kolem 25 snímků za vteřinu, pro řízení robota se slabým procesorem je to příliš velká informace, kterou málokdy zvládne zpracovat. Zároveň je tato rychlost pro některé úlohy zbytečně vysoká a v případě řízení mobilního robota je dostačující hodnotou 3-10 snímků za vteřinu.

Obrazové operátory proto musí být jednoduché a rychle proveditelné i na slabém procesoru v reálném čase. Také by tyto obrazové operátory, jejichž vlastnosti to umožňují, měly být parametrizovatelné. Například filtr, pracující s jednotlivými barevnými složkami v reprezentaci RGB, by měl mít barevnou složku, se kterou pracuje, zadánu parametrem. Parametry u obrazových operátorů značným způsobem zvyšují jejich variabilitu.

Na obrázku 3.4 je příklad postupného filtrování. Prvním obrazem je vstupní obraz 3.4a o rozměrech $m \times n$, předpokládejme m a n sudé, obsahující čtyři barvy. Cílem je detekovat zelený čtvelec v levé části obrazu. Pro tento příklad předpokládejme, že nemáme k dispozici žádný klasifikátor, který je schopen přímo rozhodnout o existenci zelené barvy v obraze, jinými slovy žádný takový, který je schopen detekovat zelenou barvu. Abychom zmenšili informaci pro práci dalších filtrů, zmenšíme rozlišení obrazu aplikací obrazového operátoru snížení rozlišení. Tím nám vznikne nový obraz 3.4b o rozměrech $k \times l$, kde $k = \frac{m}{2}$ a $l = \frac{n}{2}$.

Rozsahy čtyř barev jsou velké a tak použijeme operátor Sepia, který barvy v obraze transformuje do odstínů hnědé. Rozměry obrazu přitom nemění. Z transformovaného obrázku 3.4c je patrné, že hledaný objekt se stal nejtmavší oblastí v obraze. Nyní můžeme použít například klasifikátor, který používá prahování k detekci objektů v obraze. Bez předchozí úpravy pomocí filtrů by jeho práce byla značně ztížena, nebo zcela nemožná.



Obrázek 3.4: Příklad filtrace

3.2 Napojení na řízení robota

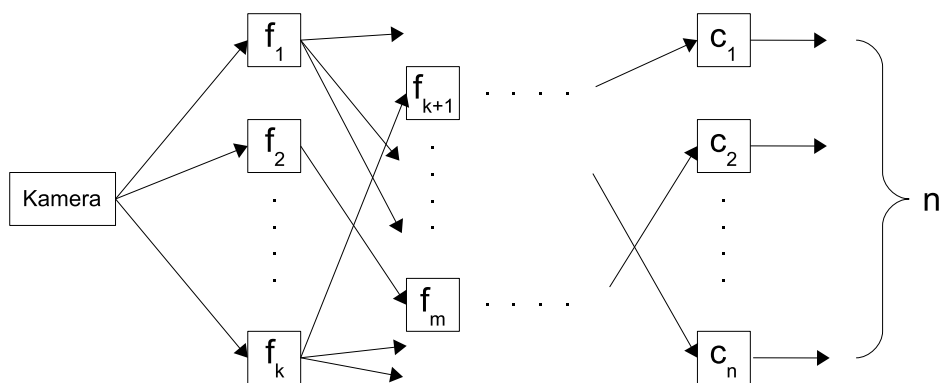
V případě obyčejných senzorů, jako jsou například enkodéry, řízení přečte právě jednu hodnotu z každého senzoru a vyhodnotí situaci. V případě kamerového vstupu není počet hodnot, které z jednoho obrazu můžeme získat, prakticky nijak omezen. Z jednoho obrazu zachyceného kamerou můžeme získat několik různých hodnot, tedy například pozici některého objektu v obraze

a vzdálenost robota od objektu (obrazové operátory, které tohoto dosáhnou, jsou popsány v příloze A o použitých operátorech).

Z předchozího popisu návrhu vyplývá, že k interpretaci obrazu sestavíme skupinu filtrů zakončenou jedním klasifikátorem. Předpokládejme, že řízení očekává n hodnot, interpretací, z kamerového vstupu. V takovém případě bychom měli zkonstruovat soustavu obrazových operátorů s n výstupy, kde každý výstup je výstupem vhodného klasifikátoru.

3.3 Kolony

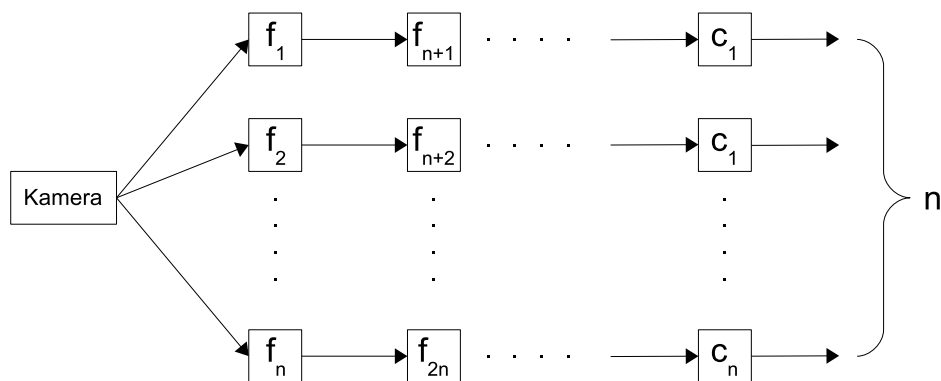
Z předchozí části vyplývá, že řešení vede na konstrukci kolon obrazových operátorů. Předpokládejme, že máme k dispozici množinu $\mathcal{F} = \{f_1, f_2, \dots, f_r\}$, $r \in \mathbb{Z}^+$ obsahující filtry a množinu $\mathcal{C} = \{c_1, c_2, \dots, c_s\}$, $s \in \mathbb{Z}^+$ obsahující klasifikátory. Z těchto množin sestavíme potřebné kolony. Jelikož máme k dispozici pro všechny kolony stejný vstupní obraz, nabízí se možnost tvořit stromovou acyklickou strukturu, ve které jednotlivé uzly představují filtry a klasifikátory. Obrázek 3.5 znázorňuje ideu těchto obecných kolon, kde $f_i \in \mathcal{F}$ a $c_i \in \mathcal{C}$.



Obrázek 3.5: Obecný návrh kolon

Takto obecný návrh má výhody z hlediska výkonu výpočtu. Za určitých okolností je v něm možné využít výstupu jednoho filtru pro více kolon. Problém, který může nastat u tohoto řešení je přílišné provázání jednotlivých filtrů. I malá změna jednoho filtru nebo jeho parametrů může ovlivnit více výstupů. Tento problém lze vyřešit tak, že nebudeme vyžadovat více výstupů z jednoho filtru, ale budeme konstruovat samostatné sekvence filtrů

pro každý výstup zvlášť. Je zřejmé, že jsme tímto možností kombinací filtrů ani výpočetní schopnosti sekvencí nezměnili. Změny jednotlivých filtrů v těchto samostatných sekvencích neovlivní sekvence ostatní, ovšem za cenu, že nebudeme moci použít výstup jednoho filtru vícekrát a tedy zvýšíme náročnost výpočtu. Tento návrh zobrazuje schéma 3.6.



Obrázek 3.6: Návrh kolon

Zůstává však problém sestavení těchto kolon. Ruční sestavování kolon je možné, ale velmi pracné a vyžaduje kvalifikovaný odhad. Pokud není na první pohled zřejmé, které filtry a klasifikátory je potřeba použít, je nutné vyzkoušet velké množství kombinací. Přihlédneme-li k tomu, že filtry a klasifikátory mají své parametry ovlivňující jejich chování, značným způsobem se nám zvětšuje prostor, ze kterého kombinace vybíráme.

V kapitole o teoretických základech jsme představili techniku prohledávání rozsáhlého stavového prostoru pomocí genetických algoritmů. Při vhodné reprezentaci jedinců by genetický algoritmus mohl sestavit kolony i nastavit parametry jednotlivých obrazových operátorů.

3.4 Genetický algoritmus

Naše představa řešení je taková, že budeme sestavy kolon považovat za jedince v rámci populace genetického algoritmu. Každý jedinec tedy představuje jedno řešení. Jeho konkrétní reprezentaci popíšeme v kapitole o implementaci. K dispozici máme mobilního robota s pevně daným řízením a předpokládáme, že nejsme schopni jej měnit.

Při řešení úlohy pomocí genetického algoritmu budeme potřebovat ohodnotit nalezená řešení, obvykle ve velkém množství. Implementovat nalezená řešení do řízení reálného robota a testovat chování tohoto řešení přímo v reálném prostředí je vzhledem k potřebě značné rychlosti nevhodné a k výpočtům genetického algoritmu nepraktické. V tomto případě se jeví jako vhodná varianta použít některý z dostupných robotických simulátorů, v němž budeme mít k dispozici model reálného robota. Ohodnocení nalezených řešení pak provedeme v simulátoru. Dále si popíšeme vlastnosti několika dostupných robotických simulátorů.

3.5 Simulátor

Pro řešení úlohy ohodnocení jedince jsme uvažovali celkem čtyři hlavní dostupné robotické simulátory. V následující části krátce shrneme jejich výhody a nevýhody.

3.5.1 Player/Stage/Gazebo

Projekt Player¹ je robotický simulátor pro platformu Linux. Samotný simulátor je rozdělen do tří vrstev. Player, sloužící jako síťový server, Stage a Gazebo jsou pak samotné simulační jednotky pro 2D respektive 3D simulaci. Nativním jazykem tohoto simulátoru je C/C++ a jeho architektura umožňuje rozsáhle distribuované výpočty, avšak v současné době pro něho není žádný vhodný model robota e-puck.

3.5.2 Pyrobot

Pyrobot² je open-source robotický simulátor pro platformu Linux podporující programovací jazyk Python. Jazyk Python je interpretovaný a nehodí se proto pro takové experimenty, které jsou výpočetně náročné. Mezi podporované roboty také zatím nepatří robot e-puck.

¹<http://playerstage.sourceforge.net/>

²<http://pyrorobotics.org/>

3.5.3 Microsoft Robotics Studio

Microsoft Robotics Studio³ je teprve nedávno uvolněný projekt robotického simulátoru založeného na platformě Windows s nativní podporou jazyka C# a framework .NET. V současnosti je projekt v neustálém vývoji s postupně přibývajícimi možnostmi pro stavbu prostředí a robotů, avšak většinou ve fázi beta verzí. Stejně jako v případě simulátoru Pyrobotics je nativní jazyk simulátoru interpretovaný a tedy nevhodný pro výpočetně náročné experimenty.

3.5.4 Webots

Webots⁴ je komerční robotický simulátor pro platformu Windows podporující jazyk C/C++ a Java. Ve svém provedení obsahuje nástroje jak pro stavbu a modelování světa tak pro stavbu robota. Modelování světa je realizováno prostřednictvím interaktivního editoru pracujícího s jazykem VRML. Obsahuje také věrný model robota e-puck. Kvůli dobrému modelu a podpoře robota e-puck jsme se rozhodli použít právě tento simulátor.

³<http://msdn.microsoft.com/robotics/>

⁴<http://www.cyberbotics.com/>

Kapitola 4

Implementace

Předpokládáme, že uživatel aplikace má mobilního robota s připraveným řízením, které očekává určitý počet hodnot z kamerového vstupu. Pro danou úlohu ovšem nezná správné složení kolon obrazových operátorů. V tomto případě může použít implementovanou aplikaci k nalezení buď správné kombinace nebo alespoň blízkou aproximaci této kombinace obrazových operátorů.

Aplikace by měla být dobře ovladatelná a pro uživatele srozumitelná. Uživatel by měl být schopen měnit co nejvíce parametrů genetického algoritmu tak, aby odpovídaly potřebám vývoje. Uživatel by také měl mít možnost sledovat statistiku vyvíjené populace. Také by měla umožňovat exportovat vyvinutou populaci do souboru, případně začít vývoj s importovanou populací jedinců.

Aplikace by měla být schopná dlouho trvající operace, které je možné paralelizovat, distribuovat mezi výpočetní jednotky. U této úlohy je nejdéle trvající úloha ohodnocení v simulátoru, aplikace umožňuje uživateli použít více simulátorů a urychlit tak výpočet.

Při vývoji aplikace byl použit framework wxWidgets 2.8.6¹ a vývojové prostředí Microsoft Visual Studio 2005². Aplikace pracuje pod operačním systémem Microsoft Windows XP SP2³.

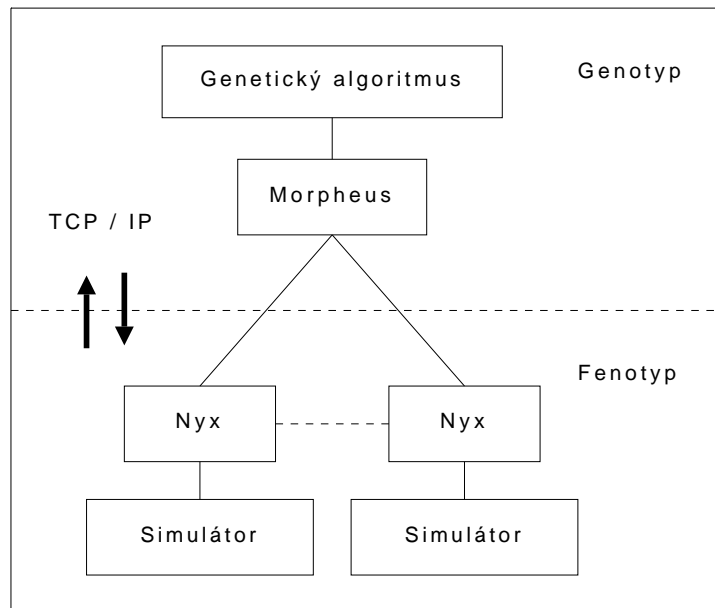
¹<http://www.wxwidgets.org/>

²<http://msdn2.microsoft.com/en-us/vstudio/default.aspx>

³<http://www.microsoft.com/windows/products/windowsxp/default.msp>

4.1 Moduly

Aplikace je rozdělena do dvou modulů, které zajišťují její chod. Hlavní modul reprezentuje implementaci genetického algoritmu a ovládání aplikace, druhý vedlejší modul zajišťuje spojení s testovacím prostředím a zpracováním obrazové informace. Není zcela nutné, aby testovacím prostředím byl simulátor Webots případně se nemusí jednat vůbec o robotický simulátor. Rozvržení modulů je zobrazeno na obrázku 4.1

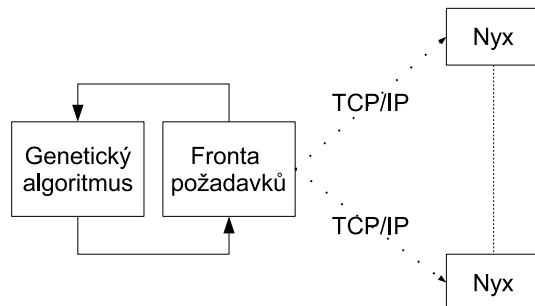


Obrázek 4.1: Schéma modulů

Jelikož časově náročná operace je v našem případě otestování robota v simulátoru, je možné připojit k hlavnímu modulu více modulů vedlejších a vyhodnocovat výsledky paralelně. Spojení je zajištěno přes protokol TCP/IP.

4.1.1 Morpheus

Modul Morpheus je hlavním modulem aplikace. V tomto modulu je možné nastavit všechny aspekty a chování aplikace, stejně tak připojení vedlejších modulů. Dále je z něj možné kontrolovat běh aplikace a sledovat veškeré dostupné statistické charakteristiky vyvíjené populace a s touto populací pracovat (export a import). Modul Morpheus implementuje genetický algoritmus. Propojení s vedlejšími moduly ilustruje obrázek 4.2.



Obrázek 4.2: Schéma fronty

4.1.2 Nyx

Modul Nyx je vedlejším modulem, doplňkem modulu Morpheus. Tento modul zajišťuje rozhraní mezi hlavním modulem a testovacím prostředím. Je schopen přijmout genotyp jedince a předat ho dále testovacímu prostředí skrze rozhraní ovladače. V tomto modulu je také možné otestovat jednotlivé implementované obrazové operátory a nastavit parametry síťového spojení mezi hlavním modulem a ovladačem.

4.1.3 Testovací prostředí

Testovací prostředí může reprezentovat libovolný simulátor nebo jiné prostředí, například jednotlivé soubory z adresáře emulující kamerový vstup. Komunikace a přenos genotypu k testovacímu prostředí je zajištěno obecným rozhraním ovladače. Vzhledem k různým proprietárním řešením simulátorů ve formě dll souborů potřebných k jejich spuštění je nutné, aby modul Nyx byl zkompileován s konkrétním jedním ovladačem.

4.1.4 Ovladač prostředí

Komunikaci mezi modelem Nyx a testovacím prostředím zajišťuje obecné rozhraní ovladače. Toto rozhraní umožňuje předat jedince k otestování v průběhu vývoje (*test*), zvolit znovunastavení (*reset*) prostředí anebo otestovat výsledného jedince (*long test*).

- Testování přijatého jedince v testovacím prostředí (TestIndividual)
- Trvalé testování přijatého jedince v testovacím prostředí (LongTestIndividual)

- Znovunastavení testovacího prostředí (Reset)

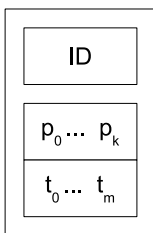
4.2 Repräsentace obrazu

Repräsentace obrazu je důležitá pro pozdější implementaci zpracování obrazu. Z možných variant jsme vybrali nativní repräsentaci použitého simulátoru Webots. Výstupem simulované kamery je 24 bitový obraz v RGB repräsentaci, který je realizován třemi bytovými poli velikosti $m \times n$. K těmto polím je znám pouze výchozí rozměr. Celá repräsentace obrazu v aplikaci je řešena obrazovým korpusem, který uchovává jak zdrojová pole s jednotlivými barevnými složkami obrazu, tak další informace o obrazu jako jsou rozměry, bitová hloubka a velikosti polí. S tímto korpusem pak pracují jednotlivé kolony a obrazové operátory. V případě použitého modelu e-puck je $m = 52$ a $n = 39$.

4.3 Obrazový operátor

Implementace obrazového operátoru je založena na obecném rozhraní, které posléze implementují jednotlivé obrazové operátory. To dává testovacímu prostředí možnost rozšiřitelnosti o další obrazové operátory bez nutnosti závažných zásahů do kódu. Toto rozhraní umožňuje spustit oprátor s daným obrazem a přečíst výsledek. Výsledek je, jak je naznačeno v návrhu řešení, reálné číslo z intervalu $\langle 0, 1 \rangle$ pokud se jedná o klasifikátor nebo transformace obrazu $\mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{l \times k}$, $m, n, l, k \in \mathbb{Z}^+$ v případě, že se jedná o filtr.

Toto abstraktní rozhraní definuje parametry $\{p_0, \dots, p_k\}$ a přechody mezi parametry $\{t_0, \dots, t_m\}$. Typ obrazového operátoru je určen unikátním číslem ID, rozdílným pro filtry a pro klasifikátory. Obrázek 4.3 ukazuje zjednodušené schéma obrazového operátoru.



Obrázek 4.3: Obrazový operátor

Při implementaci jsme vytvořili dvě skupiny operátorů, jednu pro filtry a druhou pro klasifikátory. Z těchto skupin genetický algoritmus vybírá nové filtry pro vyvíjené jedince.

4.4 Kolony

Kolony sestávají z jednotlivých obrazových operátorů. Jsou implementovány jako obalující třída, která přijme obraz ke zpracování, vytvoří si kopii a s tou poté pracuje. Výsledkem práce kolony je výsledek klasifikujícího obrazového operátoru. Kolona je vytvářena až v samotném testovacím prostředí, které je schopno transformovat přijatou reprezentaci jedince na kolony obrazových operátorů.

4.5 Genetický algoritmus

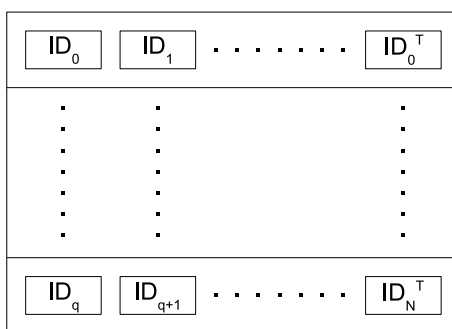
Základní verze algoritmu poskytuje dobré mechanismy pro práci s reprezentací jedinců pomocí bitového řetězce. Genetický algoritmus v základní verzi není pro některé úlohy dostatečný a je třeba změnit například reprezentaci a upravit genetické operátory tak, aby pracovaly s touto novou pozměněnou reprezentací. Případně je vhodné přidat další genetické operátory, které s danou strukturou pracují. V další části si popíšeme implementovanou reprezentaci jedince a genetické operátory. Algoritmus 3 ukazuje pozměněnou strukturu genetického algoritmu.

Algoritmus 3 Schéma implementovaného genetického algoritmu

- 1: Vytvoření populace nebo import již existující
 - 2: Inicializace modulů Nyx a spuštění výpočtu ohodnocení
 - 3: **repeat**
 - 4: Výběr nové populace ze staré
 - 5: Aplikace implementovaných genetických operátorů
 - 6: Výpočet ohodnocení jedinců
 - 7: Elitismus, stárnutí, protekce
 - 8: Nahrazení staré populace novou
 - 9: **until** Podmínka ukončení
-

4.5.1 Reprezentace jedince

Základní reprezentace jedince bývá obvykle řetězec bitů, u některých aplikací je vhodné tuto reprezentaci upravit. Aplikace v návrhu řešení pracuje se sekvencemi obrazových operátorů a také s jejich parametry. Udržovat a měnit strukturu v řetězci bitů, které reprezentují složitou strukturu, je obtížné. Proto jsme implementovali strukturu, která odděluje identifikaci operátorů a jeho parametrů. To nám umožňuje pracovat odděleně se strukturou jedince z hlediska typu obrazových operátorů a odděleně z hlediska jejich parametrů. Obrázek 4.4 ukazuje schéma reprezentace jedince, ID_i jsou jednotlivé filtry, ID_j^T jsou jednotlivé klasifikátory.



Obrázek 4.4: Schéma jedince

Fenotypem je hotový jedinec vytvořený na základě genotypu. Reprezentuje již vytvořené soustavy kolon pro testování v testovacím prostředí. V případě této konkrétní implementace se jedná o vytvoření kolon obrazových operátorů, které poté spolu s řízením a tvarem robota tvoří fenotyp.

4.5.2 Výběr

Výběr je důležitým faktorem pro správný vývoj populace. Pokud je tlak na výběr nejlepších jedinců příliš velký může genetický algoritmus velmi rychle skončit v lokálním maximu a dále nerozvíjet populaci. Pokud je tlak příliš malý, zůstává diverzita jedinců příliš velká a genetický algoritmus není efektivní. Proto je třeba mechanismu výběru věnovat velkou pozornost. Aplikace umožňuje použít tři mechanismy výběru, které jsou naznačeny v první kapitole.

Výběr turnajem

Výběr turnajem je výpočetně velmi efektivní způsob výběru populace. Výběr je realizován náhodným zvolením k jedinců z nichž ten s největším ohodnocením postoupí do nové populace. Hodnota k velmi závisí na velikosti populace, pokud je poměr příliš velký pak metoda vytváří příliš velký tlak na výběr dobrých jedinců. Při testech s populací o velikost 20 jedinců vedly hodnoty $k = 4$ a $k = 5$ k příliš rychlé konvergenci výpočtu k lokálnímu maximu. Naopak pro $k = 3$ nebo $k = 2$ byl tlak rovnoměrný. Pro konkrétní implementaci jsme vybrali $k = 3$, neboť tato hodnota poskytovala rovnoměrný tlak po celou dobu vývoje.

Randomizovaný výběr turnajem

Randomizovaný výběr turnajem je speciální variantou výběru turnajem pro $k = 2$, kde navíc dochází ke skutečnému turnaji mezi těmito dvěma jedinci. V případě, že náhodně zvolené číslo mezi $\langle 0, 1 \rangle$ je menší než zvolená konstanta t , postupuje horší jedinec, v opačném případě postupuje lepší jedinec. Konstanta t by měla být obvykle ostře menší než 0.5, aby výběr preferoval lepší jedince s větší pravděpodobností. V implementaci jsme použili $t = 0.3$.

Výběr ruletou

Třetím implementovaným způsobem výběru je výběr ruletou. Je implementován shodně s předlohou popsanou v první kapitole.

4.5.3 Druhy

Každý nově vzniklý jedinec je zařazen k určitému druhu podle tvaru a struktury obrazových operátorů v něm obsažených. Nový jedinec, který je zároveň i novým druhem by měl být po určitý počet evolučních kroků chráněn [17]. Aplikace je schopná detekovat vznik takovýchto nových druhů, neboť uchovává o každém jedinci záznam o použitých operátorech. Nový druh je tak určen sekvencí identifikačních čísel jednotlivých operátorů.

4.5.4 Elitismus

Elitismus je možné implementovat několika způsoby. Jedním z možných řešení je uschovat si n nejlepších jedinců, selekci provést s p jedinci a dále pracovat s $p - n$ jedinci, kde p je celková velikost populace. Po provedení evolučního kroku doplníme $p - n$ vyvinutých jedinců o n schovaných. Dalším

možným způsobem je schovat n nejlepších jedinců a dále vyvíjet populaci s p jedinci. Po provedení evolučního kroku nahradíme n nejhorších jedinců n nejlepšími schovanými jedinci. V naší implementaci používáme první způsob, jelikož druhů vyžaduje dodatečné ohodnocení jedinců v testovacím prostředí a doba potřebná k provedení jednoho evolučního kroku se tak významně prodlužuje (více například v [9]).

4.5.5 Stárnutí

Někteří jedinci mohou díky pravděpodobnostnímu výběru i pravděpodobnostní aplikaci operátorů zůstat v populaci velmi dlouho aniž by výrazně přispěli k diverzitě druhů a populace. Jedním ze způsobů řešení je prvek stárnutí jedinců.

Každý jedinec má implementován čítač s a každý nově vzniklý jedinec má tento čítač nastaven na hodnotu $s = S$, která je zadána uživatelem. Každý evoluční krok je tato hodnota snížena o 1. Tento čítač tedy udává počet evolučních kroků, po které jedinec nepodléhá operátoru stárnutí.

Pokud hodnota čítače klesne na 0, jedinec zestárl. Pokud je jeho ohodnocení menší než průměrné ohodnocení populace tak s pravděpodobností p_s bude nahrazen jedincem novým.

4.5.6 Protekce

V případě, že vznikne nový druh přírůstkem nebo odebráním obrazového operátoru, je tento jedinec velmi zranitelný. Jeho parametry nejsou optimálně vyladěny a jeho ohodnocení je velmi malé, ačkoliv může představovat perspektivní řešení. Každý nově detekovaný druh je proto chráněn protekcí. Protekce představuje počet evolučních kroků P , při kterých se jedinec neúčastní destruktivního vývoje (viz. kapitola o operátorech pracujících se strukturou). Každý evoluční krok je protekce snížena o jednu jednotku a jedinec se tak účastní pouze vývoje zaměřeného na rozvoj parametrů (viz. operátory pracující s parametry) [17].

4.5.7 Genetické operátory

Základní genetický algoritmus implementuje pouze dva genetické operátory - mutaci a křížení. Jelikož základní reprezentace jedince je řetězec bitů jsou

tyto dva operátory dostačující. V případě, že je reprezentace složitější, je vhodné upravit škálu genetických operátorů a rozšířit ji o genetické operátory, které jsou schopny s danou reprezentací pracovat [9].

Zvolená reprezentace dává genetickému algoritmu částečnou znalost o struktuře parametrů, kterou je možno využít při konstrukci křížení. Jelikož je reprezentace složená z operátorů a jejich parametrů, je vhodné zařadit i operátory, které pracují se strukturou kolony a s parametry obrazových operátorů.

Kvůli rozdílným charakteristikám filtrů a klasifikátorů aplikace umožňuje nastavit pravděpodobnosti zvlášť.

4.5.8 Operátory pracující s parametry

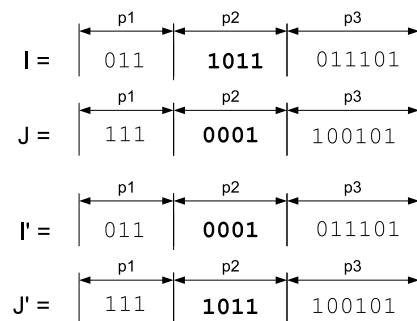
Genetické operátory pracující s parametry jsou principem shodné se základními operátory. Jedná se o mutaci a křížení, které pouze kvůli odlišení nazýváme parametrická mutace a parametrické křížení.

Parametrická mutace

Všechny parametry obrazových operátorů v jedinci tvoří bitový řetězec. Algoritmus parametrické mutace projde každý z nich a s určitou pravděpodobností p_m jej změni hodnotu bitu na opačnou. Mutace by neměla být hlavním prostředkem genetického algoritmu k nalezení optimálního řešení, měla by proto být používána pouze k navrácení ztracené diverzity populace [8].

Parametrické křížení

Původní křížení jak je definováno v základním provedení genetického algoritmu nepočítá se žádnou strukturou parametrů. To může vést k nesmyslným hodnotám a s velkou pravděpodobností nepovede ke zlepšení jedince. Implementovaná úprava bere v úvahu rozložení jednotlivých parametrů v řetězci a body pro křížení jsou vybírány na rozhraní parametrů. Schéma 4.5 zobrazuje příklad takového křížení. Křížení může probíhat pouze mezi stejnými obrazovými operátory.



Obrázek 4.5: Křížení se znalostí struktury

4.5.9 Operátory pracující se strukturou

Operátory pracující se strukturou přímo mění pořadí, druh a počet operátorů v koloně. Za použití těchto operátorů může dojít k rozvoji kolony a vzniku nových druhů.

Transformace

Kvůli smíšené reprezentaci jedince nemá parametrická mutace žádnou možnost jak změnit typ obrazového operátoru na jiný. K tomuto účelu slouží operátor transformace, který s určitou pravděpodobností změní obrazový operátor na jiný druh.

Vložení obrazového operátoru

K rozvoji kolon je nutné mít operátor, který je rozvíjí přidáváním nových obrazových operátorů. Tento operátor s určitou pravděpodobností vloží na náhodně zvolené místo v koloně nový operátor.

Vyjmutí obrazového operátoru

Operátor vyjmutí je opakem k operátoru vložení, z náhodné pozice odebere obrazový operátor.

Zvětšení jedince

Operátory vložení a vyjmutí umísťují nebo odebírají obrazové operátory z náhodných pozic v koloně. Oproti tomu operátor zvětšení jedince přidá obrazový operátor vždy na konec kolony mezi poslední filtr a klasifikátor.

Zmenšení jedince

Operátor zmenšení jedince je opakem ke zvětšení jedince, z poslední pozice odebere obrazový operátor.

Posunutí obrazového operátoru

Některá řešení kolon mohou být perspektivní vzhledem k vybraným obrazovým operátorům, avšak jejich pořadí může být nevhodné. K nápravě tohoto stavu slouží operátor posunutí, který s určitou pravděpodobností vybere operátor a posune jej o náhodný počet pozic dopředu.

Inverze obrazových operátorů

Inverze obrazových operátorů je zobecněním operátoru posunutí. Vybere náhodný podřetězec obrazových operátorů v koloně a provede inverzi jejich pořadí.

Prohození obrazových operátorů

Prohození je analogie k posunutí, rozdíl je, že v tomto případě jsou náhodně vybrány dva obrazové operátory v koloně a následně prohozeny.

4.5.10 Omezení velikosti kolony

Za určitých předpokladů, pokud se genetický algoritmus dostal například do lokálního maxima, může docházet k rozvoji velmi dlouhých kolon, které nemusí mít vhodné využití. V aplikaci je proto možné omezit velikost vyvíjené kolony. Jedinci, kteří dosáhnou určené velikosti jsou vyloučeni z aplikací genetických operátorů vložení a zvětšení.

4.5.11 Ohodnocení jedince

Ohodnocení jedince je pro genetický algoritmus klíčové. U některých typů aplikací je obtížné správně rozlišit potenciálně dobrá řešení od těch špatných. Například, má-li genetický algoritmus nalézt řešení ohodnocení logické formule, nelze ohodnotit kvalitu částečných řešení (více viz. [9]). Obdobný problém nastává u řešení vývoje kolon, bez možnosti ovlivnit řízení robota. Výběr obrazových operátorů může být vhodný, ale řízení robota může selhat. Ohodnocení zde proto hraje klíčovou roli. Testovaný jedinec by měl být odměněn již za drobné úspěchy a zároveň by neměl být penalizován za selhání řízení.

4.6 Navržené ovladače

Každý modul Nyx může nést jeden zkompileovaný ovladač prostředí. Pro tuto práci jsme zvolili simulátor Webots, pro který je připraven modul Nyx s příslušným ovladačem.

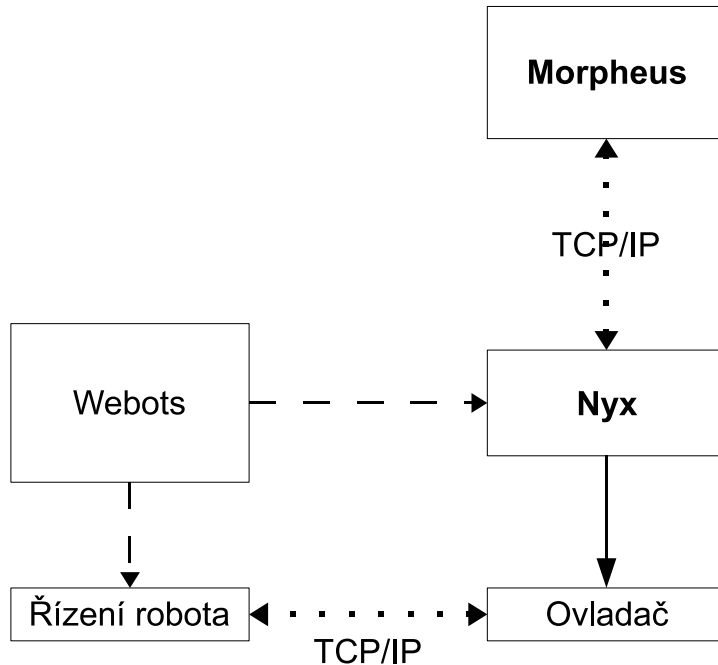
4.6.1 Emulace kamerového vstupu

Jednoduchý test je emulace kamerového vstupu prostřednictvím obrázků načtených z adresáře. Aplikace má k dispozici seznam obrázků a k nim očekávané ohodnocení, které zadává uživatel prostřednictvím grafického rozhraní. Tyto informace jsou poté uloženy do souboru a představují tak uniformní nezávislé prostředí. Aplikace přijímá obrázky pouze ve formátu BMP ve 24 bitové reprezentaci.

4.6.2 Simulátor Webots

Simulátor Webots má ke sledování simulace k dispozici tzv. supervisor. Supervisor je proces, který je schopen kontrolovat a měnit některé charakteristiky simulace, například umožňuje změnit pozici robota a hýbat s překážkami v simulovaném prostředí. Ovládání robota v simulátoru Webots pak zajišťuje tzv. controller. Supervisor a controller jsou simulátorem Webots spouštěny jako samostatné procesy.

Tohoto můžeme využít pro modul Nyx a nahradit tak tohoto supervisory naší aplikací. Integraci modulu Nyx spolu s Webots ukazuje schéma 4.6, přerušované čáry naznačují komunikaci přes protokol TCP/IP, plná čára komunikaci vnitřní a tečkovaná čára značí samostatný proces spouštěný aplikací, z níž šipka vychází.



Obrázek 4.6: Schéma propojení modulů se simulátorem Webots

Schéma 4 znázorňuje rozvržení těchto dvou procesů. Tohoto faktu využijeme a v simulátoru Webots nahradíme supervisor modulem Nyx a controller vlastním řízením, jak je znázorněno na obrázku 4.6.

Algoritmus 4 Schéma controlleru

- 1: Reset()
 - 2: **repeat**
 - 3: Run()
 - 4: **until** Robot není vypnut
-

Modul Morpheus, implementující genetický algoritmus, komunikuje s modulem Nyx prostřednictvím protokolu TCP/IP. Tento modul Nyx poté přes obecně navržené rozhraní ovladače komunikuje se systémem řízení, controllerem, taktéž pomocí TCP/IP. Detaily použitého protokolu jsou uvedeny v příloze B.

4.7 Statistické charakteristiky populace

Jelikož je otázka ukončení genetického algoritmu obtížná a není jasně stanoveno kdy má být vývoj zastaven, je vhodné mít k dispozici statistické veličiny, které vyjadřují vlastnosti populace. Můžeme například sledovat průměrné a nejlepší ohodnocení a v případě, že je dostatečně vysoké anebo dosáhlo maxima můžeme vývoj ukončit.

V aplikaci jsou mimo grafu nejhoršího, nejlepšího a průměrného ohodnocení k dispozici ještě tři další charakteristiky, které si dále popíšeme.

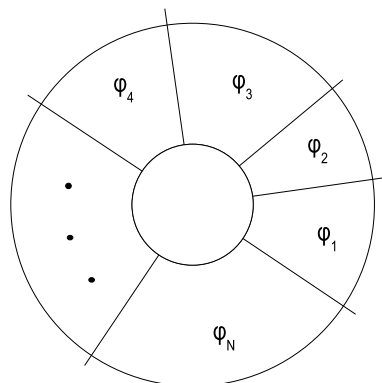
4.7.1 Hustota ohodnocení

Hustota ohodnocení sleduje jednotlivé hladiny dosaženého ohodnocení. Je reprezentována barevným sloupcem, který je rozdělen na k horizontálních hladin, přičemž dolní část odpovídá minimálnímu ohodnocení a horní část maximálnímu ohodnocení, hladiny mezi těmito dvěma pak zastupují ostatní hodnoty ohodnocení. Ohodnocení každého jedince je transformováno do intervalu $\langle 0, k \rangle$.

Barva každé z k hladin je určena počtem n jedinců, jejichž ohodnocení spadá do vyšší nebo stejné hladiny. Barva hladiny je určena jako trojice $(255, c, c)$, kde $c = 255 - n \frac{255}{N}$ a N je počet jedinců v populaci. Je tedy zřejmé, že barva hladiny s vyšším počtem takovýchto jedinců jsou tmavší a hladiny s nižším počtem naopak světlejší. V naší implementaci jsme použili $k = 395$.

4.7.2 Rozložení ohodnocení

Rozložení ohodnocení sleduje poměr mezi ohodnoceními celé populace. Je vhodné ke sledování, zda nějaký jedinec dosáhl výrazně vyššího ohodnocení. Implementace je realizována podobně jako ruletové kolo, jednotlivá ohodnocení jsou transformována na velikost úhlu $\varphi \in \langle 0, 360 \rangle$. Ukázka rozložení je na obrázku 4.7.



Obrázek 4.7: Rozložení ohodnocení

4.7.3 Histogramy obrazových operátorů

Další sledovanou charakteristikou může být množství jednotlivých typů obrazových operátorů. K tomu vytvoříme histogram zobrazující všechny operátory v populaci, zvlášť pro filtry a zvlášť pro klasifikátory. Více o histogramech jako o statistické veličině je uvedeno v příloze A o použitých operátorech.

4.8 Řízení robota

Při implementaci řízení robota pro tuto aplikaci vyvstává problém jak oddělit kvalitu samotného řízení od kvality výpočtu genetického algoritmu. Jelikož genetický algoritmus řízení nijak nemění a má možnost ovlivnit pouze sestavu operátorů, je nutné zvolit řízení jednoduché. Velmi dobře řešené řízení může kompenzovat nedostatky vyvíjených kolon, a znesnadnit tak jejich správný vývoj, naproti tomu špatně řešené řízení může způsobit, že dobře vyvinuté kvalitní kolony dostanou nízké ohodnocení. Konkrétní použité implementace řízení uvádíme v jednotlivých kapitolách o experimentech.

Kapitola 5

Experimenty

Tato kapitola se věnuje experimentům provedeným s výše popsanou implementací. Jelikož aplikace může pracovat prostřednictvím ovladačů s různými druhy prostředí navrhli jsme ovladač pro simulátor Webots a dále pak ovladač pro emulaci kamerového vstupu, oba implementované ovladače si dále popíšeme.

Veškeré zde uvedené testy byly provedeny na počítači s CPU Core2 Quad (2.4 GHz) se 3 GB operační paměti a grafickou kartou s čipem nVidia 7600 s 512 MB video paměti a 32 bitovým operačním systémem Windows XP Professional (SP2). Systémy řízení byly zkompileovány pomocí Microsoft Visual Studio 2005. Experimenty v simulátoru jsme provedli s Webots PRO (5.7.3).

5.1 Emulace kamerového vstupu

Emulace kamerového vstupu umožňuje částečně nahradit prostředí simlátoru, vstup pro kolony filtrů je reprezentován sekvencí 24 bitových obrázků ve formátu BMP umístěných v adresáři. Aplikace umožňuje nastavit počet kolon k , které se mají vstupem zabývat a také obsahuje rozhraní pro nastavení vstupních dat. Vstupními daty jsou posloupnosti N jednotlivých obrázků p_1, p_2, \dots, p_N . Dále posloupnost očekávaných hodnot $o_1^{(i)}, o_2^{(i)}, \dots, o_k^{(i)}$ na výstupu kolon, pro jednotlivé obrázky p_i . Dalším vstupem je ohodnocení pro každý obrázek $f_1^{(i)}, f_2^{(i)}, \dots, f_k^{(i)}$. Kde $f_j^{(i)}$ je maximální počet bodů, které dostane jedinec, pokud na obrázek p_i , kolona j dá výstup $o_j^{(i)}$.

Jedinec obdrží posloupnost ohodnocení $\Phi_1^{(i)}, \Phi_2^{(i)}, \dots, \Phi_k^{(i)}$, přičemž ohodnocení $\Phi_j^{(i)}$ spočteme podle (5.1), kde předpokládáme, že $o_j^{(i)}$ je očekávaná

hodnota j -té kolony u obrázku p_i a $r_j^{(i)}$ je skutečný výstup kolony j na obrázku p_i .

$$\Phi_j^{(i)} = \begin{cases} f_j^{(i)}, & \text{je-li } o_j^{(i)} = r_j^{(i)} \\ -\frac{f_j^{(i)}}{2}, & \text{je-li } o_j^{(i)} < r_j^{(i)} \\ -\frac{f_j^{(i)}}{3}, & \text{je-li } o_j^{(i)} > r_j^{(i)} \end{cases} \quad (5.1)$$

Výsledné ohodnocení každého jedince i je pak prostým součtem podle vzorce (5.2). Je zřejmé, že tento způsob ohodnocení není vhodný pro každou úlohu a pro některé konkrétní úlohy je nutné jej programově změnit.

$$\sum_N^{i=1} \sum_k^{j=1} \Phi_j^{(i)} \quad (5.2)$$

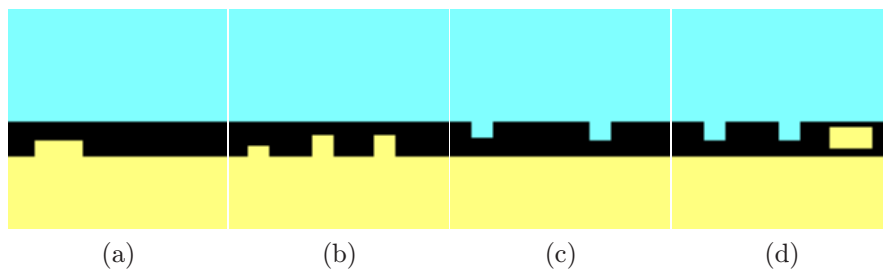
5.1.1 Experimenty s emulací kamerového vstupu

Postupně jsme provedli pět testů s emulací kamerového vstupu. Na těchto testech chceme demonstrovat funkčnost navrženého algoritmu, neboť v případě emulace řešení úlohy nijak nezávisí na použitém řízení mobilního robota. Vývoj genetického algoritmu jsme ukončili manuálně v případech, kdy řešení bylo po delší čas ustálené, dosáhlo maxima anebo v případech, kdy v populaci převládaly podobné obrazové operátory. Avšak aplikace umožňuje nastavit předem daný počet evolučních kroků, které mají být provedeny.

Test 1

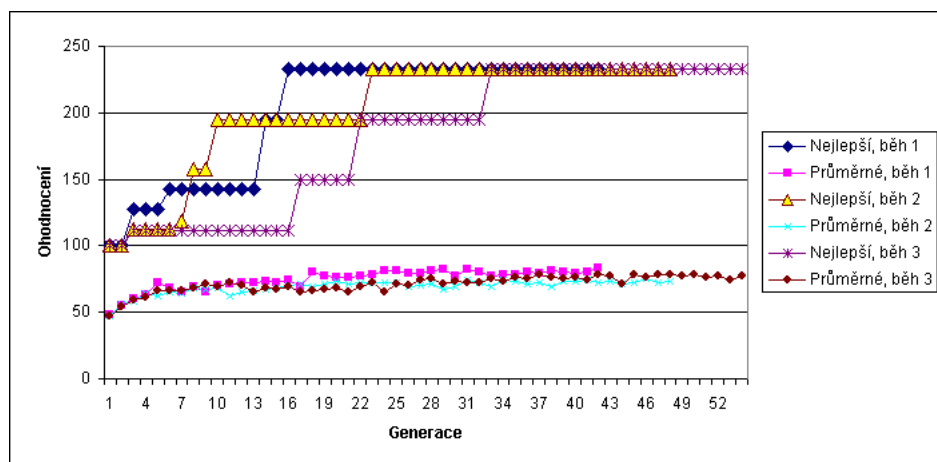
Obrázky v prvním testu jsou obrázky tak jak je vidí robot, je na nich znázorněna podlaha (okrová barva), zeď (černá barva) a nebe (modrá barva). Obrázků je celkem 40 a rozdělili jsme je do dvou skupin po 20 obrázcích. První skupina jsou obrázky, kde je zeď ve spodní části vykrojená, tato situace je znázorněna na obrázcích 5.1a a 5.1b. Druhá skupina má vykrojené horní části nebo obsahuje různé dodatečné obrazce, jak ukazují obrázky 5.1c a 5.1d. Na první sadu obrázků jsme z kolony očekávali výstup 1 a na obrázky z druhé skupiny výstup 0. Za úspěšné rozpoznání obrázku z první skupiny jsme stanovili ohodnocení 7 bodů a za úspěšné rozpoznání obrázku z druhé

skupiny ohodnocení 5 bodů. Kompletní sada obrázků použitá při tomto testu je k nalezení na příloženém CD.



Obrázek 5.1: Ukázka vstupních dat pro první test

Chtěli jsme zkonstruovat jednu kolonu, která bude detekovat obrázky z první skupiny, tedy takové, které mají spodní vykrojení. Test jsme spustili celkem třikrát s různými parametry genetického algoritmu, abychom ověřili robustnost výpočtu. Parametry jsou k nalezení v příloze G.1. Při prvním běhu došlo za 10 minut k vývoji 42 generací, při druhém za 11 minut ke 48 generacím a při třetím za 16 minut k 64 generacím. Průběh těchto výpočtů je znázorněn v grafu 5.2.



Obrázek 5.2: Průběh vývoje při emulaci kamery v prvním testu

V průběhu všech tří běhů došlo k vývoji shodného nejlepšího jedince složeného z filtru řádku a klasifikátoru prahování, tyto nejlepší jedinci jsou vypsáni v tabulce 5.3. Filtr z celého vstupního obrazu ponechá pouze jeden řádek, v němž součet hodnot pixelů musí překročit práh. Úspěšnost, tedy schopnost

rozpoznání obrázků podle požadovaných očekávaných jedinců, těchto jedinců na testovací množině je 97.5%.

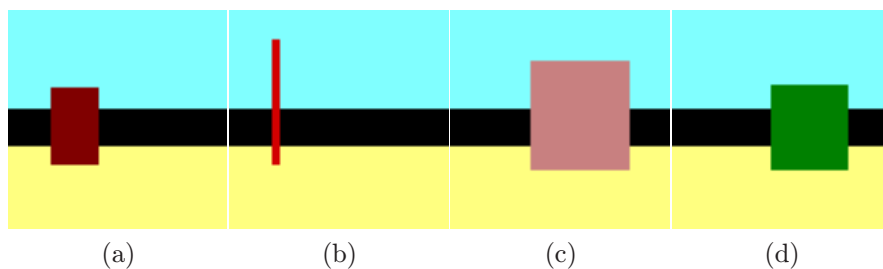
V průběhu všech běhů došlo k vývoji podobných řešení – kombinace řádkového filtru (Row) a prahování (Threshold). Parametr 40 u řádkového filtru po přepočtu ($\frac{n}{62}40$, kde $n = 42$ a vyjadřuje výšku obrázku) znamená použití řádku 28. Poté rozhodne klasifikátor prahování.

Nejlepší jedinec z běhu 1, 232.5 bodů		
Kolona 1	Row	Threshold
Parametry	101000 (40)	11011000 (216)
Nejlepší jedinec z běhu 2, 232.5 bodů		
Kolona 1	Row	Threshold
Parametry	101000 (40)	11010110 (214)
Nejlepší jedinec z běhu 3, 232.5 bodů		
Kolona 1	Row	Threshold
Parametry	101000 (40)	10100000 (160)

Obrázek 5.3: Výslední jedinci testu 1

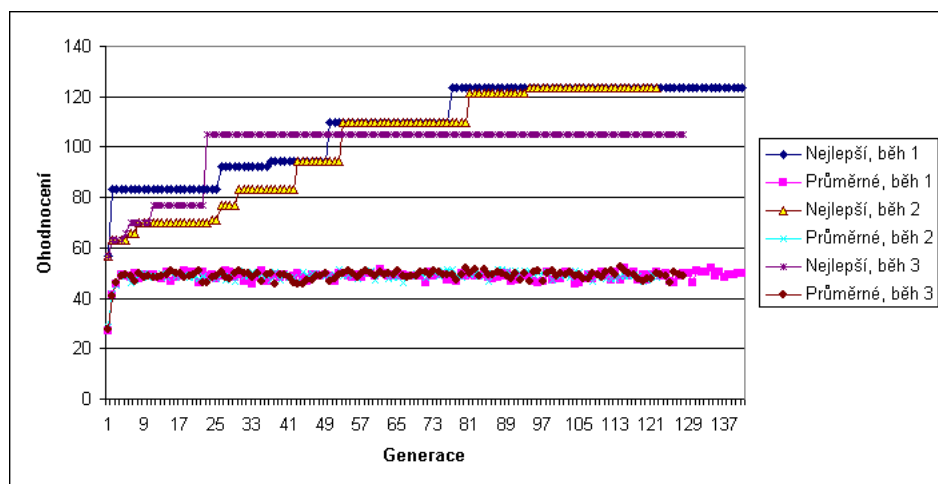
Test 2

Při druhém testu jsme chtěli overit, zda se genetický algoritmus dokáže naučit rozpoznávat různé velké objekty v odstínech tmavě červené barvy. Připravili jsme 30 obrázků, ve kterých jsme ponechali podlahu, zeď a nebe v barvách shodných s předchozím experimentem. Těchto 30 obrázků jsme opět rozdělili na dvě skupiny. První skupina obsahuje obrázky s různě velkými čtverci vyvedených v tmavě červené barvě, dva příklady ukazuje obrázek 5.4a a 5.4b. Druhá skupina obrázků navíc obsahuje různé jiné objekty, jak ukazuje obrázek 5.4d, a také některé objekty vyvedené ve světle červené barvě, jak ukazuje obrázek 5.4c. Na první sadu obrázků jsme z kolony očekávali výstup 1 a na obrázky z druhé skupiny výstup 0. Za úspěšné rozpoznání obrázku jsme u obou skupin stanovili ohodnocení 5 bodů. Kompletní sada obrázků použitá při tomto testu je k nalezení na příloženém CD.



Obrázek 5.4: Ukázka vstupních dat pro druhý test

Test jsme spustili opět celkem třikrát s pozměněnými parametry, které jsou k nalezení v příloze G.2. Při prvním běhu došlo za 28 minut k vývoji 141 generací, při druhém za 24 minut ke 122 generacím a při třetím za 26 minut k 128 generacím. Průběh těchto výpočtů je znázorněn v grafu 5.5. První dva běhy vykazovaly velmi dobré výsledky, ovšem z průběhu třetího běhu je zřejmé, že výpočet zůstal v lokálním maximu.



Obrázek 5.5: Průběh vývoje při emulaci kamery ve druhém testu

Oproti předchozímu testu nebylo řešení ustálené, v každém z běhů došlo u nejlepších jedinců k vývoji jiného filtru a u třetího běhu i k vývoji jiného klasifikátoru. Nejlepší jedinci jsou vypsáni v tabulce 5.6. Úspěšnost řešení je v případě prvních dvou běhů 86% u třetího pouze 80%.

V prvním běhu došlo k vývoji zajímavé kolony, která obsahuje filtr vyřiznutí sloupce (Column cut), prahy 42 a 9 po přepočtu odpovídají sloupci

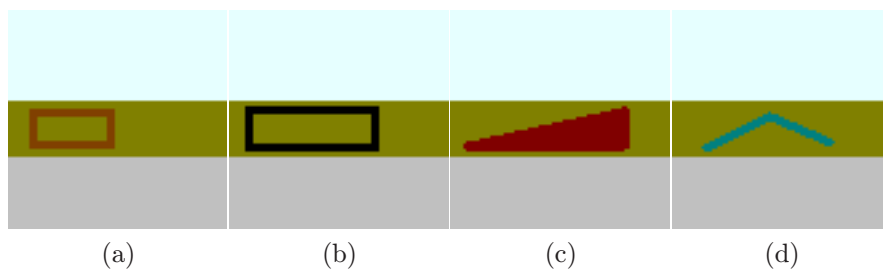
mezi řádkem 8 a řádkem 36. Tato kolona je zakončena histogramovým klasifikátorem (Histogram), který má práh nastaven na 20 (podle parametru 1) a počítá hodnoty v histogramu zelené složky mezi 0 a 8 (po přepočtu parametrů 0 a 1). Ve druhé koloně pak jedinec, jehož první filtr (Box cut) vyřízne čtverec o straně 39 pixelů. Následován opět histogramovým klasifikátorem (Histogram). Ve třetím běhu pak filtr s pokročilým prahovým segmentováním (Adv. threshold segmentation) s prahy 32 a 76, následován prahovým klasifikátorem (Threshold).

Nejlepší jedinec z běhu 1, 123.3 bodů		
Kolona 1 Parametry	Column Cut 101010 001001 (42, 9)	Histogram 001 01 00001 00000 (1, 1, 1, 0)
Nejlepší jedinec z běhu 2, 123.3 bodů		
Kolona 1 Parametry	Box Cut 111010 (58)	Histogram 001 01 00000 01010 (1, 1, 0, 10)
Nejlepší jedinec z běhu 3, 105 bodů		
Kolona 1 Parametry	Adv. threshold segmentation 00100000 01001100 (32, 76)	Threshold 00000010 (2)

Obrázek 5.6: Výslední jedinci testu 2

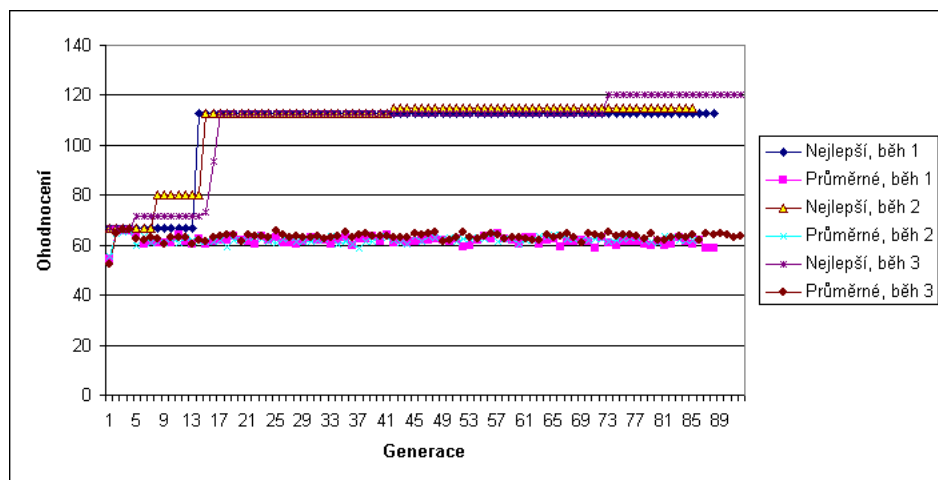
Test 3

Při třetím testu jsme chtěli ověřit, zda genetický algoritmus dokáže sestavit kolony rozpoznávající obrazce. Sadu obrázků jsme rozdělili opět na dvě skupiny. První skupina o 20 obrázcích obsahovala různě velké obdélníky nakreslené na zdi, tyto obdélníky znázorňují obrázky 5.7a a 5.7b. Druhá skupina, také o 20 obrázcích, má na zdech nakresleny různé jiné obrazce, vesměs trojúhelníky 5.7c a různě barevné čáry 5.7d. Opět u první skupiny očekáváme výstup 1 a u druhé 0, přičemž ohodnocení jsme stanovili na 5 bodů v obou skupinách. Kompletní sada obrázků použitá při tomto testu je k nalezení na příloženém CD.



Obrázek 5.7: Ukázka vstupních dat pro třetí test

Test jsme spustili opět třikrát s parametry, které jsou k nalezení v příloze G.3. Při prvním běhu došlo za 20 minut k vývoji 88 generací, při druhém za 27 minut k 85 generacím a při třetím běhu za 23 minut 91 generací. Průběh těchto výpočtů je znázorněn v grafu 5.8.



Obrázek 5.8: Průběh vývoje při emulaci kamery ve třetím testu

Oproti předchozím testům se ukázalo, že toto je již náročná úloha, která vedla jen k nižší úspěšnosti řešení. Nejlepší jedinec z prvního běhu dosahoval při řešení úlohy úspěšnosti jen 67.5%, zatímco jedinci ze dvou dalších běhů dosahovali 70%. Nejlepší jedinci jsou vypsáni v tabulce 5.9. Dosažená řešení se značným způsobem liší a ani v případě několika běhů algoritmu nekonverguje řešení k jednomu tvaru kolony jako v předchozích případech.

Při prvním běhu došlo k vývoji jedince, který používá filtr Gamma korekce (Gamma) a klasifikátor detekující červenou barvu pomocí histogramu

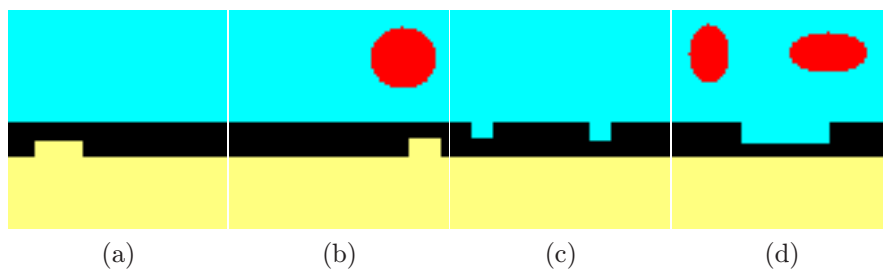
(Red color) s prahem 17 ($1\frac{mn}{100}$). Ve druhém pak k vývoji Kirschova konvolučního operátoru a histogramového klasifikátoru, který pracuje s červenou složkou obrazu s prahem 10 ($10 + 0 \times 10$). Třetí vyvinutý jedinec používá ekvalizaci histogramu (Histogram equalization) a jako klasifikátor opět histogram pracující s červenou složkou obrazu.

Nejlepší jedinec z běhu 1, 112.5 bodů		
Kolona 1	Gamma	Red color
Parametry	-	00000001 (1)
Nejlepší jedinec z běhu 2, 115.0 bodů		
Kolona 1	Kirsch	Histogram
Parametry	-	000 00 01000 00010 (0, 0, 8, 2)
Nejlepší jedinec z běhu 3, 120.0 bodů		
Kolona 1	Histogram equalization	Histogram
Parametry	-	000 00 11111 11101 (0, 0, 31, 29)

Obrázek 5.9: Výslední jedinci testu 3

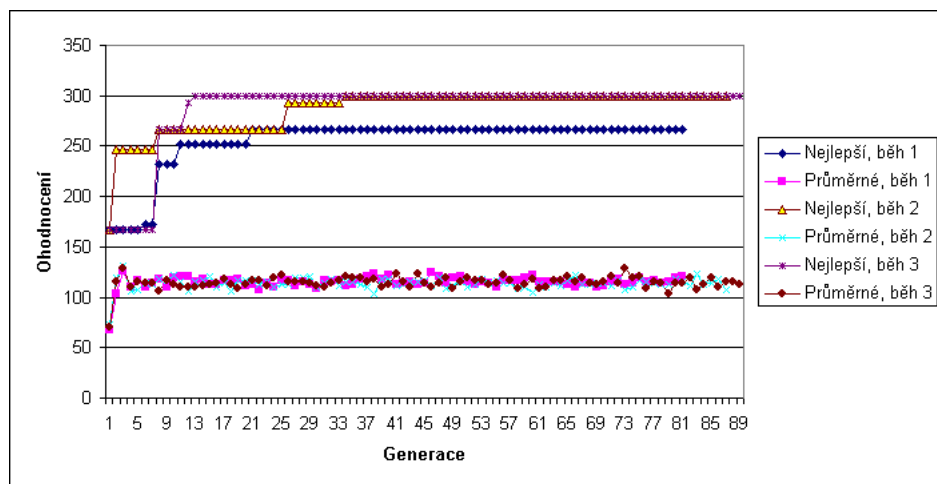
Test 4

Při čtvrtém testu jsme chtěli ověřit, že je genetický algoritmus schopen sestavit i kombinaci několika kolon. Použili jsme stejnou sadu obrázků jako ve druhém testu, ale do každého druhého obrázku jsme přidali červené tečky v horní části a také jsme vytvořili tmavší reprezentaci modrého nebe. Tyto tečky jsou znázorněny na obrázcích 5.10b a 5.10d. Očekáváme tedy sestavení dvou kolon. První, která bude řešit stejný úkol jako ve druhém testu a druhou, která bude detekovat červené tečky v obraze. U první kolony očekáváme u prvních 20 obrázků výstup 1 a u druhé skupiny obrázků výstup 0. U druhé kolony očekáváme výstup 0 u každého obrázku s červenými tečkami a výstup 0 u obrázku bez červených teček. Ve všech případech je ohodnocení stanoveno na 5 bodů. Kompletní sada obrázků použitá v tomto testu je k nalezení na příloženém CD.



Obrázek 5.10: Ukázka vstupních dat pro čtvrtý test

Test jsme pustili opět třikrát, s parametry G.4. Při prvním běhu došlo za 38 minut k vývoji 81 generací, při druhém za 41 minut ke 87 generacím a při třetím za 38 minut k 89 generacím. Průběh těchto výpočtů je znázorněn v grafu 5.11.



Obrázek 5.11: Průběh vývoje při emulaci kamery ve čtvrtém testu

Při vývoji došlo k vyvinutí perspektivních jedinců, nejlepší jsou vypsány v tabulce 5.12. Kolony v tomto případě nepracují příliš dobře, neboť první kolona zcela selhává v detekci v 50% případů. Při výpočtu genetický algoritmus zůstal v lokálním maximum, které zřejmě způsobil systém ohodnocení, který dostatečným způsobem nepenalizoval neúspěch obou kolon jako celku. Ačkoliv je úspěšnost jedinců shodná, ohodnocení se liší. To je dáno systémem ohodnocení, který v případě, že je hodnota očekávaného výsledku menší než výsledek práce kolony, odečítá méně bodů. U jedinců z druhého a třetího běhu byl v několika případech výsledek blízký hodnotě 0.9 zatímco očeká-

vaný výsledek 1.0.

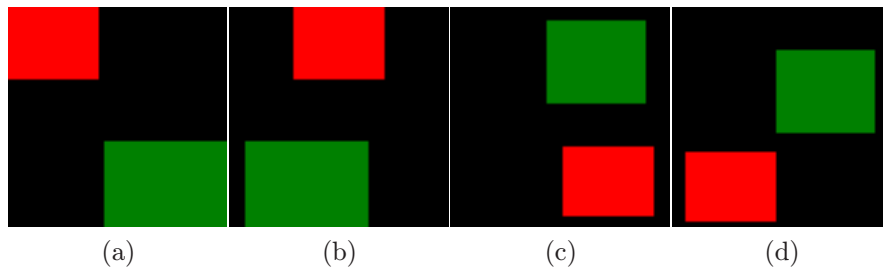
Nyní si popíšeme jednoho vybraného vyvinutého jedince. Ve druhém běhu došlo k vývoji jedince, který v první koloně používá klasifikátor vnímání černé (Sensing black) s parametrem 2, čemuž odpovídají vnitřní prahy 9 a 15. Klasifikátor tedy uvažuje pouze objekty, které jsou ve velikosti 9 až 15 pixelů. Druhá kolona pak filtr řádku (Row), který ponechá řádek 9. Na tento je následně použit klasifikátor červené barvy s parametrem 166.

Nejlepší jedinec z běhu 1, 266 bodů		
Kolona 1	-	Gap red
Parametry	-	01 (1)
Kolona 2	Histogram equalizer	Red color
Parametry	-	11011111 (223)
Nejlepší jedinec z běhu 2, 300 bodů		
Kolona 1	-	Sensing black
Parametry	-	10 (2)
Kolona 2	Row	Red color
Parametry	001101 (13)	10100110 (166)
Nejlepší jedinec z běhu 3, 300 bodů		
Kolona 1	Greyscale	Sensing black
Parametry	-	10 (2)
Kolona 2	Multiple threshold segmentation	Green color
Parametry	01001100 01011010 01101111 01100010 (76, 90, 111, 98)	00000110 (6)

Obrázek 5.12: Výslední jedinci testu 4

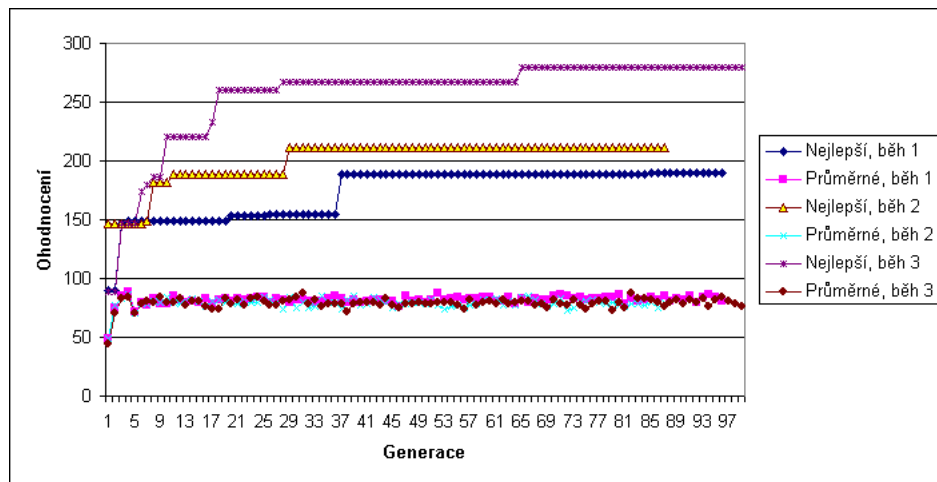
Test 5

V pátém testu jsme chtěli vyvinout dvě kolony, které rozlišují obrázky s červenými a zelenými čtverci. Sadu 30 obrázků jsme rozdělili na dvě skupiny po 15. První skupina obsahuje různě velké zelené a červené čtverce, přičemž první skupina obrázků má červený čtverec vždy nahoře a zelený čtverec vždy dole. Druhá skupina má rozložení čtverců právě naopak, zelený čtverec nahoře a červený čtverec dole. U první skupiny obrázků očekáváme výstup 1 u obou kolon a u druhé skupiny obrázků očekáváme výstup 0 u obou kolon. Zástupci první skupiny jsou obrázky 5.13a a 5.13b. Zástupci druhé skupiny jsou obrázky 5.13c a 5.13d. Ohodnocení jsme stanovili na 5 v obou případech. Kompletní sada obrázků použitá při tomto testu je k nalezení na příloženém CD.



Obrázek 5.13: Ukázka vstupních dat pro pátý test

Test jsme pustili opět třikrát, s parametry G.5. Při prvním běhu došlo za 45 minut k vývoji 96 generací, při druhém za 37 minut k 87 generacím a při třetím za 42 minut k 99 generacím. Průběh těchto výpočtů je znázorněn v grafu 5.14.



Obrázek 5.14: Průběh vývoje při emulaci kamery v pátém testu

Při prvních dvou bězích došlo k vývoji podobného řešení, avšak toto řešení dosahuje stejně jako v případě předchozím dobrých výsledků pro každou kolonu zvlášť, nejlepší jedinec v prvním běhu má úspěšnost 75%, nejlepší ve druhém 80%. Ale ve třetím běhu došlo k vývoji velmi dobrého jedince, který úlohu velmi dobře řeší, jeho úspěšnost při rozpoznávání je 95%. Nejlepší jedinci jsou vypsaní v tabulce 5.15.

Dále si popíšeme nejlepšího jedince ze všech tří běhů. Při třetím běhu došlo k vývoji velmi dobrého jedince. První kolona obsahuje řádkový filtr (Row), který ponechá pouze řádek 9 ($13 \times \frac{42}{62}$), následně je použit klasifikátor červené barvy s parametrem 147. Druhá kolona pak má stejné operátory, přičemž filtr ponechá řádek 11 ($16 \times \frac{42}{62}$) a klasifikátor červené barvy má parametr 246.

Nejlepší jedinec z běhu 1, 190 bodů		
Kolona 1 Parametry	Column Cut 111100 011101 (60, 29)	Threshold 11111011 (251)
Kolona 2 Parametry	Row 010001 (17)	Red color 00111100 (60)
Nejlepší jedinec z běhu 2, 210 bodů		
Kolona 1 Parametry	Column Cut 001111 000001 (15, 1)	Threshold 01110110 (118)
Kolona 2 Parametry	Row Cut 010011 000110 (19, 6)	Red color 00000001 (1)
Nejlepší jedinec z běhu 3, 280.00 bodů		
Kolona 1 Parametry	Row 001101 (13)	Red color 10010011 (147)
Kolona 2 Parametry	Row 010000 (16)	Red color 11110110 (246)

Obrázek 5.15: Výslední jedinci testu 5

Provedené testy ukazují, že genetický algoritmus je schopen sestavit kolony pro emulaci kamerového vstupu. v další části se budeme věnovat experimentům s mobilním robotem v simulátoru Webots.

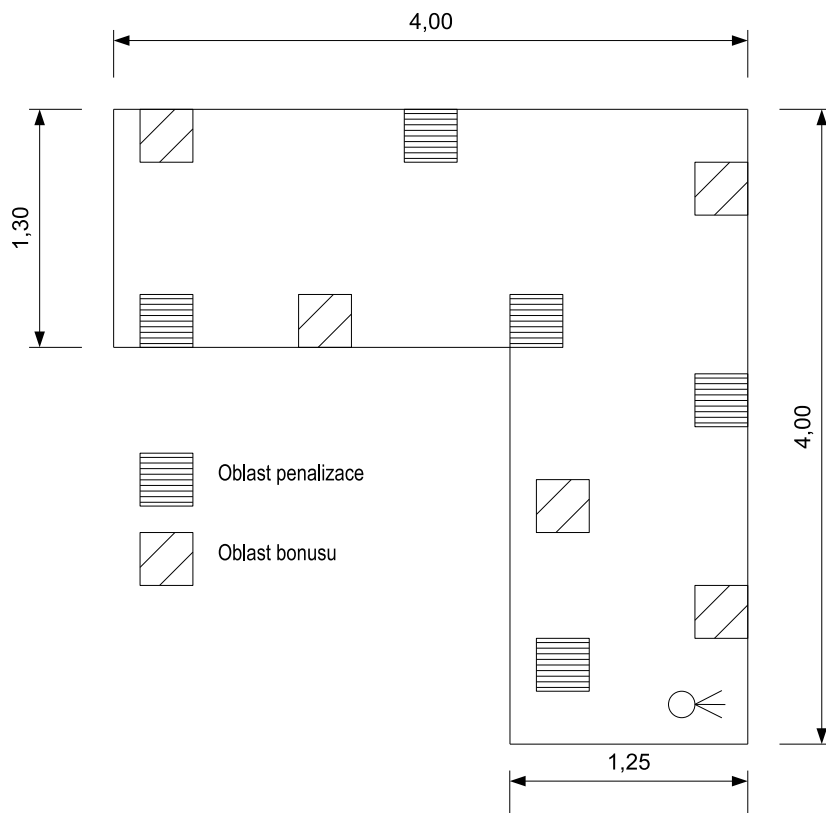
5.2 Simulátor Webots

V simulátoru Webots jsme navrhli dvě prostředí v nichž jsme postupně měnili robotův úkol. Tato prostředí a úkoly jsme konstruovali s důrazem na potřebu co nejjednoduššího řízení robota, jelikož jej genetický algoritmus nijak neovlivňuje. Narozdíl od předchozí emulace je ohodnocení robota různé pro každý test, systém ohodnocení a konkrétní úkoly popisujeme dále. Rozměry popisovaných prostředí jsou udávány v jednotkách, které používá simulátor Webots a jsou znázorněny u schémat jednotlivých prostředí.

Prostředí první

Prvním z prostředí je scéna tvaru písmene L, ve kterém má robot za úkol sesbírat co nejvíce bonusových bodů a co nejméně záporných bodů. Schéma této scény je na obrázku 5.16 a místa, kde je robot oceněn bonusovými body jsou označena vertikálně šrafovanými čtverci a místa se zápornými body jsou označena vodorovně šrafovanými čtverci. Přesný model prostředí je možné nalézt na příloženém CD.

V momentě, kdy se robot nachází v některém ze čtverců je odměněn nebo penalizován, je důležité říci, že průjezd každým ze čtverců je počítán pouze jedenkrát.

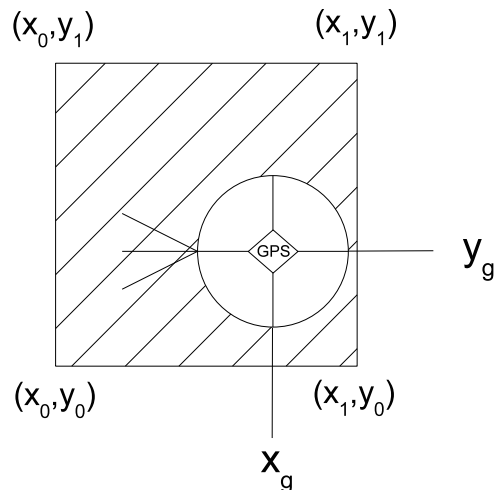


Obrázek 5.16: Prostředí první v simulátoru Webots

Robot sám neví, ve kterých oblastech získává bonusové body, ani ve kterých body ztrácí. Ma ale možnost tyto oblasti rozpoznat, pomocí vizuálního vjemu, kterými jsou tyto oblasti reprezentovány. Pro příklad si můžeme představit, že oblasti bonusových čtverců jsou určeny virtuálními zelenými krychlemi a oblasti čtverců s penalizací virtuálními krychlemi červenými. Robot by pak měl například rozlišovat zelenou barvu v obraze a vyhledávat pouze krychle zelené. Robotovi jsme postupně v jednotlivých experimentech měnili reprezentaci těchto čtverců, což popíšeme dále v části o konkrétních experimentech.

Pro počítání polohy robota využíváme připojený simulovaný modul GPS¹, který reálný robot e-puck nemá. Používáme jej pouze pro potřeby počítání ohodnocení. Tento modul je umístěn uprostřed robota. Robot vjel do oblasti s bonusem nebo penalizací za předpokladu, že jeho modul GPS udává souřadnice uvnitř oblasti. Toto ilustruje schéma 5.17.

¹<http://www.cyberbotics.com/cdrom/common/doc/webots/reference/section3.20.html>



Obrázek 5.17: Robot v bonusové oblasti

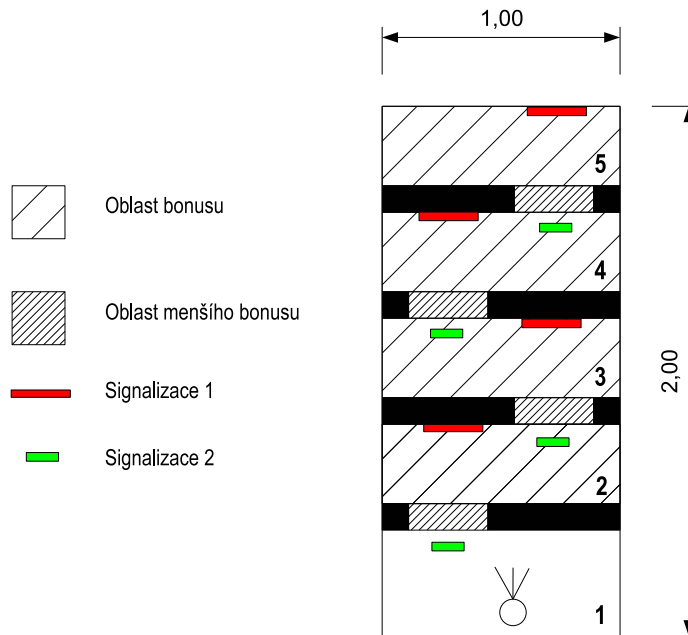
Prostředí druhé

Druhé prostředí je obdélníkového tvaru a je rozděleno čtyřmi přepážkami. Tyto přepážky rozdělují prostředí na pět oblastí vyznačených číslicemi 1 až 5 na obrázku 5.18. Cílem robota je dojet z první oblasti do poslední cílové páté oblasti. První z oblastí je výchozí, na té robot začíná. Každá z přepážek je na určitém krajním místě přerušena výřezem a prostor, který tak vniká je dostatečně velký, aby robot mohl projet z jedné oblasti do druhé. Tyto výřezy jsou označeny světelnou signalizací, na obrázku *Signalizace 2*. Tato signalizace má tvar malého kvádrů, který je robot schopen detekovat. Aby měl robot práci ztíženu, je tato signalizace umístěna před výřez. Tato pozice je zvolena tak, že robot nacházející se u signalizace nevidí skrz další výřez signalizaci pro další přepážku. Tedy *Signalizace 2* slouží k tomu, aby se robot byl schopen dostat před výřez, pokud se tak stane, *Signalizace 2* zhasne. Robot je za nalezení výřezu ohodnocen bonusovými body.

Pokud je robot před výřezem má šanci detekovat správný směr dalšího pohybu do oblasti pomocí *Signalizace 1*. Ta je reprezentována velkým obdélníkem na protější přepážce, tuto situaci ilustruje obrázek 5.18. Pokud se robot dostane do další oblasti, pak i *Signalizace 2* v této oblasti zhasne.

Systém ohodnocení je progresivní, v první výchozí oblasti nedostává robot žádné body. Ve druhé oblasti pak získá základní bonus, ve třetí 1.2 násobek základního bonusu, ve čtvrté pak 1.5 násobek. V případě, že se robot dostane

až do poslední oblasti, získá zvláštní bonus.



Obrázek 5.18: Prostředí druhé v simulátoru Webots

V následující části práce je popis provedených experimentů. V důsledku použitého simulátoru nemusí elitismus nutně zajistit vždy stejné ohodnocení jednoho jedince, jak jsme popsali blíže v kapitole o implementaci elitismu. Použitý simulátor má dobře propracovanou simulaci fyziky prostřednictvím knihovny ODE, a díky tomu také obsahuje prvek prokluzu kol. Může se tedy stát, že robot se v daném prostředí může zachovat jinak. Výsledné řešení tak v kombinaci s řízením robota i přes vysoké ohodnocení nemusí být vždy dostatečně robustní. Z výsledných řešení s velkým ohodnocením může být tedy nutné vybrat řešení, které je robustní. V důsledku tohoto tak může ohodnocení nejlepšího jedince kolísat i v průběhu vývoje.

5.2.1 Experiment první

Tato část popisuje testy s prvním prostředím v simulátoru Webots.

Test 1

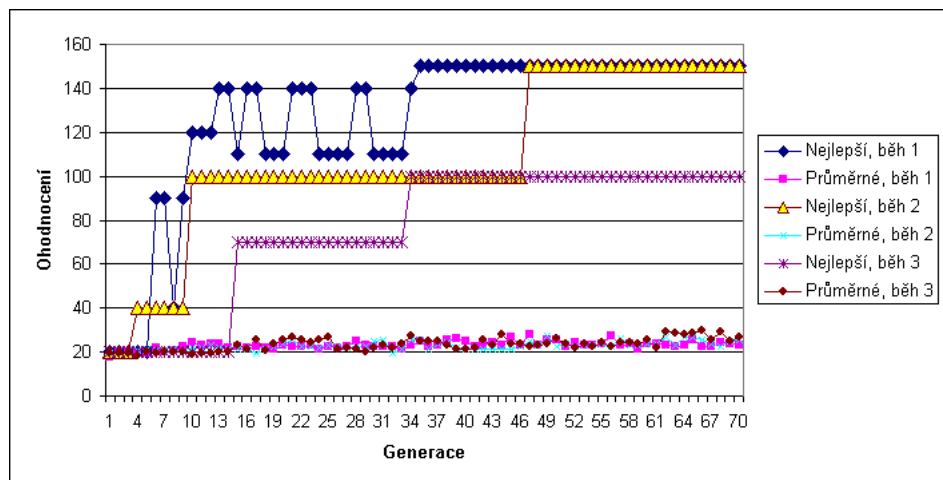
V prvním testu jsme vytvořili jednoduché podmínkové řízení, které je znázorněno v algoritmu 5. Bonusové oblasti reprezentuje obrazec ve tvaru tmavě zelené krychle, oblasti s penalizací pak světle červené krychle. Počáteční ohodnocení jedince jsme stanovili na 20. Robot získává za bonusové oblasti 50 bodů a za oblasti s penalizací naopak ztrácí 30 bodů. Ohodnocení za nalezený bonus je záměrně vyšší než penalizace, neboť chceme preferovat řízení, která najdou alespoň nějaké bonusové oblasti.

Logika řízení spočívá v použití dvou kolon, z nichž první určuje, zda robot vidí bonusovou oblast přímo před sebou. Druhá kolona zajišťuje směřování robota k bonusové oblasti.

Algoritmus 5 Podmínkové řízení, test 1

```
1: if Kolona1 > 0.45 and Kolona1 < 0.55 then  
2:   Motory vpřed                ▷ Levý motor: +300, pravý motor: +300  
3: else if Kolona2 < 0.40 then  
4:   Pojezd vlevo                ▷ Levý motor: +270, pravý motor: +300  
5: else if Kolona2 > 0.60 then  
6:   Pojezd vpravo              ▷ Levý motor: +300, pravý motor: +270  
7: else  
8:   Otáčeť se vlevo            ▷ Levý motor: -150, pravý motor: +150  
9: end if
```

Test jsme opakovali celkem třikrát, vždy s mírně pozměněnými parametry nastavení G.6. Průběh vývoje je znázorněn v grafu 5.19, z něho je patrné, že genetický algoritmus měl problémy najít řešení této úlohy i po 70 generacích.



Obrázek 5.19: Průběh vývoje při testu 1 v simulátoru

Při tomto testu se projeví nedostatky řízení, především se robot otáčí jen za předpokladu, že výstup druhé kolony je v intervalu $(0.4, 0.45) \cup (0.55, 0.60)$. To mu nedává příliš dobré možnosti ke směřování v případě, že je bonusová oblast vpravo a robot započal směřování pojezdem vpravo směrem k nalezené oblasti. Výslední jedinci jsou uvedeni v tabulce 5.20.

Nyní si popíšeme vybraného jedince z prvního běhu. V první koloně došlo k vývoji klasifikátoru přibližování k zelené (Proximity green) s parametrem 2. Tento parametr určuje, že klasifikátor uvažuje pouze zelené objekty, které jsou nejbližší robotovi a jejichž horizontální velikost je alespoň 5 pixelů. Druhá kolona používá filtr vystřížení řádků (Row cut) a ponechá v obraze pouze řádky mezi druhým a dvacátým řádkem obrazu ($2 \times \frac{42}{62}$ a $29 \times \frac{42}{62}$). Poté použije detektor zelené barvy s parametrem 145. Videozáznam chování jedinců z tohoto testu je na příloženém CD ve formátu AVI.

Nejlepší jedinec z běhu 1, 150 bodů		
Kolona 1	-	Proximity green
Parametry		10 (2)
Kolona 2	Row cut	Green color
Parametry	000010 011101 (2, 29)	10010001 (145)
Nejlepší jedinec z běhu 2, 150 bodů		
Kolona 1	Gaussian	Sensing green
Parametry	-	01 (1)
Kolona 2	Sepia	Red color
Parametry	-	10110011 (179)
Nejlepší jedinec z běhu 3, 100 bodů		
Kolona 1	Robinson	Blue color
Parametry	-	01000001 (65)
Kolona 2	Sepia	Green color
Parametry	-	10110101 (181)

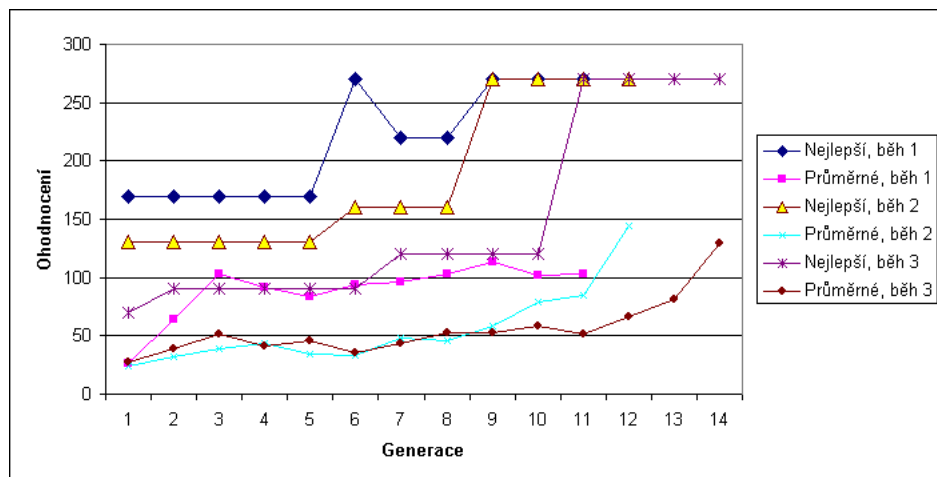
Obrázek 5.20: Výslední jedinci testu 1

Tento test jsme zopakovali, ale s pozměněným řízením a nastavením G.9. Logika zůstala podobná, ale došlo ke zpřesnění prahů. Robot se nyní otáčí jen za předpokladu, že výstup z první kolony je mimo interval (0.45, 0.55) a výstup kolony druhé je v intervalu (0.45, 0.55). Toto řízení umožňuje robotovi směřovat k bonusové oblasti. Schéma řízení je znázorněno v algoritmu 6.

Algoritmus 6 Upravené podmínkové řízení, test 1

- 1: **if** Kolona₁ > 0.45 **and** Kolona₁ < 0.55 **then**
 - 2: Motory vpřed ▷ Levý motor: +300, pravý motor: +300
 - 3: **else if** Kolona₂ ≤ 0.45 **and** Kolona₂ > 0 **then**
 - 4: Pojezd vlevo ▷ Levý motor: +270, pravý motor: +300
 - 5: **else if** Kolona₂ ≥ 0.55 **then**
 - 6: Pojezd vpravo ▷ Levý motor: +300, pravý motor: +270
 - 7: **else**
 - 8: Otáčeť se ▷ Levý motor: -150, pravý motor: +150
 - 9: **end if**
-

Test se změněným řízením jsme pustili opět třikrát. Z průběhu vývoje znázorněném v grafu 5.21 je patrné, že se změněným řízením neměl genetický algoritmus větší problémy nalézt řešení s maximálním ohodnocením.



Obrázek 5.21: Průběh vývoje při testu 1 s vylepšeným řízením

V tabulce 5.22 jsou vypsáni nejlepší jedinci z výsledné populace. Nyní si popíšeme vybraného jedince ze třetího běhu. První kolona využívá filtru gamma korekce (Gamma) a detekce žluté barvy v prostoru HSV (HSV Yellow color), v naší implementaci není parametr uvažován. Druhá kolona pak používá klasifikátor směřování k zeleným objektům (Direction green). Videozáznam chování jedinců z tohoto testu je na přiloženém CD ve formátu AVI.

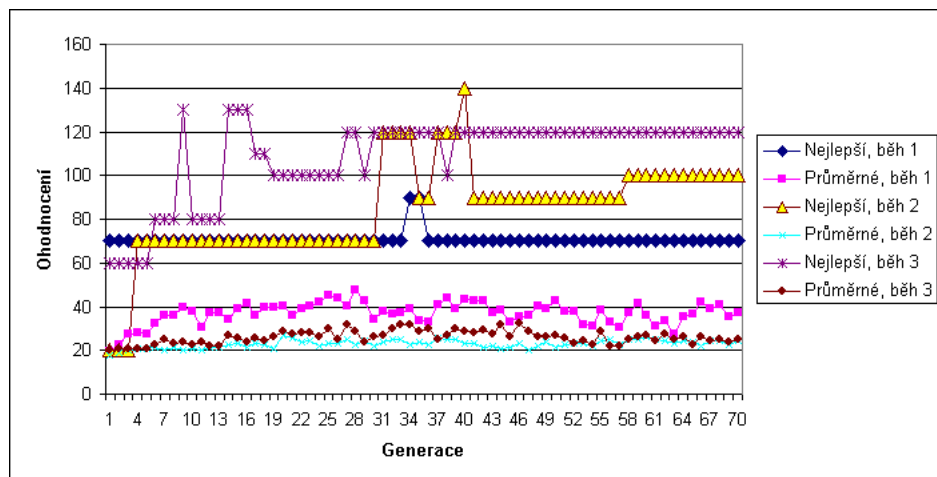
Nejlepší jedinec z běhu 1, 270 bodů		
Kolona 1	Multiple Thresholding Segmentation	HSV Black color
Parametry	11001111 00011100 01001010 01010101 (207, 28, 74, 85)	0110101 (53)
Kolona 2	Gaussian	HSV Green color
Parametry	-	0111110 (62)
Nejlepší jedinec z běhu 2, 270 bodů		
Kolona 1	Multiple Thresholding Segmentation	Histogram
Parametry	01011100 01010100 10101010 00111111 (92, 84, 170, 63)	110 11 11100 11111 (6, 3, 28, 31)
Kolona 2	-	Direction Green
Parametry	-	-
Nejlepší jedinec z běhu 3, 270 bodů		
Kolona 1	Gamma	HSV Yellow color
Parametry	-	1000110 (70)
Kolona 2	-	Direction green
Parametry	-	-

Obrázek 5.22: Výslední jedinci testu 1 s vylepšeným řízením

Test 2

Ve druhém testu jsme změnilí reprezentaci bonusových oblastí. Nyní je reprezentují červené krychle se žlutým pruhem v horní části. Ohodnocení jsme ponechali shodné s předchozím testem. V tomto testu jsme použili obě doposud prezentovaná řízení.

Test jsme opakovali opět třikrát s nastavením G.7. Průběh vývoje při tomto testu znázorněný na grafu 5.23 potvrdil nevhodnost prvního řízení pro řešení tohoto typu úloh. Zároveň ukázal, že metoda a průběh řešení jsou velmi citlivé na použité řízení robota.



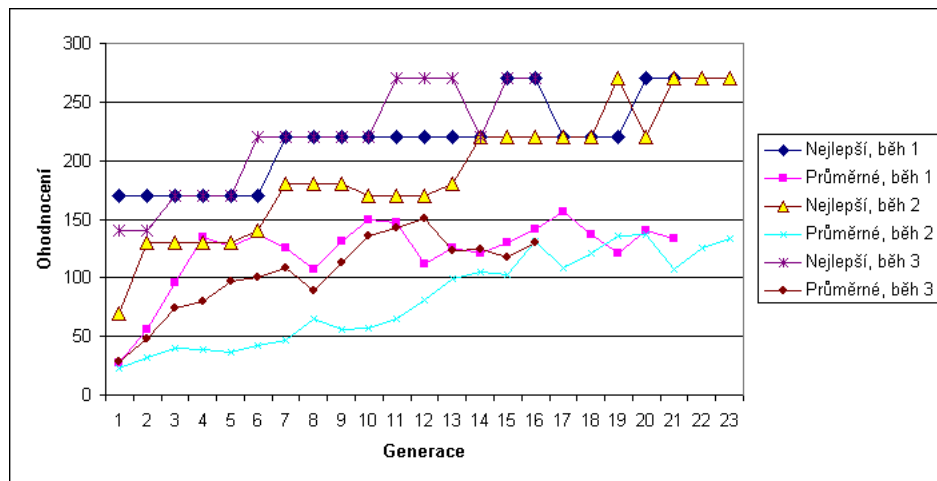
Obrázek 5.23: Průběh vývoje při testu 2

V průběhu došlo k vývoji jedinců vypsanych v tabulce 5.24. Při třetím běhu došlo k vývoji zajímavého řešení. Jedinec v první koloně používá Kirschův konvoluční operátor (Kirsch) a jako klasifikátor vnímání černé barvy (HSV Black color), v použité implementaci není parametr uvažován. Druhá kolona ponechá, prostřednictvím filtru vystřížení řádků (Row cut), pouze řádky mezi 0 a 37 ($54 \times \frac{42}{62}$). Klasifikátorem druhé kolony je pokročilý histogram (Advanced histogram). Videozáznam chování jedinců z tohoto testu je na příloženém CD ve formátu AVI.

Nejlepší jedinec z běhu 1, 70 bodů		
Kolona 1	-	Proximity green
Parametry	-	01 (1)
Kolona 2	-	Threshold
Parametry	-	11100110 (230)
Nejlepší jedinec z běhu 2, 100 bodů		
Kolona 1	Kirsch	Gap black
Parametry	-	01 (1)
Kolona 2	-	Advanced histogram
Parametry	-	10010001 00010100 11 01001010 11000101 (145, 20, 3, 74, 197)
Nejlepší jedinec z běhu 3, 120 bodů		
Kolona 1	Kirsch	HSV Black color
Parametry	-	1011010 (90)
Kolona 2	Row cut	Advanced histogram
Parametry	110110 000000 (54, 0)	01110110 11101011 00 10100001 11110100 (118, 235, 0, 161, 244)

Obrázek 5.24: Výslední jedinci testu 2

Stejný test jsme zopakovali s vylepšeným řízením z předchozího testu a nastavením G.10. Z průběhu, znázorněném v grafu 5.25 je patrné, že kombinace tohoto řízení s danou úlohou vede k vývoji velmi dobrých jedinců. Genetický algoritmus našel stabilní řešení při prvním běhu za 20 generací, při druhém za 21 a při třetím již po 15 generacích.



Obrázek 5.25: Průběh vývoje při testu 2 s vylepšeným řízením

Výslední nejlepší jedinci jsou vypsáni v tabulce 5.26. Ve třetím běhu došlo k vývoji zajímavého řešení. Jedinec v první koloně používá filtr gamma korekce (Gamma) a histogramový klasifikátor s parametrem 66. Ve druhé koloně pak Sobelův konvoluční operátor a jako klasifikátor směřování k zeleným objektům (Direction green). Videozáznam chování jedinců z tohoto testu je na přiloženém CD ve formátu AVI.

Nejlepší jedinec z běhu 1, 270 bodů		
Kolona 1 Parametry	Histogram equalization -	HSV Blue color 0111111 (63)
Kolona 2 Parametry	Sobel -	Direction green -
Nejlepší jedinec z běhu 2, 270 bodů		
Kolona 1 Parametry	Multiple Threshold Segmentation 01010110 11000000 11100101 11010011	HSV Blue color 0100010 (86, 192, 229, 211, 34)
Kolona 2 Parametry	- -	Direction green -
Nejlepší jedinec z běhu 3, 270 bodů		
Kolona 1 Parametry	Gamma correction -	Histogram 1000010 (66)
Kolona 2 Parametry	Sobel -	Direction green -

Obrázek 5.26: Výslední jedinci testu 2 s vylepšeným řízením

Test 3

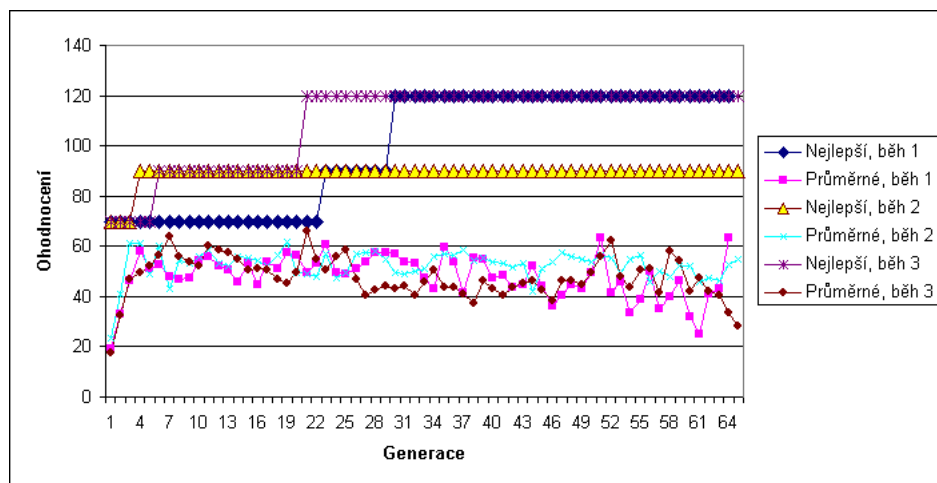
V předchozích testech jsme použili řízení, které očekává výstup ze dvou kolon. Ve třetím testu jsme použili řízení, které očekává výstup pouze z jedné kolony. Jeho schéma je znázorněno v algoritmu 7. Ohodnocení při tomto testu zůstává stejné.

Logika řízení je podobná jako v předchozích testech. První podmínka směřuje robota přímo k cíli, zbylé dvě podmínky oproti předchozím řízením směřují robota ne k cíli, ale od něj. Chceme tak znevýhodnit filtry, které určují pozici objektu v obraze.

Algoritmus 7 Řízení pro třetí test

```
1: if Kolona1 > 0.45 and Kolona1 < 0.55 then  
2:   Motory vpřed           ▷ Levý motor: +300, pravý motor: +300  
3: else if Kolona1 ≤ 0.45 then  
4:   Pojezd vpravo         ▷ Levý motor: +300, pravý motor: +270  
5: else if Kolona1 ≥ 0.55 then  
6:   Pojezd vlevo          ▷ Levý motor: +270, pravý motor: +300  
7: else  
8:   Otáčej se vlevo       ▷ Levý motor: -150, pravý motor: +150  
9: end if
```

Pro tento test jsme použili nastavení G.8. Průběh, znázorněný v grafu 5.27, ukazuje, že genetický algoritmus nebyl schopen nalézt řešení ani po 65 generacích.



Obrázek 5.27: Průběh vývoje při testu 3

Výslední jedinci jsou vypsáni v tabulce 5.28. Ačkoliv řízení neumožňuje úlohu zcela vyřešit, genetický algoritmus našel perspektivní jedince a správně našel jako řešení klasifikátor směřování k zelené barvě alespoň v jednom z běhů. Videozáznam chování jedinců z tohoto testu je na přiloženém CD ve formátu AVI.

Nejlepší jedinec z běhu 1, 120 bodů		
Kolona 1	Column cut	Direction green
Parametry	011110 010001 (30, 17)	-
Nejlepší jedinec z běhu 2, 90 bodů		
Kolona 1	RGB	Direction black
Parametry	01 (1)	-
Nejlepší jedinec z běhu 3, 120 bodů		
Kolona 1	RGB	Proximity black
Parametry	01 (1)	11 (3)

Obrázek 5.28: Výslední jedinci testu 3

Pokud řízení upravíme tak, že změním směr korekce a podmínky otáčení robota podobně jako v řízení z prvního testu, je řešením jedinec s jedním jediným klasifikátorem – směřování k zelené barvě. Schéma tohoto řízení je v algoritmu 8. Jelikož se jedná jen o jeden obrazový operátor tak je velmi pravděpodobné, že bude již v počáteční populaci. Proto jsme test neopakovali s tímto řízením. Videozáznam chování jedince z tohoto testu je na příloženém CD ve formátu AVI.

Algoritmus 8 Upravené řízení pro třetí test

- 1: **if** Kolona₁ > 0.45 **and** Kolona₁ < 0.55 **then**
 - 2: Motory vpřed ▷ Levý motor: +300, pravý motor: +300
 - 3: **else if** Kolona₁ ≤ 0.45 **and** Kolona₁ > 0 **then**
 - 4: Pojezd vlevo ▷ Levý motor: +270, pravý motor: +300
 - 5: **else if** Kolona₁ ≥ 0.55 **then**
 - 6: Pojezd vpravo ▷ Levý motor: +300, pravý motor: +270
 - 7: **else**
 - 8: Otáčeť se vlevo ▷ Levý motor: -150, pravý motor: +150
 - 9: **end if**
-

Test 4

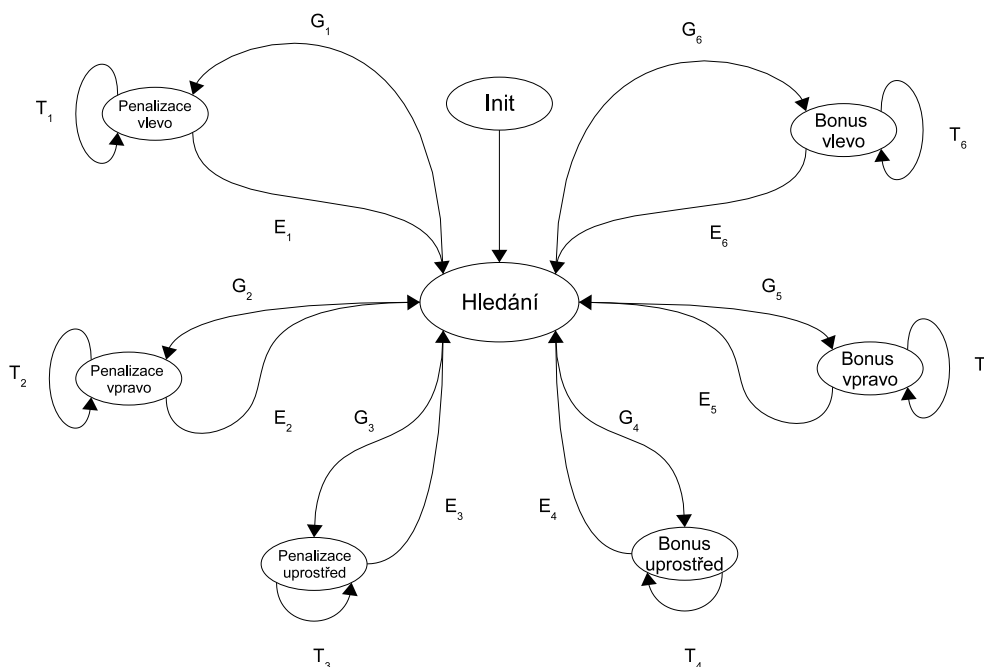
Pro další test jsme implementovali řízení konečným automatem, který obsahuje stavy korigující směřování robota k cíli. Schéma 5.30 ukazuje jakým způsobem je toto řízení řešeno. Automat má 8 stavů, hlavním je stav vyhledávání, kdy se robot snaží nalézt cíl. Hledání realizujeme otáčením robota. Z tohoto stavu může řízení přejít do dvou skupin stavů, stavů směřování k bodovému místu a stavů směřování od místa s penalizací.

Tabulka 5.29 popisuje přechody automatu, k_i je doba vyjádřená v interních cyklech, po kterou řízení setrvává v určitém stavu. V naší implementaci jsme použili $k_i = 20$ pro všechny stavy. Tj. stav do kterého se řízení dostalo na základě výstupů kolon je změněn právě po 20 interních cyklech.

Pro samotný test opět reprezentujeme bonusové oblasti jako zelené krychle a oblasti s penalizací jako krychle červené. Přičemž ohodnocení jsme stanovili na 80 bodů za bonusovou oblast a ztrátu 60 bodů za oblast s penalizací.

T_i	$t_i < k_i$	E_i	$t_i = k_i$
G_1	Kolona ₁ < 0.4	G_4	Kolona ₂ ∈ ⟨0.4, 0.6⟩
G_2	Kolona ₁ > 0.6	G_5	Kolona ₂ > 0.6
G_3	Kolona ₁ ∈ ⟨0.4, 0.6⟩	G_6	Kolona ₂ < 0.4

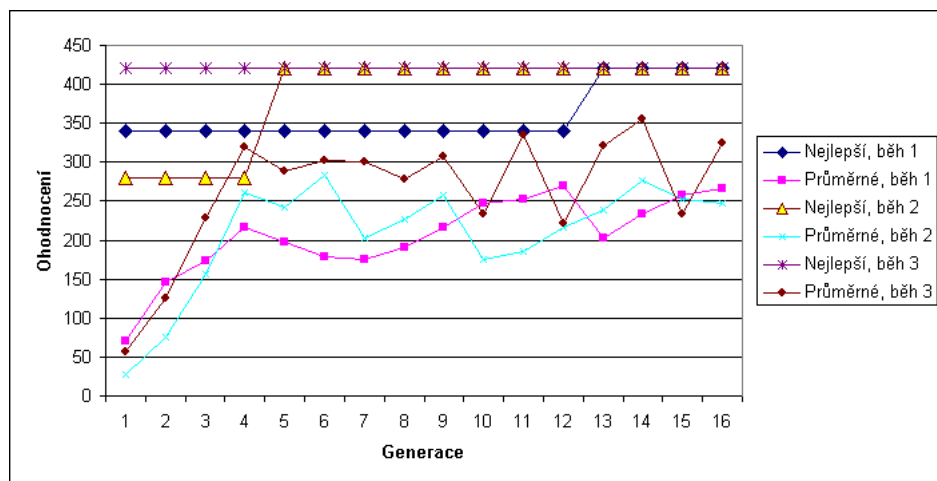
Obrázek 5.29: Tabulka přechodů



Obrázek 5.30: Řízení konečným automatem

Test jsme opakovali třikrát s nastavením G.11. Průběh vývoje ukazuje graf 5.31, z něho je patrné, že za 16 generací dosáhli jedinci ze všech tří

běhů maximálního ohodnocení. Vyvinulo se i zajímavé řešení prostřednictvím jedince ze třetího běhu. Jeho řešením je detektor mezer mezi zelenými krychlemi (Gap green), robot se pak k překážce pohybuje krouživými pohyby, namísto pojezdu vpřed. Videozáznam chování jedinců z tohoto testu je na přiloženém CD ve formátu AVI.



Obrázek 5.31: Průběh vývoje při testu 4

Výslední nejlepší jedinci jsou vypsáni v tabulce 5.32.

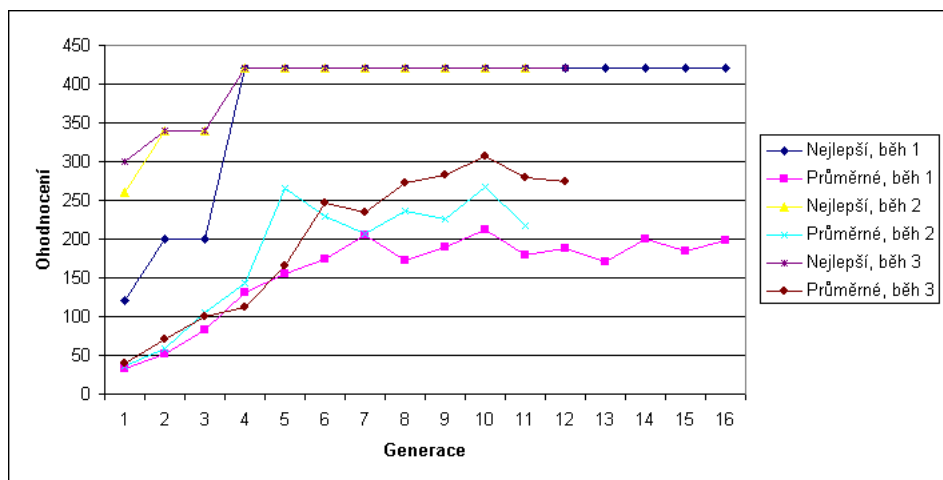
Nejlepší jedinec z běhu 1, 420 bodů		
Kolona 1 Parametry	Column cut 110110 001101 (54, 13)	HSV Blue color 0001110 (14)
Kolona 2 Parametry	Box cut 101110 (46)	HSV Green color 1100000 (96)
Nejlepší jedinec z běhu 2, 420 bodů		
Kolona 1 Parametry	- -	HSV Red color 0100100 (36)
Kolona 2 Parametry	Box cut 111100 (60)	Direction green -
Nejlepší jedinec z běhu 3, 420 bodů		
Kolona 1 Parametry	- -	Threshold 10011100 (156)
Kolona 2 Parametry	- -	Gap green 00 (0)

Obrázek 5.32: Výslední jedinci testu4

Test 5

Pro pátý test jsme ponechali řízení a systém ohodnocení z testu předchozího. Ale změnili jsme reprezentaci bonusových oblastí, nyní nejsou reprezentovány zelenými krychlemi, ale opět červenými krychlemi se žlutým pruhem v horní části.

Test jsme opakovali třikrát s nastavením G.12. Průběh vývoje znázorňuje graf 5.33.



Obrázek 5.33: Průběh vývoje při testu 5

Nejlepší jedinci jsou vypsáni v tabulce 5.34. V každém z běhů došlo k vývoji jedince, který obsahuje klasifikátor směřování k zelené. Ačkoliv je bonusová oblast reprezentována červenými krychlemi se žlutým pruhem a nikoliv zelenou, má tento klasifikátor smysl. Klasifikátor směřování k zelené je založen na segmentaci obrazu do barevného prostoru HSV, ve kterém jsou tmavé odstíny žluté velmi blízko odstínům zelené. Odstín žluté použitý pro reprezentaci bonusových oblastí spadá proto k hranicím rozlišení tohoto detektoru. Videozáznam chování jedinců z tohoto testu je na příloženém CD ve formátu AVI.

Nejlepší jedinec z běhu 1, 420 bodů		
Kolona 1	-	Blue color
Parametry	-	00001010 (10)
Kolona 2	-	Direction green
Parametry	-	-
Nejlepší jedinec z běhu 2, 420 bodů		
Kolona 1	Gaussian	HSV Blue color
Parametry	-	1110010 (114)
Kolona 2	-	Direction green
Parametry	-	-
Nejlepší jedinec z běhu 3, 420 bodů		
Kolona 1	-	Proximity black
Parametry	-	00 (0)
Kolona 2	-	Direction green
Parametry	-	-

Obrázek 5.34: Výslední jedinci testu 5

5.2.2 Experiment druhý

Test 1

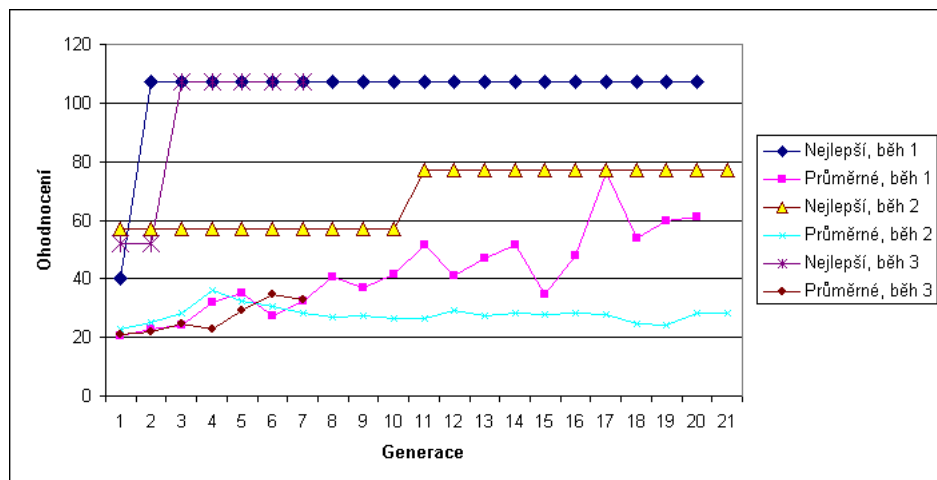
Pro test ve druhém modelovém prostředí jsme implementovali nové řízení, jehož schéma je znázorněno v algoritmu 9. Řízení očekává vstupy ze dvou kolon a jeho úkolem je směřovat robota k prvnímu cíli (kolona 1) a v případě, že první cíl není v dohledu, tak k cíli druhému (kolona 2). Tyto dva cíle odpovídají signalizacím popsaným v prostředí.

Signalizaci 1 reprezentujeme jako světle červený obdélník a *Signalizaci 2* jako světle zelený obdélník. Ohodnocení při nalezení výřezu jsme stanovili na 5 bodů, ohodnocení za vjezd do nové oblasti pak na 10 bodů a do poslední cílové oblasti 30 bodů. Využíváme zde systém progresivního ohodnocení, popsaný výše.

Algoritmus 9 Podmínkové řízení, test 1

```
1: if Kolona1 > 0.45 and Kolona1 < 0.55 then
2:   Motory vpřed           ▷ Levý motor: +300, pravý motor: +300
3: else if Kolona2 > 0.45 and Kolona2 < 0.55 then
4:   Motory vpřed           ▷ Levý motor: +300, pravý motor: +300
5: else if Kolona1 ≤ 0.45 and Kolona1 > 0 then
6:   Pojezd vlevo           ▷ Levý motor: +270, pravý motor: +300
7: else if Kolona1 ≥ 0.55 then
8:   Pojezd vpravo          ▷ Levý motor: +300, pravý motor: +270
9: else if Kolona2 ≤ 0.45 and Kolona2 > 0 then
10:  Pojezd vlevo           ▷ Levý motor: +270, pravý motor: +300
11: else if Kolona2 ≥ 0.55 then
12:  Pojezd vpravo          ▷ Levý motor: +300, pravý motor: +270
13: else
14:  Otáčej se               ▷ Levý motor: -100, pravý motor: +100
15: end if
```

Test jsme opakovali celkem třikrát s parametry G.13. Při prvním a třetím běhu došlo k vývoji jedinců řešících úlohu, avšak ve druhém běhu uvízl genetický algoritmus v lokálním maximu. Ačkoliv klasifikátory jsou správně vyvinuty na vnímání červené a zelené, jejich typ nestačí k úspěšnému řešení. V průběhu vývoje se ukázalo, že tuto úlohu velmi dobře řeší kombinace klasifikátoru směřování k červeným objektům (Direction red) a směřování k zeleným objektům (Direction green). Videozáznam chování jedinců z tohoto testu je na přiloženém CD ve formátu AVI.



Obrázek 5.35: Průběh vývoje při testu 1

Nejlepší jedinci jsou vypsáni v tabulce 5.36.

Nejlepší jedinec z běhu 1, 107 bodů		
Kolona 1	-	Direction red
Parametry	-	-
Kolona 2	Column cut	Direction green
Parametry	001000 100111 (8, 39)	-
Nejlepší jedinec z běhu 2, 77 bodů		
Kolona 1	-	Sensing green
Parametry	-	10 (2)
Kolona 2	-	Sensing red
Parametry	-	10 (2)
Nejlepší jedinec z běhu 3, 107 bodů		
Kolona 1	Box cut	HSV Red Color
Parametry	100111 (39)	1011000
Kolona 2	Prewitt	HSV Green Color
Parametry	-	0010110

Obrázek 5.36: Výslední jedinci testu 1

Kapitola 6

Závěr

Cílem této práce bylo navrhnout a implementovat řešení pro automatický vývoj filtrování obrazu z kamerového vstupu pro mobilní roboty s využitím metod zpracování obrazu. Předpokládali jsme, že je možné z jednoduchých obrazových operátorů tvořit složitější kolony, a že za předpokladu vhodné reprezentace je toto sestavování možné realizovat pomocí genetických algoritmů.

Navrhli jsme reprezentaci obrazových operátorů a způsob sestavování těchto obrazových operátorů do kolon. Toto sestavování jsme ponechali na genetickém algoritmu. Dále jsme navrhli pozměněné genetické operátory, které pracují s navrhovanou strukturou a jsou schopny přispět k řešení.

Navržený postup jsme implementovali v podobě aplikace rozdělené do dvou modulů, které jsou schopny pracovat s různými simulátory nebo i s jiným testovacím prostředím v podobě například emulace kamerového vstupu.

Implementovanou aplikaci jsme vyzkoušeli na skupině experimentů rozdělených na jednotlivé testy. Prvním experimentem byla emulace kamerového vstupu, na kterém jsme chtěli ověřit schopnosti genetického algoritmu a implementovaných obrazových operátorů bez závislosti na použitém řízení mobilního robota. Při tomto experimentu jsme provedli sérii pěti testů s různými vstupními sadami obrázků. Tato série testů prokázala, že námi implementovaný genetický algoritmus je schopen sestavit funkční kolony obrazových operátorů.

Druhý a třetí experiment jsme navrhli v prostředí vybraného robotického simulátoru s modelem robota e-puck. Při těchto testech jsme použili různé typy řízení a po robotovi vyžadovali vyhledávání bonusových oblastí a vyhý-

bání se oblastem s penalizací. Provedené testy ukázaly, že implementované řešení je velmi citlivé na použité řízení mobilního robota. I velmi dobře sestavené kolony obrazových operátorů mohou zcela selhat za předpokladu špatně nebo, pro danou úlohu, nevhodně navrženého řízení. V případě, že je řízení vhodně navržené, genetický algoritmus je schopen sestavit vhodné kolony a řešit tak danou úlohu.

Pro další rozvoj tohoto řešení je vhodné zvážit rozšíření návrhu o možnost vyvíjet i řízení robota, například ve formě neuronové sítě. V současné době nemusí aplikace najít vhodné obrazové operátory, neboť řízení mobilního robota očekává jiné hodnoty na výstupu kolon než které skutečně dávají. Možnost měnit řízení robota by zcela nepochybně zvýšila schopnosti aplikace při vývoji potřebného řešení. Je ale zřejmé, že takovéto rozšíření by značně zvětšilo prostor možných řešení prohledávaný genetickým algoritmem. To bude pravděpodobně vyžadovat například paralelní implementaci genetického algoritmu, zúžení repertoáru filtrů a klasifikátorů, ze kterých vybíráme, a jejich případné zjednodušení.

Příloha A

Použité operátory

Tato příloha stručně popisuje implementované obrazové operátory. Tyto operátory byly optimalizovány pro práci s kamerovým vstupem robota e-puck s rozlišením 52×39 pixelů. U různých obrázků proto nemusí fungovat optimálně.

A.1 Filtry

Segmentace prahováním

Hlavním úkolem segmentace obrazu je rozčlenění jeho obsahu na základě obrazových charakteristik. Můžeme provést segmentaci úplnou kdy výsledkem jsou zcela oddělené objekty v obraze nebo segmentaci parciální, která nebere v úvahu žádné objekty v obraze [16]. Dále se budeme zabývat segmentací parciální, kterou můžeme dosáhnout například prahováním.

Jednoduché prahování

Jednoduché prahování¹ pracuje pouze s jedním prahem P , který je zadán parametricky. Jednoduché prahování segmentuje obrazové body vstupního obrazu do dvou barev, v našem případě do černé a bílé. Algoritmus pro každý bod (x, y) obrazu I sečte jeho barevné složky reprezentované maticemi R pro červenou, G pro zelenou a B pro modrou složku o rozměrech $m \times n$ a spočte průměr intenzit těchto tří složek. V případě, že tento součet přesáhne práh, pak obrazový bod bude bílý, bílá barva je v naší implementaci reprezentována jako intenzita 255 všech tří barevných složek obrazu. V případě, že průměr

¹Thresholding

práh nepřesáhne je obrazový bod černý, černá je reprezentována jako nulová intenzita všech tří barevných složek. Princip jednoduchého prahování zobrazuje algoritmus 10 a příklad výstupu je pak na obrázku H.1.

Algoritmus 10 Jednoduché prahování

Require: $I(x, y) = R(x, y)G(x, y)B(x, y)$

Require: $P \geq 0$

```
1: function THRESHOLDING( $I(x, y), P$ )
2:   for  $x = 0, 0 \dots m - 1$  do
3:     for  $y = 0, 0 \dots n - 1$  do
4:        $S \leftarrow \frac{R(x, y) + G(x, y) + B(x, y)}{3}$ 
5:       if  $S > P$  then
6:          $R(x, y) = G(x, y) = B(x, y) = 255$ 
7:       else
8:          $R(x, y) = G(x, y) = B(x, y) = 0$ 
9:       end if
10:    end for
11:  end for
12:  return  $I(x, y)$ 
13: end function
```

Pokročilé prahování

Předchozí jednoduché prahování pracuje pouze s jedním prahem, to však může být nevhodné. Pokročilé prahování² pracuje se dvěma prahy P_i , $i \in \{1, 2\}$ a pixely, které budou obarveny bíle jsou právě takové, jejichž průměr součtu barevných složek leží mezi těmito dvěma prahy. Algoritmus 11 ukazuje práci pokročilého prahování a příklad výstupu na obrázku H.2.

²Advanced thresholding

Algoritmus 11 Pokročilé prahování

Require: $I(x, y) = R(x, y)G(x, y)B(x, y)$

Require: $P_1 \geq 0, P_2 \geq 0$

```
1: function ADVANCED_THRESHOLDING( $I(x, y), P_1, P_2$ )
2:   for  $x = 0, 0 \dots m - 1$  do
3:     for  $y = 0, 0 \dots n - 1$  do
4:        $S \leftarrow \frac{R(x,y)+G(x,y)+B(x,y)}{3}$ 
5:       if  $S > P_1$  and  $S < P_2$  then
6:          $R(x, y) = G(x, y) = B(x, y) = 255$ 
7:       else
8:          $R(x, y) = G(x, y) = B(x, y) = 0$ 
9:       end if
10:    end for
11:  end for
12:  return  $I(x, y)$ 
13: end function
```

Prahování s více prahy

Předpokládáme, že pro řešení určité skupiny úloh bude robot potřebovat informace o cílovém objektu, eventuálním objektu označujícím penalizaci, stěnách a případném dalším robotovi, celkem tedy o čtyřech možných objektech v obraze. Prahování s více prahy segmentuje obraz do pěti barev, čtyři pro vyjmenované objekty a jedna pro všechno ostatní.

Prahování s více prahy³ segmentuje celkem do pěti barev a využívá k tomu čtyři prahy P_i , $i \in \{1, 2, 3, 4\}$. Těmito barvami jsou červená, zelená, modrá, žlutá a černá. Algoritmus 12 nastiňuje práci tohoto filtru a příklad výstupu je pak na obrázku H.3. Jednotlivé barvy jsou v prostoru RGB reprezentovány takto, červená $(r, g, b) = (255, 0, 0)$, zelená $(r, g, b) = (0, 255, 0)$, modrá $(r, g, b) = (0, 0, 255)$, žlutá $(r, g, b) = (255, 255, 0)$ a černá $(r, g, b) = (0, 0, 0)$, přičemž přiřazení $I(x, y) = \textit{barva}$ vyjádříme jako $R(x, y) = r$, $G(x, y) = g$ a $B(x, y) = b$, přičemž R, G a B jsou matice o velikosti $m \times n$ reprezentující jednotlivé barevné složky obrazu $I(x, y)$.

³Mutliple thresholding

Algoritmus 12 Prahování s více prahy

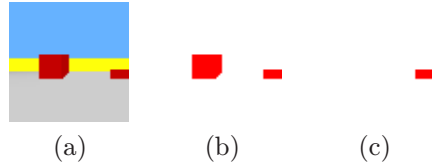
Require: $I(x, y) = R(x, y)G(x, y)B(x, y)$

Require: $P_1 \geq 0, P_2 \geq 0, P_3 \geq 0, P_4 \geq 0$

```
1: function MULTIPLE THRESHOLDING( $I(x, y), P_1, P_2, P_3, P_4$ )
2:   for  $x = 0, 0 \dots m - 1$  do
3:     for  $y = 0, 0 \dots n - 1$  do
4:        $S \leftarrow \frac{R(x,y)+G(x,y)+B(x,y)}{3}$ 
5:       if  $S < P_1$  then
6:          $I(x, y) = \text{červená}$ 
7:       else if  $S \geq P_1$  and  $S < P_2$  then
8:          $I(x, y) = \text{zelená}$ 
9:       else if  $S \geq P_2$  and  $S < P_3$  then
10:         $I(x, y) = \text{modrá}$ 
11:      else if  $S \geq P_3$  and  $S < P_4$  then
12:         $I(x, y) = \text{žlutá}$ 
13:      else if  $S \geq P_4$  then
14:         $I(x, y) = \text{černá}$ 
15:      end if
16:    end for
17:  end for
18:  return  $I(x, y)$ 
19: end function
```

Segmentace v prostoru HSV

U řízení robota s vizuálním vnímáním bude v mnoha úlohách hrát roli barva objektů a jejich vzdálenost od robota. Předpokládáme-li, že po segmentaci zbyde v obraze několik stejně barevných objektů, pak ty co jsou od robota vzdálenější budou zabírat menší plochu a naopak. Tohoto jevu můžeme velmi dobře využít. Na obrázku A.1 je znázorněn postup segmentace, na prvním obrázku A.1a je vidět vstupní obraz, který obsahuje dva červené čtverce. Předpokládejme, že segmentujeme obraz do shodné barvy, které mají tyto dva čtverce. Obrázek A.1b ukazuje již segmentovaný obraz, nyní se můžeme rozhodnout, který obrazec preferujeme, zda vzdálenější nebo bližší. Poslední obrázek A.1c ukazuje výsledný obraz po odstranění čtverce s větší plochou. Pro použití tohoto postupu ovšem musíme předpokládat, že hledané objekty jsou stejně velké, jinak měření vzdálenosti pomocí velikosti snímaného objektu nelze použít. V následující části si popíšeme konkrétní implementaci.



Obrázek A.1: Postup segmentace

Filtr⁴ postupně vytvoří na základě vstupních dat $I(x, y)$ o rozměrech $m \times n$ dvourozměrnou obrazovou mapu $M(x, y)$ o shodných rozměrech. Přičemž platí $M(x, y) = 1 \Leftrightarrow I(x, y) = c_s$ a $M(x, y) = 0 \Leftrightarrow I(x, y) = c$, $c \in \mathcal{C} \setminus \{c_s\}$, kde $\mathcal{C} = \{c_1, c_2, \dots, c_i\}$ je množina všech možných barev a $c_s \in \mathcal{C}$ je segmentovaná barva. Filtr tedy v první fázi vytvoří obdobu binárního obrazu, jehož příklad je na obrázku A.2.

0	0	...	0	0	0	0	0	0	...	0	0	0	0	0
0	0	...	0	0	0	0	0	0	...	0	0	0	0	0
0	0	...	0	0	0	0	0	0	...	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	0	...	0	0	0	0	0	0	...	0	0	0	0	0
0	0	...	0	1	1	1	1	0	...	0	0	0	0	0
0	0	...	0	1	1	1	1	0	...	0	0	0	0	0
0	0	...	0	1	1	1	1	0	...	0	1	1	1	0
0	0	...	0	1	1	1	1	0	...	0	1	1	1	0
0	0	...	0	0	0	0	0	0	...	0	0	0	0	0
0	0	...	0	0	0	0	0	0	...	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	0	...	0	0	0	0	0	0	...	0	0	0	0	0
0	0	...	0	0	0	0	0	0	...	0	0	0	0	0
0	0	...	0	0	0	0	0	0	...	0	0	0	0	0

Obrázek A.2: Binární obraz po segmentaci

Aby filtr mohl odstranit objekty určité velikosti musíme v obrazové mapě identifikovat jednotlivé oblasti stejné barvy, k tomu využijeme rekurzivní algoritmus *connected-component labeling*, který je k nalezení například v [15]. Algoritmus invertuje obrazovou mapu, jinými slovy každé $M(x, y) = 1$ označí -1. Z implementačních důvodů je vhodné již v předchozí fázi segmentace a

⁴Segmentation HSV

vytváření obrazové mapy přisoudit segmentované barvě hodnotu -1 namísto 1, tedy předchozí podmínku nahradíme takto $M(x, y) = -1 \Leftrightarrow I(x, y) = c_s$. Hodnota -1 označuje nezpracované body a hodnota 0 není vůbec uvažována a pro další zpracování nemá význam.

Algoritmus začíná s označením komponenty $L = 1$. Poté projde celou obrazovou mapu, v případě, že najde ještě nezpracovaný bod označí jej jako komponentu L . Poté prohledáváním do hloubky najde všechny sousedící nezpracované body a také je označí. Poté co zpracuje stejným způsobem všechny takovéto body, zvýší L o 1 a pokračuje v prohledávání obrazové mapy.

Výsledkem algoritmu je pak nová obrazová mapa znázorněná ve schématu A.3, každá identifikovaná spojitá oblast je označena unikátním číslem (*label*), přičemž algoritmus zároveň spočte velikost těchto oblastí. Při výsledném vynechávání oblastí jsou zpět přepsána obrazová data na základě nové obrazové mapy.

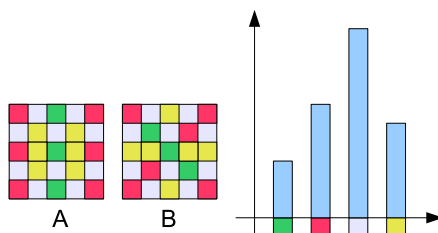
0	0	...	0	0	0	0	0	0	...	0	0	0	0	0
0	0	...	0	0	0	0	0	0	...	0	0	0	0	0
0	0	...	0	0	0	0	0	0	...	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	0	...	0	0	0	0	0	0	...	0	0	0	0	0
0	0	...	0	1	1	1	1	0	...	0	0	0	0	0
0	0	...	0	1	1	1	1	0	...	0	0	0	0	0
0	0	...	0	1	1	1	1	0	...	0	2	2	2	0
0	0	...	0	1	1	1	1	0	...	0	2	2	2	0
0	0	...	0	0	0	0	0	0	...	0	0	0	0	0
0	0	...	0	0	0	0	0	0	...	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	0	...	0	0	0	0	0	0	...	0	0	0	0	0
0	0	...	0	0	0	0	0	0	...	0	0	0	0	0
0	0	...	0	0	0	0	0	0	...	0	0	0	0	0

Obrázek A.3: Výsledek spojování komponent

Ekvalizace histogramu

Histogram je charakteristika obrazu, která kvantifikuje množství a frekvenci barev obsažených v obraze [13]. Jako statistická veličina poskytuje informaci o počtu pixelů p s určitou intenzitou i , můžeme ho však také chápat jako pravděpodobnost výskytu pixelu intenzity i v obraze. Konstrukce histogramu má charakter algoritmu, který postupně projde celý obraz a spočte pixely se stejnou intenzitou. Z kapitoly 2.2 víme, že reprezentace obrazu závisí na použitém barevném prostoru. V případě prostoru RGB je obraz složen ze tří složek červené, zelené a modré, tohoto využijeme v případě konstrukce histogramu a pro barevné obrázky v prostoru RGB budeme vytvářet histogram pro každou složku zvlášť.

Počet možných intenzit v prostoru RGB stanovíme dle implementace v rozmezí $\langle 0, 255 \rangle$ a maximální počet pixelů v histogramu jako $m \times n$, kde m, n jsou rozměry vstupního obrazu. Z uvedeného vyplývá, že $\sum_{c=0}^{255} H(c) = mn$, tedy součet všech polí histogramu je roven právě počtu pixelů v obraze.



Obrázek A.4: Histogram obrázků A a B

Jelikož histogram neposkytuje žádnou informaci o umístění jednotlivých pixelů s danou intenzitou, existuje obvykle několik obrázků, které mohou mít stejný histogram, příklad je na obrázku A.4. Obraz A obsahuje právě stejné množství pixelů stejných intenzit jako obraz B.

Obraz, který byl pořízen za nedostatečného osvětlení scény, se obvykle vyznačuje malým kontrastem, který vyjádříme jako poměr mezi nejsvětlejším a nejtmaším bodem v obraze. Tedy většina obrazových bodů bude z malého rozsahu intenzit. Na obrázku H.4 vlevo je obraz s velmi špatným kontrastem, histogramy jeho tří barevných složek jsou pak znázorněny na obrázku A.5⁵ taktéž vlevo. Z těchto histogramů je zřejmé, že obraz obsahuje obrazové body

⁵Získáno z programu IrfanView

s převážně tmavšími barvami.

Technika, která takovýto obraz zpracuje a zvýší kontrast se nazývá ekvalizace histogramu⁶, jejímž principem je rozprostřít intenzity obrazových bodů po celé šířce histogramu. Pro ekvalizovaný histogram G platí $\sum_{c=0}^k G(c) = \sum_{c=0}^k H(c)$, kde H je histogram původní [16].

Ekvalizace histogram má charakter algoritmu 13, který pro každou barevnou složku $C(x, y)$ obrazu spočte histogram H_b a poté pro každý pixel změni jeho intenzitu podle vztahu A.1, přičemž podrobný postup odvození a algoritmus je popsán v knize [16]. Vhodný algoritmus je také k nalezení v knize [13].

$$q = \frac{q_k - q_0}{mn} \sum_{i=p_0}^p H(i) + q_0 \quad (\text{A.1})$$

Algoritmus 13 Ekvalizace histogramu

Require: $C(x, y) = R(x, y) \vee G(x, y) \vee B(x, y)$

Require: Histogram H_b

```

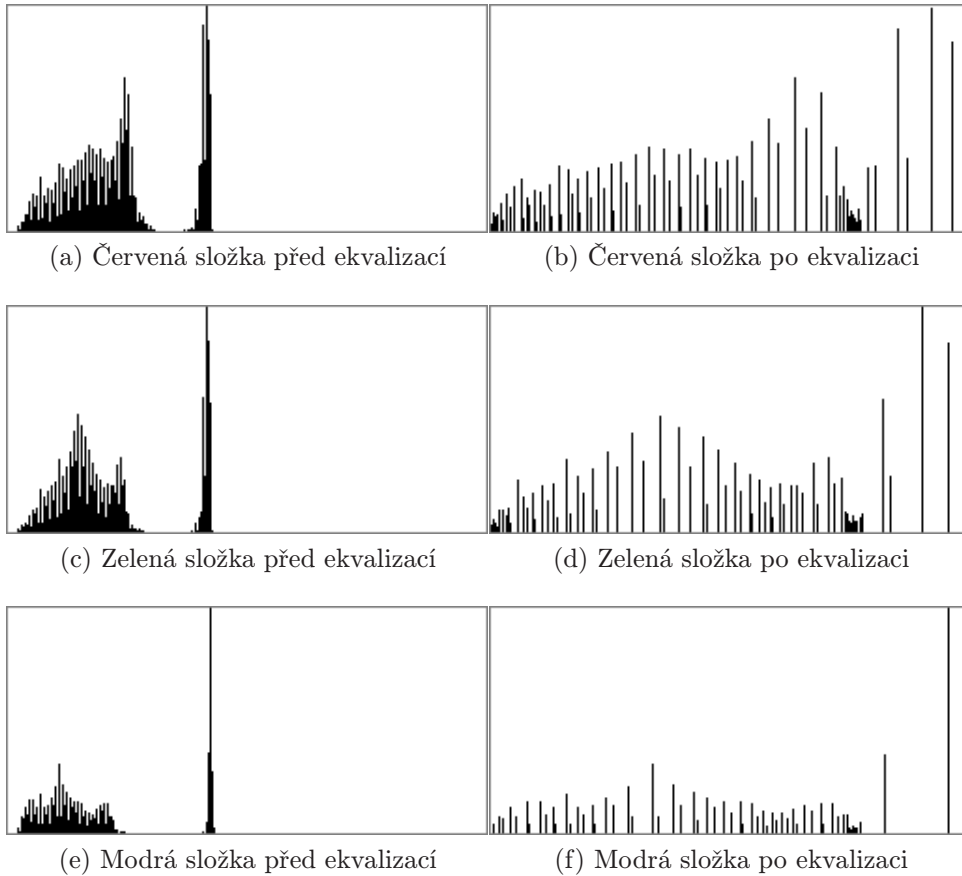
1: function HISTOGRAM EQUALIZATION( $C(x, y), H_b$ )
2:    $G \leftarrow \{0, \dots, 0\}$ 
3:    $S \leftarrow 0$ 
4:   for  $i = 0, 0 \dots q_k - 1$  do
5:      $S \leftarrow S + H_b(i)$ 
6:      $T(i) = \frac{Sq_k}{mn}$ 
7:   end for
8:   for  $x = 0, 0 \dots m - 1$  do
9:     for  $y = 0, 0 \dots n - 1$  do
10:       $C(x, y) = C(x, y)T(C(x, y))$ 
11:    end for
12:  end for
13:  return  $C(x, y)$ 
14: end function

```

Příklad výsledného běhu algoritmu je na obrázku H.4 vpravo, přičemž je vidět zvýšený kontrast a jas oproti původnímu obrazu vlevo. Histogramy pro

⁶Histogram equalization

jednotlivé barevné složky obrazu tohoto nového obrazu jsou na obrázku A.5 vpravo.



Obrázek A.5: Ekvalizace histogramu

Vystřížení čtverce

Filtr vystřížení čtverce⁷ pracuje s jedním parametrem zadaným 6 bity, který poslouží k výpočtu velikosti strany čtverce. Princip filtru je založen na umělelém zúžení pozorovacích úhlů kamery, kterého dosáhneme ořezáním krajů obrazu.

Vstupem je obraz $I(x, y)$ o šířce m a výšce n , ze zadaného prahu algoritmus vypočte délku strany čtverce b , neboť práh P přímo neudává velikost

⁷Boxcut

strany. Pro výpočet délky strany čtverce použijeme menší z velikostí obrazu $I_{\min} = \min(m, n)$. Maximální hodnota P_{\max} pak rozděljuje I_{\min} na P_{\max} částí, prostým vynásobením aktuální hodnoty prahu P velikostí těchto částí získáme velikost strany čtverce.

Se znalostí délky strany vypočteme souřadnice potřebné k určení prostoru čtverce, k tomu vypočteme hodnoty $R = \lfloor \frac{m-b}{2} \rfloor$ a $S = \lfloor \frac{n-b}{2} \rfloor$. Čtverec v obraze pak vymezují 4 souřadnice (R, R) , (R, S) , (S, R) a (S, S) . Podrobný popis je v algoritmu 13. Realizace filtru je ukázána na obrázku H.5.

Algoritmus 14 Vystřížení čtverce

Require: $I(x, y) = R(x, y)G(x, y)B(x, y)$

Require: m, n

Require: P

```

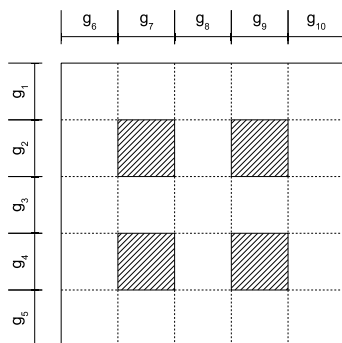
1: function BOXCUT( $I(x, y), P$ )
2:    $b \leftarrow \lceil \frac{\min(m, n)}{P_{\max}-1} P \rceil$ 
3:    $R \leftarrow \lfloor \frac{m-b}{2} \rfloor$ 
4:    $S \leftarrow \lfloor \frac{n-b}{2} \rfloor$ 
5:    $y \leftarrow 0$ 
6:   for  $i = R, i \leq S$  do
7:      $x \leftarrow 0$ 
8:     for  $j = R, j \leq S$  do
9:        $I'(x, y) \leftarrow I(i, j)$ 
10:       $x \leftarrow x + 1$ 
11:    end for
12:     $y \leftarrow y + 1$ 
13:  end for
14:  return  $I'(x, y)$ 
15: end function

```

Obdélníky

Filtr⁸ ze vstupního obrazu ořízne vše až na dva nebo čtyři výřezy uprostřed obrazu. Počet těchto výřezů je zadán 1 bitovým parametrem. Algoritmus vypočte $\text{gap}_m = \lfloor \frac{m}{r} \rfloor$ a $\text{gap}_n = \lfloor \frac{n}{s} \rfloor$, kde $k = r = 5$ v případě 4 čtverců a $k = 5, r = 3$ v případě dvou výřezů. Tyto výřezy jsou zázorněny ve schématu A.6.

⁸Boxes



Obrázek A.6: Rozdělení obrazu pro $k = r = 5$

Algoritmus poté ve vstupním obraze $I(x, y)$ změní barvu všech pixelů mimo čtverce (g_2, g_7) , (g_2, g_9) , (g_4, g_7) a (g_4, g_9) na černou, tedy nastaví všechny barevné kanály RGB na 0. Funkce $\text{InGap}(x, y)$ prostým porovnáním souřadnic ověří zda je pixel (x, y) mimo uvedené čtverce nebo nikoliv. V algoritmu 15 je výše uvedené podrobně popsáno.

Algoritmus 15 Obdélníky

Require: $I(x, y) = R(x, y)G(x, y)B(x, y)$

Require: m, n

Require: P

```

1: function BOXES( $I(x, y), P$ )
2:    $\text{gap}_m \leftarrow \lfloor \frac{m}{r} \rfloor$ 
3:    $\text{gap}_n \leftarrow \lfloor \frac{n}{s} \rfloor$ 
4:   for  $x = 0, 0 \dots m - 1$  do
5:     for  $y = 0, 0 \dots n - 1$  do
6:       if  $\text{InGap}(x, y)$  then
7:          $I'(x, y) \leftarrow I(i, j)$ 
8:       else if
9:         then  $I'(x, y) \leftarrow (0, 0, 0)$ 
10:      end if
11:    end for
12:  end for
13:  return  $I'(x, y)$ 
14: end function

```

Vystřížení sloupce

Filtr⁹ odstraní kraje obrazu a ponechá pouze vybraný sloupec. Filtr má dva 6 bitové parametry P_1 a P_2 , které slouží k zadání počáteční souřadnice sloupce a koncové souřadnice sloupce. Souřadnice s_1 a s_2 sloupce určíme stejným způsobem jako v případě vystřížení čtverce. Tedy $s_1 = \lceil \frac{m}{50} P_1 \rceil$ a s_2 analogicky. Příklad výstupu filtru na obrázku H.7.

Řádek

Filtr řádku¹⁰ zachová v obraze pouze jediný řádek, přičemž ostatní řádky jsou zahozeny. 6 bitový parametr P filtru je, podobně jako v předchozích filtrech, transformován na skutečné souřadnice řádku v obraze vzorcem $\lceil \frac{\min(m,n)}{P_{\max}-1} P \rceil$.

Vystřížení řádků

Filtr¹¹ odstraní kraje obrazu a ponechá pouze vybrané řádky, vybrané sloupce jsou definovány dvěma parametry P_1 a P_2 , které definují počáteční a konečný řádek s_1 a s_2 . Výpočet je obdobný podle $s_1 = \lceil \frac{m}{62} P_1 \rceil$ a s_2 analogicky. Příklad zpracování je na obrázku H.8.

Gamma korekce

Gamma korekce¹² je filtr, který odstraňuje nelinearitu zobrazování intenzity [13]. Princip filtru je v konstrukci mapování lineárního signálu na nelineární. Lze jej také použít pro opačné mapování. Pro každou možnou hodnotu i spočteme její novou hodnotu podle (A.2) [19], kde $\lambda \in \mathbb{R}$. Výpočetní schéma je znázorněno v algoritmu 16. Příklad práce filtru je na obrázku H.9.

$$i := i^\lambda \tag{A.2}$$

⁹Columncut

¹⁰Row

¹¹Rowcut

¹²Gamma

Algoritmus 16 Gamma korekce

Require: $I(x, y)$

```
1: function GAMMA CORRECTION( $I(x, y)$ )
2:    $\lambda \leftarrow 2.5$ 
3:   for  $x = 0, 0 \dots m - 1$  do
4:     for  $y = 0, 0 \dots n - 1$  do
5:        $I(x, y) \leftarrow 255 \left( \frac{I(x, y)}{255} \right)^\lambda$ 
6:     end for
7:   end for
8:   return  $I(x, y)$ 
9: end function
```

Šedotónový obraz

Filtr¹³ vytvoří ze vstupního barevného obrazu obraz šedotónový. Konverzi obrazového bodu z prostoru RGB do šedotónové hladiny vyjadřuje vztah (A.3). Tímto vztahem pak nahradíme pro daný pixel jeho jednotlivé barevné složky. Tento vztah byl odvozen empiricky ve vztahu k vnímání lidského oka [13]. Příklad práce filtru je na obrázku H.11.

$$S(x, y) = 0.299 \times R(x, y) + 0.587 \times G(x, y) + 0.114 \times B(x, y) \quad (\text{A.3})$$

Sépia

Filtr¹⁴ vytvoří ze vstupního obrazu obraz v odstínech hnědé. Filtr pro každý pixel obrazu $I(x, y)$ změní hodnoty jednotlivých složek podle předpisu (A.4), tyto vztahy byly odvozeny empiricky podobně jako v předchozím případě transformace obrázku do šedotónové hladiny [18]. Příklad práce filtru je na obrázku H.12.

$$\begin{aligned} R'(x, y) &= 0.393 \times R(x, y) + 0.769 \times G(x, y) + 0.189 \times B(x, y) \\ G'(x, y) &= 0.349 \times R(x, y) + 0.686 \times G(x, y) + 0.168 \times B(x, y) \\ B'(x, y) &= 0.272 \times R(x, y) + 0.534 \times G(x, y) + 0.131 \times B(x, y) \end{aligned} \quad (\text{A.4})$$

¹³Greyscale¹⁴Sepia

Snížení rozlišení

Při zpracování výpočetně náročných obrazových operátorů má velikost vstupního obrazu zásadní vliv na rychlost zpracování, což je v případě robotických aplikací obzvláště patrné. Přitom v některých případech obrazová informace, kterou daný obraz nese je invariantní k velikosti tohoto obrazu. Filtr snížení rozlišení¹⁵ odebere každý druhý řádek a každý druhý sloupec, výsledkem je pak obraz poloviční velikosti.

RGB

Filtr RGB¹⁶ ponechá v obraze pouze jednu z obrazových složek v závislosti na parametru, obě zbylé nastaví na 0. Filtr má 2 bitový parametr, který udává, která barevná složka obrazu bude zachována. Hodnoty 0 a 3 náleží složce červené, 1 složce zelené a 2 složce modré. Příklad práce tohoto operátoru je na obrázku H.10.

Konvoluční operátory

Konvoluci dvou signálů definujeme jako matematický operátor dvou funkcí (A.5) a označíme ji symbolem \star , přičemž $h(x)$ označíme jako konvoluční jádro, které určuje způsob výpočtu nové hodnoty signálu v bodě x [13].

$$f(x) \star h(x) = \int_{-\infty}^{\infty} f(x - \alpha)h(\alpha)d\alpha \quad (\text{A.5})$$

Pro nakládání s obrazy zobecníme konvoluci do dvou rozměrů a konvoluční jádro pak definujeme jako tabulku o rozměrech $k \times k$, přičemž algoritmus konvoluce projde všechny body obrazu $I(x, y)$ a na každý aplikuje toto konvoluční jádro. Nová hodnota pixelu je pak vyjádřena vztahem podle (A.6). Konvoluční jádro můžeme též nazvat konvolučním operátorem.

$$I \star h = \sum_{i=-k}^k \sum_{j=-k}^k I(x - i, y - j)h(i, j) \quad (\text{A.6})$$

¹⁵Lower resolution

¹⁶RGB

Hranové detektory

Hranové detektory jsou zaměřeny na lokalizaci změny intenzity v obraze, hrany jsou pixely, kde se prudce mění hodnota intenzity [16]. Takovýto gradient může být spocítán podle vzorce (A.7), přičemž θ je úhel hrany vůči vertikální ose obrazu [12].

$$Gr(x, y) = \frac{\delta F(x, y)}{\delta x} \cos \theta + \frac{\delta F(x, y)}{\delta y} \sin \theta \quad (\text{A.7})$$

Pro zjednodušení výpočetní náročnosti použijeme k výpočtu řádkový $G_R(x, y)$ a sloupcový gradient $G_C(x, y)$, prostorový gradient je pak dán vztahem (A.8) [12].

$$Gr(x, y) = \sqrt{G_R(x, y)^2 + G_C(x, y)^2} \quad (\text{A.8})$$

Výpočet gradientu nemusíme provádět přímo, ale můžeme využít konvoluční operátory. Následující konvoluční operátory aproximují jednotlivé gradienty, převzaty z [16]. V implementaci používáme pouze masky h_1 a h_3 . Pokud maska přesahuje okraje obrazu, pak za hodnoty chybějících částí obrazu dosazujeme 0.

Sobel

Konvoluční masky pro Sobelův filtr¹⁷:

$$h_1 = \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & 2 & -1 \\ \hline \end{array} \quad h_2 = \begin{array}{|c|c|c|} \hline 0 & 1 & 2 \\ \hline -1 & 0 & 1 \\ \hline -2 & -1 & 0 \\ \hline \end{array} \quad h_3 = \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \quad \dots$$

Příklad výsledku Sobelova filtru je na obrázku H.13.

¹⁷Sobel operator

Prewitt

Konvoluční masky pro operátor Prewitt¹⁸:

$$h_1 = \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -1 & -1 \\ \hline \end{array} \quad h_2 = \begin{array}{|c|c|c|} \hline 0 & 1 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & -1 & 0 \\ \hline \end{array} \quad h_3 = \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \quad \dots$$

Příklad výsledku filtru Prewittové je na obrázku H.14.

Kirsch

Konvoluční masky pro Kirschův operátor¹⁹:

$$h_1 = \begin{array}{|c|c|c|} \hline 3 & 3 & 3 \\ \hline 3 & 0 & 3 \\ \hline -5 & -1 & -5 \\ \hline \end{array} \quad h_2 = \begin{array}{|c|c|c|} \hline 3 & 3 & 3 \\ \hline -5 & 0 & 3 \\ \hline -5 & -5 & 3 \\ \hline \end{array} \quad h_3 = \begin{array}{|c|c|c|} \hline -5 & 3 & 3 \\ \hline -5 & 0 & 3 \\ \hline -5 & 3 & 3 \\ \hline \end{array} \quad \dots$$

Příklad výsledku Kirschova filtru je na obrázku H.15.

Robinson

Konvoluční masky pro Robinsonův operátor²⁰:

$$h_1 = \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & -2 & 1 \\ \hline -1 & -1 & -1 \\ \hline \end{array} \quad h_2 = \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline -1 & -2 & 1 \\ \hline -1 & -1 & 1 \\ \hline \end{array} \quad h_3 = \begin{array}{|c|c|c|} \hline -1 & 1 & 1 \\ \hline -1 & -2 & 1 \\ \hline -1 & 1 & 1 \\ \hline \end{array} \quad \dots$$

Příklad výsledku Robinsonova filtru je na obrázku H.16.

Průchod nulou

Velkou nevýhodou předchozích konvolučních operátorů je velká citlivost na velikost objektu a citlivost na množství šumu v obraze. První derivace obrazové funkce má extrém v bodě, kde je v obraze hrana a druhá derivace je v tom samém bodě rovna nule. Hledání maxima je ovšem obtížnější než výpočet nulových bodů druhé derivace. K výpočtu pak můžeme použít techniku LoG, kterou si popíšeme dále [16].

¹⁸Prewitt operator

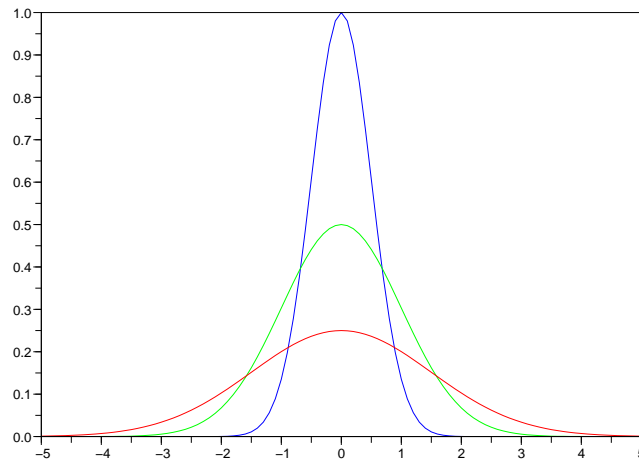
¹⁹Kirsch operator

²⁰Robinson operator

LoG

Při počítání nulových bodů druhé derivace obrazové funkce se musíme vypořádat s případným šumem v obraze, toho můžeme dosáhnout použitím Gaussova operátoru $G(x, y)$. Gaussův operátor vychází z Gaussovi funkce, která je dána vztahem (A.9), kde $a > 0$, $b > 0$ a $\sigma > 0$. Na obrázku A.7 je zobrazen průběh Gaussovy funkce.

$$f(x) = ae^{-\frac{(x-b)^2}{2\sigma^2}} \quad (\text{A.9})$$



Obrázek A.7: Průběh Gaussovy funkce pro $a = 1, c = 0.5$ (modrá), $a = 0.5, c = 1$ (zelená) a $a = 0.25, c = 1.5$ (červená)

Dvourozměrný Gaussův operátor pro $a = 1$ a $b = 0$ pak můžeme definovat vztahem (A.10), kde parametr σ vyjadřuje váhu jednotlivých pixelů, čím vzdálenější jsou pixely od středu operátoru, tím menší váhu mají.

$$G(x, y) = e^{-\frac{(x^2+y^2)}{2\sigma^2}} \quad (\text{A.10})$$

Na obrazová data $I(x, y)$ tedy aplikujeme Gaussův operátor $G(x, y, \sigma) \star I(x, y)$ a získáme mírně rozmazaný obraz zbavený šumu. K výpočtu druhé derivace dále použijeme Laplaceův operátor definovaný vztahem (A.11) [16].

$$\nabla^2 g(x, y) = \frac{\delta^2 g(x, y)}{\delta x^2} + \frac{\delta^2 g(x, y)}{\delta y^2} \quad (\text{A.11})$$

Provedeme tedy *Laplacian of Gaussian* $\nabla^2[G(x, y, \sigma) \star I(x, y)]$, tuto operaci díky linearity operátorů převedeme na $[\nabla^2 G(x, y, \sigma)] \star I(x, y)$. Místo přímého výpočtu můžeme použít aproximaci pomocí konvolučního operátoru²¹, převzato z [16]:

$$h = \begin{array}{|c|c|c|c|c|} \hline 0 & 0 & -1 & 0 & 0 \\ \hline 0 & -1 & -2 & -1 & 0 \\ \hline -1 & -2 & 16 & -2 & -1 \\ \hline 0 & -1 & -2 & -1 & 0 \\ \hline 0 & 0 & -1 & 0 & 0 \\ \hline \end{array}$$

Příklad práce tohoto filtru je pak na obrázku H.17.

Rozmazávání obrazu

Filtry pro rozmazání obrazu jsou určeny k potlačení šumu v obraze nebo nerovnoměrností v obraze [16]. Dále si popíšeme dvě techniky filtrů rozmazávajících obraz, Gaussián a průměrování.

Gaussián

Aproximaci Gaussova²² operátoru popsaného v předchozí části můžeme realizovat pomocí konvolučního operátoru, převzatého z [16].

$$h = \begin{array}{|c|c|c|c|c|c|c|} \hline 1 & 3 & 7 & 9 & 7 & 3 & 1 \\ \hline 3 & 12 & 26 & 33 & 26 & 12 & 3 \\ \hline 7 & 26 & 55 & 70 & 55 & 26 & 7 \\ \hline 9 & 33 & 70 & 90 & 70 & 33 & 9 \\ \hline 7 & 26 & 55 & 70 & 55 & 26 & 7 \\ \hline 3 & 12 & 26 & 33 & 26 & 12 & 3 \\ \hline 1 & 3 & 7 & 9 & 7 & 3 & 1 \\ \hline \end{array}$$

²¹Log

²²Gaussian

Průměrování

Rozmazání obrazu můžeme také dosáhnout pomocí průměrování²³ hodnot z osmi okolních pixelů p_i a hodnoty pixelu, který měníme. Nová hodnota pixelu je pak $p = \frac{1}{9} \sum_{i=0}^9 p_i$. Tento výpočet můžeme realizovat konvolučním operátorem [16].

$$h = \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

Příklad práce tohoto filtru je pak na obrázku H.19.

²³Mean

A.2 Klasifikátory

Histogramový klasifikátor

Na vlastnostech histogramu, které jsme zmínili v části o filtrech, můžeme založit také konstrukci některých klasifikátorů. Implementované filtry podávají informaci o zastoupení pixelů s určitou intenzitou, dále si popíšeme jejich konkrétní implementaci.

Jednoduchý histogramový klasifikátor

Jednoduchý histogramový klasifikátor²⁴ má celkem čtyři parametry, barevnou složku, práh, a dva rozsahy r_1 a r_2 . Barevná složka zabírá 2 bity s tím, že 00 a 11 jsou ekvivalentní, tedy červená složka je zvýhodněna. Práh je určen 3 bity, které poskytují 8 možných hodnot, práh je proto odstupňován po hodnotách 10 a nabývá tedy hodnot $\{10, 20, \dots, 90\}$ vyjádřeno procentuálně.

Histogramový filtr pak spočítá počet pixelů mezi rozsahem₁ a rozsahem₂. Pokud počet pixelů přesáhne práh je výstupem $T = 1$, pokud nepřesáhne je výstupem $T = 0$, výpočet vyjadřuje vztah (A.12)

$$T = \begin{cases} 1, & \sum_{c=r_1}^{r_2} H(c) > \text{práh} \\ 0, & \text{jinak} \end{cases} \quad (\text{A.12})$$

Pokročilý histogramový klasifikátor

Pokročilý histogramový klasifikátor²⁵ pracuje na stejném principu jako jednoduchý histogramový filtr, avšak má navíc druhý práh a tyto prahy přímo určují rozsahy. Hodnota součtu pixelů musí být mezi prahem₁ a prahem₂.

Výsledek je dán následujícím vztahem:

$$T = \begin{cases} 1, & \text{práh}_1 < \sum_{c=r_1}^{r_2} H(c) < \text{práh}_2 \\ 0, & \text{jinak} \end{cases}$$

²⁴Histogram

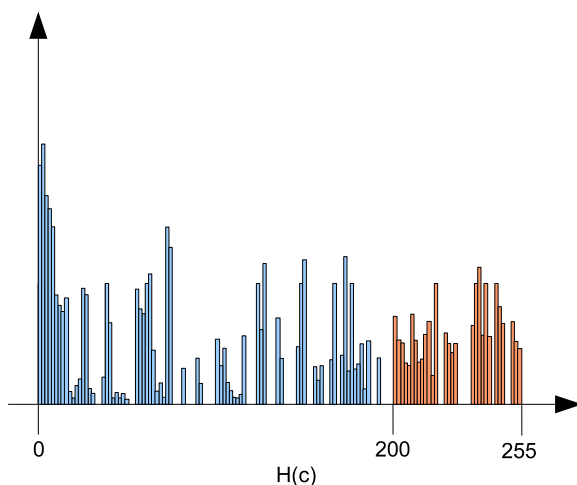
²⁵Advanced histogram

Prahování

Klasifikátor prahování²⁶ počítá všechny pixely s nenulovou intenzitou v obraze, pokud součet přesáhne stanovený práh, pak je výsledkem detekce $T = 1$, jinak $T = 0$, pokud je alespoň nenulový, pak je výstupem $T = 0.5$.

Detektor barev v RGB prostoru

U detekce barev v RGB²⁷ prostoru se nejedná přímo o detekci konkrétní barvy, ale o detekci zastoupení barevné složky v obraze. Klasifikátor má pevně nastaven rozsah odpovídající v histogramu intenzitám 200 až 255, a zároveň má parametricky zadaný práh, jež musí být při součtu počtu pixelů v daném rozsahu překročen. Schéma části histogramu pro výpočet je znázorněno na obrázku A.8.



Obrázek A.8: Část histogramu pro detekci barev

Následující vztah definuje výpočet detektoru barev v prostoru RGB:

$$T = \begin{cases} 1, & \sum_{200}^{255} H(c) > \text{práh} \\ 0, & \text{jinak} \end{cases}$$

²⁶Thresholding

²⁷Color detector

Detektor barev v HSV prostoru

Barevný prostor RGB se nehodí pro detekci konkrétních barev, neboť není intuitivně zřejmé, která barva vznikne kombinací poměrů tří intenzit barevných složek. Oproti tomu v barevném prostoru HSV je zřejmé, která barva je reprezentována. Barevný tón prostoru udává přímo barvu, bez nutnosti míchání poměrů barevných složek RGB. Saturace neboli sytost barvy udává zastoupení jiné barvy a konečně jasová hodnota definuje světlost barvy.

V případě, že chceme reprezentovat například červenou v prostoru RGB pak ji zcela jistě definuje hodnota poměru RGB (255, 0, 0), avšak není jednoduché zjistit, jakými poměry jsou zastoupeny jednotlivé odstíny červené barvy. v prostoru HSV je červená definována trojicí (0, 1, 1), další odstíny získáme například snižováním sytosti, při hodnotě $s = 0.5$ bude barva růžová. Změnou jasu můžeme červenou barvu detekovat za různých světelných podmínek při snímání kamerou.

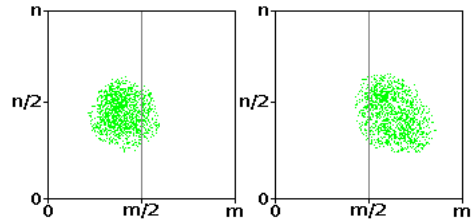
Klasifikátor barev v prostoru HSV²⁸ nejprve transformuje jednotlivé barvy z prostoru RGB do prostoru HSV podle dispozic popsanych v kapitole o teoretických základech. Poté v takto získaném obraze detekuje barvy. Implementováno je celkem pět přednastavených detektorů pro detekci černé, žluté, červené, zelené a modré barvy. Výstupem je poměr zastoupení barvy v obraze.

Hustota

Klasifikátor hustoty²⁹ rozdělí obraz vertikálně na dvě poloviny, v případě liché šířky je levá strana o jeden pixel širší. Dále s těmito dvěma polovinami nakládá jako se samostatnými obrazy, kdy pro každý z nich spočítá histogram pro jednotlivé základní barvy prostoru RGB, červenou, zelenou a modrou. Klasifikátor spočte z histogramu počty pixelů s intenzitou od 240 do 255 pro každý z obrazů a hodnoty porovná s dříve uloženými. Pokud se hodnota výrazně změnila, tj. hustota zastoupení pixelů s intenzitami od 240 do 255 v jedné ze sledovaných základních barev se přesunula více na levou nebo na pravou stranu pak jeho výsledkem je $T = 0.5$ pro pohyb na levou stranu a $T = 1$ pro pohyb na pravou stranu, $T = 0$ v případě, že pixely nejsou vůbec zastoupeny.

²⁸Color HSV

²⁹Density



Obrázek A.9: Zobrazení hustoty

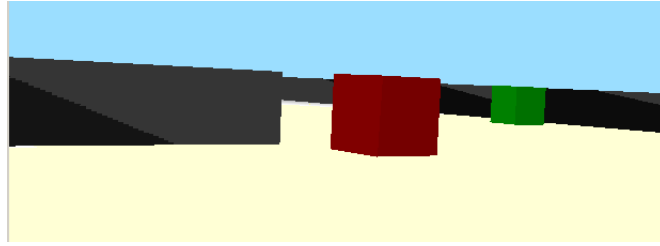
Detekce objektů

Vnímání objektů v obraze je obecně obtížná úloha, v závislosti na úhlu snímání kamery a velikosti jejího rozlišení mohou být tvary značně zkresleny. V článku [11] autoři navrhuji detekci objektů prostřednictvím segmentace vstupního obrazu. Metoda spočívá v segmentaci vstupního obrazu do čtyř barev, černé, červené, zelené a bílé, tato segmentace je ukázána na obrázku A.10b. Všechny obrazové body v odstínech černé, tj. včetně šedých a tmavě šedých jsou transformovány do černé barvy. Všechny body v odstínech červené, například tmavě růžová, jsou transformovány do jasně červené barvy, respektive zelené. Bílá barva reprezentuje všechny ostatní obrazové body.

Nyní vytvoříme křivku zastoupení jednotlivých obrazových bodů zvlášť pro jednotlivé barvy. Křivku pro barvu b reprezentuje vektor v_b , který vytváříme podle algoritmu 17, přičemž segmentovaný obraz označíme jako I_s o rozměrech $m \times n$. Vektor je délky m , a jeho každá složka nabývá hodnot $\{0, 1, \dots, n\}$. Hodnoty složek vektoru určují pozici posledního pixelu dané barvy v segmentovaném obraze, přičemž hodnota 0 znamená, že pixel s touto barvou není vůbec zastoupen. Grafické zobrazení těchto vektorů je na obrázcích A.10c, A.10d a A.10e.

Z uvedeného vyplývá, že jsme velmi rychlým způsobem získali informaci o šířce objektů s danými barvami, o jejich počtu a také informaci o jejich vzdálenosti. V případě, že se robot pohybuje v prostředí například s černými zdmi, pak jsme získali hloubkovou mapu těchto zdí. Tyto informace vytvářejí základ pro několik klasifikátorů, které můžeme využít a dále si je popíšeme podrobněji. Jedná se o vnímání objektů, směřování k objektům, detektory mezer a vzdálenost od objektu.

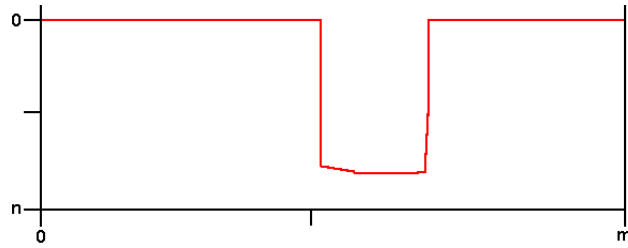
Všechny implementované detektory založené na výše uvedeném využívají metodu nalezení objektů (`NajdiObjekty()`). Metoda projde celý získaný vek-



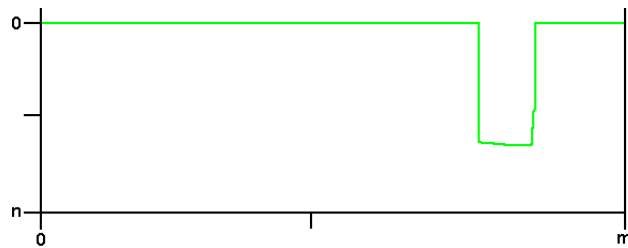
(a) Původní obraz



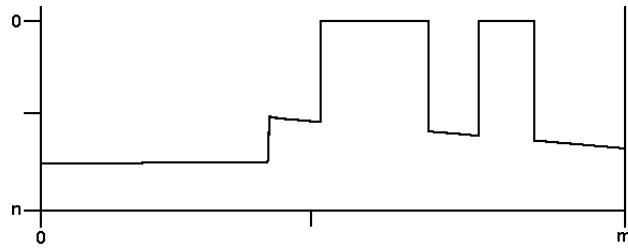
(b) Segmentace do čtyř barev



(c) Červená složka



(d) Zelená složka



(e) Černá složka

Obrázek A.10: Vnímání objektů

Algoritmus 17 Vytváření křivky

```
1:  $v_b \leftarrow \{0, \dots, 0\}$ 
2: for  $x = 0, 0 \dots m - 1$  do
3:   pozice = 0
4:   for  $y = 0, 0 \dots n - 1$  do
5:     if  $I_s(x, y) = b$  then
6:       pozice =  $y$ 
7:     end if
8:   end for
9:    $v_b[x] = \text{pozice}$ 
10: end for
```

tor a zaznamenává změny jednotlivých složek. V případě, že je $|v_{i+1}^b - v_i^b| > 3$ a zároveň $v_{i+1}^b \neq 0$ pak je detekován nový objekt, struktura tvořící informace o objektu pak nese informaci o poloze v křivce a horizontální a vertikální velikosti objektu. Počet takto nalezených objektů označíme jako N_o .

Vnímání objektů

Předchozí algoritmus můžeme využít k detekci objektů barev, do kterých jsme segmentovali obraz a ze kterých jsme získali křivku rozložení obrazových bodů v obraze, tj. objekty černé, zelené a červené barvy³⁰.

Jelikož křivka opíše všechny objekty utčité barvy poskytuje nám možnost určit jejich velikost, respektive vzdálenost. Vnímání objektů rozpoznává objekty na základě jejich šířky ve čtyřech režimech, vnímání velmi vzdálených objektů, vzdálených, blízkých a velmi blízkých objektů. Jednotlivé objekty rozlišuje podle prahu P .

Směrování k objektům

Směrování k objektům³¹ rozdělí vytvořený vektor $v = \{v_1, v_2, \dots, v_m\}$, kde m je šířka obrazu, reprezentující křivku obepínající objekty v obraze na dvě poloviny, poté pro každou polovinu sečte hodnoty tohoto vektoru. Výsledek

³⁰Sensing

³¹Direction

T je pak určen porovnáním těchto hodnot podle (A.13), l a r reprezentují uložený součet pro první respektive druhou polovinu vektoru.

$$T = \begin{cases} 0.2, & \text{jestliže } l > r \\ 0.8, & \text{jestliže } l < r \\ 0.5, & \text{jestliže } l = r \\ 0, & \text{jinak} \end{cases} \quad (\text{A.13})$$

Detektor mezer

Jelikož jsme z vytvořených grafů získali hloubkovou mapu prostředí můžeme ji využít k detekování mezer mezi objekty. Doposud jsme na objekty nahlíželi jako na výkyvy ve vytvořené křivce, avšak z obrázku A.10e je patrné, že pokud je výkyv co nejmenší objekt se v obraze nachází daleko nebo není zastoupen vůbec. Detektor mezer³² tohoto faktu využívá a jeho výstupem je pozice v obraze, kde je výkyv křivky nejmenší. Nevýhodou tohoto řešení je chybná detekce v případě zákrytu jiným předmětem.

Vzdálenost objektů

Podobným způsobem jako detektor mezer pracuje i detektor vzdálenosti objektů³³, v závislosti na parametru pracuje ve dvou režimech, hledání nejvzdálenějšího objektu, nejbližšího objektu a dvou speciálních režimech hledání vzdáleného velkého objektu a hledání blízkého velkého objektu. Detektor podle nastaveného poarmetru uvažuje pouze vzdálené, blízké, vzdálené a větší, a blízké a větší objekty.

³²Gap

³³Proximity

Příloha B

Komunikační protokol

Tato příloha stručně přibližuje použitý komunikační protokol mezi jednotlivými moduly implementované aplikace.

Morpheus a Nyx

Handshake

[REQUESTING]...jednoduchý handshake, kterým hlavní modul ohlásí připravenost

Přenos jedince

[#F|C|N|No|P|PAR...]...řetězec se zakódovaným jedincem
[BEGIN TRANSFER]...přepne modul do režimu příjmu jedince
[END TRANSFER]...přepne modul zpět z režimu příjmu jedince

Přenos nastavení

[INPUTS %i]...předá informaci o počtu kolon

Ovládání prostředí

[RESET]...žádost o znovunastavení prostředí
[TEST]...žádost o o testování jedince

[RESPOND]...žádost o odpověď pro potřeby testu připravenosti

Ovladač a Webots controller

[BEGIN TRANSFER]...přepne modul do režimu příjmu jedince

[END TRANSFER]...přepne modul zpět z režimu příjmu jedince

[LONG TEST]...žádost o testování jedince

[#F|C|N|No|P|PAR...]. . . řetězec se zakódovaným jedincem

Příloha C

Formáty souborů

Tato příloha popisuje formáty používaných konfiguračních a exportních souborů.

Morpheus

Hlavní konfigurační soubor `morpheus_appset.cfg`

`useDefaultPath=0/1` upravuje používání přednastavené cesty v aplikaci
`defaultSettingsPath=cesta` cesta s umístěním konfiguračního souboru

Konfigurační soubor `morpheus_defset.cfg`

`defaultIp=X.Y.Z.W` IP adresa vedlejšího modulu Nyx

`defaultPort=N` číslo portu modulu Nyx

`useMultipleNyx=0/1` upřesňuje použití více modulů Nyx

`availableIps=N` počet IP adres pro modul Nyx

`nyxIp.i=X.Y.Z.W` IP adresa i-tého modulu Nyx

`useTimeout=0/1` upřesňuje použití timeoutu

`timeout=N` velikost timeoutu (ms)

`useBackup=0/1` zapíná zálohování

`backup=N` zálohování každých N generací

`inputs=N` počet vstupů řízení

`numberOfLoops=N` počet generací genetického algoritmu

`populationSize=N` velikost populace genetického algoritmu

`selectionType=N` mechanismus výběru jedinců v genetickém algoritmu

`useElitism=0/1` upřesňuje použití elitismu

`individualsForElitism=N` počet jedinců pro elitismus

initialChainSize=N počáteční velikost filtrů
initialClassifierSize=N počáteční velikost klasifikátorů
maximumSize=N maximální velikost jedince
convoyType=0 typ kolon
enableAging=0/1 upřesňuje použití stárnutí
age=N počet generací pro stárnutí
enableProtection=0/1 upřesňuje použití protekce
protection=N počet generací pro protekci
probChainMutation=N pravděpodobnost parametrické mutace (filtr)
probChainParametricCrossover=N pravděpodobnost parametrického křížení (filtr)
probChainIncrement=N pravděpodobnost inkrementu jedince (filtr)
probChainDecrement=N pravděpodobnost dekrementu jedince (filtr)
probChainInsert=N pravděpodobnost vložení nového operátoru (filtr)
probChainRemove=N pravděpodobnost vyjmutí nového operátoru (filtr)
probChainSwap=N pravděpodobnost prohození dvou operátorů (filtr)
probChainTotalInversion=N pravděpodobnost inverze dvou operátorů (filtr)
probChainPartialInversion=N pravděpodobnost parciální inverze dvou operátorů (filtr)
probChainTransformation=N pravděpodobnost transformace (filtr)
probClassMutation=N pravděpodobnost parametrické mutace (klasifikátor)
probClassParametricCrossover=N pravděpodobnost parametrického křížení (klasifikátor)
probClassIncrement=N pravděpodobnost inkrementu (klasifikátor, eventuální rozšíření)
probClassDecrement=N pravděpodobnost dekrementu (klasifikátor, eventuální rozšíření)
probClassRemove=N pravděpodobnost vyjmutí operátoru (klasifikátor, eventuální rozšíření)
probClassInsert=N pravděpodobnost vložení operátoru (klasifikátor, eventuální rozšíření)
probClassTransformation=N pravděpodobnost transformace operátoru (klasifikátor)

Export populace

Header=Morpheus backup file hlavička souboru
Date=DD.MM.RRRR datum pořízení exportu

Time=H:M:S čas pořízení exportu
NumberOfIndividuals=N počet jedinců
Inputs=N počet vstupů
ConvoyType=0 typ kolony
NumberOfConvoys_i=N počet kolon jedince i
Fitness_i=N ohodnocení jedince i
Age_i=N věk jedince i
Protection_i=N protekce jedince i
MaxSize_i=N maximální velikost jedince i
NumberOfGenotypes_i_j=N počet operátorů jedince i kolony j
NumberOfChains_i_j=N počet filtrů jedince i kolony j
NumberOfTerminators_i_j=N počet klasifikátorů jedince i kolony j
Parameters_i_j_k=011... parametry operátoru k jedince i kolony j
Id_i_j_k=N typ operátoru k jedince i kolony j
Name_i_j_k=... jméno operátoru k jedince i kolony j
NumberOfParameters_i_j_k= počet parametrů operátoru k jedince i kolony j

Export jedince

Header=Morpheus backup file hlavička souboru
Type=Individual typ souboru
Inputs=N počet vstupů
ConvoyType=0 typ kolony
NumberOfConvoys=N počet kolon
Fitness=N ohodnocení jedince
NumberOfGenotypes_i=N počet operátorů kolony i
NumberOfChains_i=N počet filtrů kolony i
NumberOfTerminators_i=N počet klasifikátorů kolony i
Parameters_i_j=011... parametry operátoru j kolony i
Id_i_j=N typ operátoru j kolony i
Name_0_0=... jméno operátoru j kolony i
NumberOfParameters_i_j=N počet parametrů operátoru j kolony i

Nyx

Hlavní konfigurační soubor

useDefaultSettingsPath=0/1 upravuje používání přednastavené cesty v aplikaci

defaultSettingsPath=cesta cesta s umístěním konfiguračního souboru

useDefaultPath=0/1 upravuje používání přednastavené cesty v aplikaci

Konfigurační soubor

enableLogging=0/1 upravuje používání logování

useMorpheusTimeout=0/1 upravuje používání timeoutu

morpheusTimeout=N timeout směrem k modulu Morpheus

morpheusPort=N velikost timeoutu (ms)

useDriverTimeout=0/1 upravuje používání timeoutu

driverTimeout=N timeout směrem k ovladači

Příloha D

Návod na přidání nového obrazového operátoru

Tato příloha stručně přibližuje princip přidávání nového obrazového operátoru, pro tento příklad jej nazvěme *NewOperator* a třídu, která jej implementuje jako *CNewOperator*. Nejprve je nutno vytvořit unikátní identifikační číslo v souboru *species.h* v případě, že se jedná o filtr pak do enumerátoru *ChainId*, v případě, že se jedná o klasifikátor pak do enumerátoru *TerminatorId*.

```
namespace ChainId { 1
    enum 2
    { 3
        ID_... , 4
        ID_NewOperator , 5
    }; 6
}; 7
8
namespace TerminatorId { 9
    enum 10
    { 11
        ID_... , 12
        ID_NewOperator , 13
    }; 14
}; 15
```

Poté je třeba vytvořit samotný operátor, každý obrazový operátor dědí z obecné třídy *COperator* a implementuje její rozhraní. Korpus hlavičkového souboru nového operátoru je následující:

```

#ifndef _NEW_OPERATOR_H_ 1
#define _NEW_OPERATOR_H_ 2
3
#include "../.. / core / operator .h" 4
#include "../.. / .. / genotype / genotypepart .h" 5
#include "../.. / .. / genotype / genotypes .h" 6
7
class CNewOperator : public COperator 8
{ 9
public : 10
    CNewOperator (); 11
    ~CNewOperator (); 12
13
    void RunOperator(CImage * image); 14
}; 15
16
#endif 17

```

Nyní je třeba vytvořit specifikaci genotypu tohoto nového operátoru, který dědí z třídy CGenotypePart. Tento specifikujeme v souboru genotypes.h respektive genotypes.cpp kam je nutno přidat nový genotyp podle následujícího vzoru:

```

class CNewOperatorGenotype : public CGenotypePart 1
{ 2
public : 3
    CNewOperatorGenotype (); 4
    ~CNewOperatorGenotype (); 5
}; 6

```

Genotyp tohoto operátoru specifikuje veškeré jeho vlastnosti týkající se počtu parametrů, jejich velikosti a také identifikační číslo.

```

CNewOperatorGenotype::CNewOperatorGenotype() 1
{ 2
    id = ChainId::ID_NewOperator; 3
    species = Species::Chaining; 4
    // pro klasifikátor pak: 5
    // id = TerminatorId::ID_NewOperator; 6
    // species = Species::Terminating; 7
8
    // Počet parametrů 9

```

```

transitionsCount = k;                                10
transitions = new STransition[k];                    11
// Jednotlivé přechody parametrů (A-B včetně)        12
// Počáteční bit parametru                            13
transitions[0].start = 0;                             14
// Koncový bit parametru, včetně                       15
transitions[0].end = ...;                              16
// Délka parametru v bitech                            17
transitions[0].size = ...;                             18
...                                                    19
                                                    20
// Celková délka parametrů v bitech                    21
parametersLength = L;                                  22
parameters = new int[parametersLength];                23
                                                    24
for (int i = 0; i < parametersLength; i++)            25
    parameters[i] =                                    26
        CRandomizer::GetRandom() % 2;                 27
}                                                       28

```

Mezi vlastnosti operátoru patří `id`, což je unikátní identifikátor typu. Dále pak náležení do skupiny operátorů `species`, v případě klasifikátorů se jedná o `Species::Terminating`, a v případě filtrů o `Species::Chaining`.

Proměnná `transitionsCount` udává počet parametrů a `transitions` jednotlivé předchody mezi nimi pro křížení, přičemž `start` a `end` udává počáteční a koncovou pozici parametru včetně a `size` celkovou velikost. Proměnná `parametersLength` definuje délku všech parametrů.

V této chvíli máme vytvořen nový obrazový operátor, jeho unikátní identifikační číslo a genotyp s vlastnostmi. Nyní zbývá zařadit tento nový operátor do systému. K tomu slouží dva soubory, `pool.cpp` (v modulu `Nyx`, respektive v cílovém prostředí) a `genoms.cpp` (v modulu `Morpheus`). Tyto soubory obsahují statické třídy, které na základě identifikátorů vrací genotyp operátoru, operátor samotný a jméno.

V případě, že nový operátor je filtr, bude v souboru `pool.cpp` bude náš nový operátor vypadat takto, v případě klasifikátoru analogicky:

```

COperator * CPool::GetChainOperator(int id)           1
{                                                       2
    switch (id)                                        3

```

```

    {
        case ChainId::...
        case ChainId::ID_NewOperator:
            return new CNewOperator();
        ...
    }
}
wxString CPool::GetChainName(int id)
{
    switch (id)
    {
        case ChainId::...
        case ChainId::ID_NewOperator:
            return wxT("New_operator_name");
        ...
    }
}

```

V souboru genoms.cpp pak v modulu Morpheus takto, v případě klasifikátoru analogicky:

```

CGenotypePart * CGenoms::GetChainGenotype(int id)
{
    switch (id)
    {
        case ChainId::...
        case ChainId::ID_NewOperator:
            return new
                CNewOperatorGenotype();
        ...
    }
}

```

Příloha E

Konstrukce ovladače

Tato příloha shrnuje konstrukci nového ovladače pro modul Nyx. Každý ovladač je potomkem abstraktní třídy CDriver, který poskytuje univerzální rozhraní mezi prostředím a modulem Nyx. Každý nový ovladač musí implementovat šest základních metod.

virtual wxString GetManifest() metoda vrací proměnnou wxString, která obsahuje informace o ovladači, jeho verzi, název atp. Tato informace není povinná, a je potřebná pouze pro uživatele, modul Nyx tuto informaci vypisuje při startu ovladače.

virtual void Stop() metoda zastaví exekuci v testovacím prostředí, v mnoha případech toto zastavení nemusí být vhodné. Je vhodnější nechat zastavení na samotném prostředí.

virtual int Init(long timeout) metoda inicializuje testovací prostředí, v případě emulace to například znamená nahrání obrázků do paměti. Zároveň je možné inicializovat timeout případného síťového spojení.

virtual int Reset() metoda znovunastaví testovací prostředí, například vrátí robota do výchozí pozice.

virtual double Test(CUndecodedIndividual * ind) metoda očekává na vstupu jedince, tohoto otestuje v testovacím prostředí a vrátí získané ohodnocení.

virtual void LongTest(CUndecodedIndividual * ind) metoda očekává na vstupu jedince a předá jej testovacímu prostředí na otestování bez

počítání ohodnocení.

Při konstrukci ovladače je také možné využít rozhraní IDriver, které poskytuje další vazbu mezi modulem Nyx a ovladačem. Potomek tohoto rozhraní je hlavní modul. Pokud chceme toto rozhraní použít, je nutné vytvořit konstruktor ovladače v tomto tvaru `NewDriver(IDriver * iDriverInterface)`. Poté můžeme využívat dvě následující metody.

virtual void PrintLog(const wxString& logMessage) metoda vytiskne zprávu logMessage do hlavního okna aplikace s aktuálním časem a odřádkuje.

virtual void PrintLog(wxString logMessage, int i) metoda pracuje stejným způsobem jako předchozí, ale předpokládáme, že uživatel bude nejčastěji chtít vypsat hodnoty. Pro jiné hodnoty než je int je nutné použít předchozí metodu a využít vlastností wxString.

V modulu Nyx je poté nutné v souboru mainframe.cpp v metodě void MainFrame::OnRunDriver(wxCommandEvent& event) zaměnit vytváření ovladače za nově vytvořený.

Příloha F

Obsah příloženého CD

Součástí práce je také příložené CD, které obsahuje implementované moduly aplikace. Tato příloha stručně shrnuje jeho obsah.

Experimenty Adresář obsahuje výsledné populace a jedince z provedených experimentů. Dále obsahuje zkompilovaná řízení a moduly aplikace. Součástí jsou také soubory simulátoru Webots s modelovými prostředími.

Instalátory Adresář obsahuje instalátor simulátoru Webots 5.7.3¹ a použité knihovny wxWidgets 2.8.6²

Moduly Adresář obsahuje zkompilované moduly.

Videa Adresář obsahuje nasnímaná videa s chováním vyvinutých jedinců v daných prostředích ve formátu AVI s použitým kodekem DivX 4.12.

Zdrojové kódy Adresář obsahuje všechny zdrojové kódy použitých řízení a modulů.

Zkompilované knihovny wxWidgets 2.8.6 Adresář obsahuje zkompilované použité knihovny wxWidgets 2.8.6

¹<http://www.cyberbotics.com>

²<http://www.wxwidgets.org>

Příloha G

Nastavení genetického algoritmu v experimentech

Filtry			
Parametrická mutace	0.30	Vyjmutí	0.10
Parametrické křížení	0.30	Vložení	0.35
	0.30		0.15
	0.20		0.35
Transformace	0.35	Inkrement	0.35
			0.35
			0.25
		Dekrement	0.10
Klasifikátory			
Parametrická mutace	0.30	Parametrické křížení	0.30
Transformace	0.35		
	0.45		
	0.35		
Ostatní parametry			
Velikost populace	200	Elitismus	2
Vstupy	2	Maximální velikost	2
Stárnutí	0	Vstupy	1
Protektce	0	Algoritmus výběru	Turnaj

Obrázek G.1: Parametry experimentu emulace kamery 1

Filtry			
Parametrická mutace	0.20	Vyjmutí	0.10 0.10 0.15
Parametrické křížení	0.20 0.30 0.20	Vložení	0.35 0.35 0.25
Transformace	0.35	Inkrement	0.25 0.35 0.35
		Dekrement	0.10
Klasifikátory			
Parametrická mutace	0.30	Parametrické křížení	0.30
Transformace	0.30		
Ostatní parametry			
Velikost populace	200	Elitismus	2
Vstupy	2	Maximální velikost	2
Stárnutí	0	Vstupy	1
Protektce	0	Algoritmus výběru	Turnaj

Obrázek G.2: Parametry experimentu emulace kamery 2

Filtry			
Parametrická mutace	0.30	Vyjmutí	0.20 0.10 0.10
Parametrické křížení	0.30 0.30 0.50	Vložení	0.30 0.30 0.20
Transformace	0.35	Inkrement	0.35 0.35 0.25
		Dekrement	0.20 0.10 0.10
Klasifikátory			
Parametrická mutace	0.30	Parametrické křížení	0.30 0.30 0.50
Transformace	0.35 0.35 0.25		
Ostatní parametry			
Velikost populace	200	Elitismus	2 1 3
Vstupy	2	Maximální velikost	2
Stárnutí	0 0 5	Vstupy	1
Protektce	4 3 3	Algoritmus výběru	Turnaj

Obrázek G.3: Parametry experimentu emulace kamery 3

Filtry			
Parametrická mutace	0.20	Vyjmutí	0.20
Parametrické křížení	0.30	Vložení	0.40
			0.30
Transformace	0.35	Inkrement	0.45
			0.45
			0.40
		Dekrement	0.20
Klasifikátory			
Parametrická mutace	0.30	Parametrické křížení	0.30
Transformace	0.40		
	0.40		
	0.35		
Ostatní parametry			
Velikost populace	200	Elitismus	2
Vstupy	2	Maximální velikost	2
Stárnutí	0	Vstupy	2
	0		
	5		
Protektce	2	Algoritmus výběru	Turnaj

Obrázek G.4: Parametry experimentu emulace kamery 4

Filtry			
Parametrická mutace	0.30	Vyjmutí	0.10
Parametrické křížení	0.50	Vložení	0.60
			0.70
Transformace	0.35	Inkrement	0.65
			0.35
			0.45
		Dekrement	0.10
Klasifikátory			
Parametrická mutace	0.30	Parametrické křížení	0.50
Transformace	0.35		
	0.35		
	0.45		
Ostatní parametry			
Velikost populace	200	Elitismus	3
Vstupy	2	Maximální velikost	2
Stárnutí	5	Vstupy	2
Protektce	3	Algoritmus výběru	Turnaj

Obrázek G.5: Parametry experimentu emulace kamery 5

Filtry			
Parametrická mutace	0.20	Vyjmutí	0.10
Parametrické křížení	0.30	Vložení	0.30
Transformace	0.35	Inkrement	0.35
			0.30
		Dekrement	0.20
Klasifikátory			
Parametrická mutace	0.30	Parametrické křížení	0.30
Transformace	0.35		
	0.25		
	0.25		
Ostatní parametry			
Velikost populace	40	Elitismus	1
Vstupy	2	Maximální velikost	2
Stárnutí	0	Vstupy	2
Protektce	2	Algoritmus výběru	Turnaj

Obrázek G.6: Parametry experimentu 1, test 1

Filtry			
Parametrická mutace	0.20	Vyjmutí	0.20
Parametrické křížení	0.30	Vložení	0.30
			0.10
Transformace	0.15	Inkrement	0.10
			0.25
		0.25	
		Dekrement	0.20
			0.20
		0.10	
Klasifikátory			
Parametrická mutace	0.30	Parametrické křížení	0.30
Transformace	0.15		
	0.25		
	0.25		
Ostatní parametry			
Velikost populace	40	Elitismus	1
Vstupy	2	Maximální velikost	2
Stárnutí	0	Vstupy	2
Protektce	2	Algoritmus výběru	Turnaj

Obrázek G.7: Parametry experimentu 1, test 2

Filtry			
Parametrická mutace	0.20	Vyjmutí	0.20
Parametrické křížení	0.30	Vložení	0.10
Transformace	0.25	Inkrement	0.20
		Dekrement	0.10
Klasifikátory			
Parametrická mutace	0.30	Parametrické křížení	0.30
Transformace	0.25		
Ostatní parametry			
Velikost populace	40	Elitismus	1
Vstupy	2	Maximální velikost	2
Stárnutí	0	Vstupy	2
Protektce	2	Algoritmus výběru	Turnaj

Obrázek G.8: Parametry experimentu 1, test 3

Filtry			
Parametrická mutace	0.20	Vyjmutí	0.10
Parametrické křížení	0.30	Vložení	0.10
Transformace	0.30	Inkrement	0.10
		Dekrement	0.10
Klasifikátory			
Parametrická mutace	0.30	Parametrické křížení	0.30
Transformace	0.30		
Ostatní parametry			
Velikost populace	40	Elitismus	1
Vstupy	2	Maximální velikost	2
Stárnutí	0	Vstupy	2
Protektce	2	Algoritmus výběru	Turnaj

Obrázek G.9: Parametry experimentu 1, test 1, vylepšené řízení

Filtry			
Parametrická mutace	0.20	Vyjmutí	0.10
Parametrické křížení	0.30	Vložení	0.10
Transformace	0.20	Inkrement	0.10
	0.30		
	0.30		
Swap	0.10	Dekrement	0.10
Partial inversion	0.10	Total inversion	0.00
Klasifikátory			
Parametrická mutace	0.30	Parametrické křížení	0.30
Transformace	0.20		
	0.30		
	0.30		
Ostatní parametry			
Velikost populace	40	Elitismus	1
Vstupy	2	Maximální velikost	3
Stárnutí	0	Vstupy	2
Protektce	3	Algoritmus výběru	Turnaj

Obrázek G.10: Parametry experimentu 1, test 2, vylepšené řízení

Filtry			
Parametrická mutace	0.20	Vyjmutí	0.10
Parametrické křížení	0.30	Vložení	0.10
Transformace	0.30	Inkrement	0.10
		Dekrement	0.10
Klasifikátory			
Parametrická mutace	0.30	Parametrické křížení	0.30
Transformace	0.20		
Ostatní parametry			
Velikost populace	40	Elitismus	1
Vstupy	2	Maximální velikost	2
Stárnutí	0	Vstupy	2
Protektce	5	Algoritmus výběru	Turnaj
	3		
	3		

Obrázek G.11: Parametry experimentu 1, test 4

Filtry			
Parametrická mutace	0.20	Vyjmutí	0.10
Parametrické křížení	0.30	Vložení	0.10
	0.20		
Transformace	0.25	Inkrement	0.10
	0.35		0.20
	0.353		0.20
		Dekrement	0.10
Klasifikátory			
Parametrická mutace	0.30	Parametrické křížení	0.30
Transformace	0.25		
	0.35		
	0.35		
Ostatní parametry			
Velikost populace	20	Elitismus	1
Vstupy	2	Maximální velikost	2
Stárnutí	0	Vstupy	2
Protekcce	2	Algoritmus výběru	Turnaj

Obrázek G.12: Parametry experimentu 1, test 5

Filtry			
Parametrická mutace	0.20	Vyjmutí	0.10
Parametrické křížení	0.20	Vložení	0.10
	0.30		
Transformace	0.30	Inkrement	0.20
	0.30		
	0.35		
		Dekrement	0.10
Klasifikátory			
Parametrická mutace	0.30	Parametrické křížení	0.30
Transformace	0.30		
	0.30		
	0.35		
Ostatní parametry			
Velikost populace	40	Elitismus	1
Vstupy	2	Maximální velikost	2
Stárnutí	0	Vstupy	2
Protekcce	3	Algoritmus výběru	Turnaj
	3		
	0		

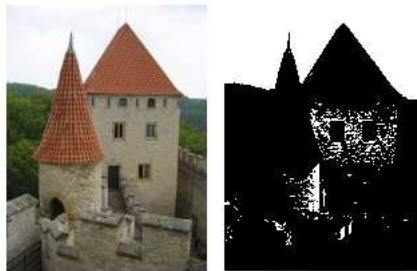
Obrázek G.13: Parametry experimentu 2, test 1

Příloha H

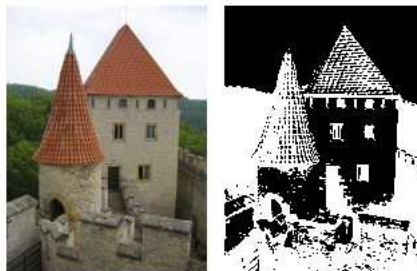
Obrazová příloha

Tato příloha obsahuje některé vybrané příklady práce filtrů.

Použité filtry



Obrázek H.1: Prahování s parametrem 140



Obrázek H.2: Pokročilé prahování s parametry 20 a 100



Obrázek H.3: Prahování s více filtry s parametry 20, 100, 130 a 240



Obrázek H.4: Ekvalizace histogramu



Obrázek H.5: Vystřížení čtverce s parametrem 30



Obrázek H.6: Čtverce



Obrázek H.7: Vystřížení sloupců s parametry 30 a 60



Obrázek H.8: Vystřížení řádků s parametry 30 a 60



Obrázek H.9: Gamma korekce



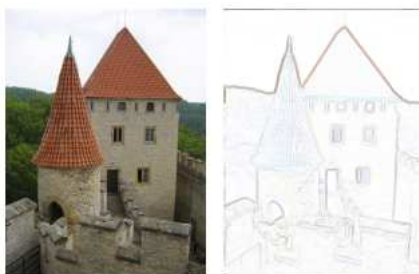
Obrázek H.10: RGB filtr s parametrem 1



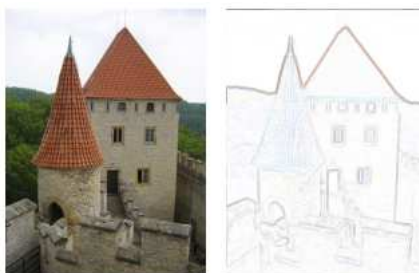
Obrázek H.11: Šedotónový obrázek



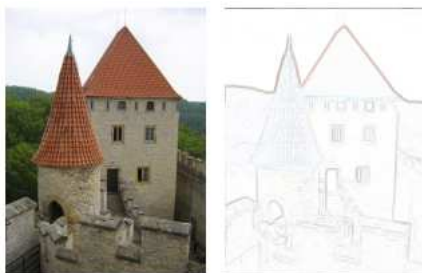
Obrázek H.12: Sépia



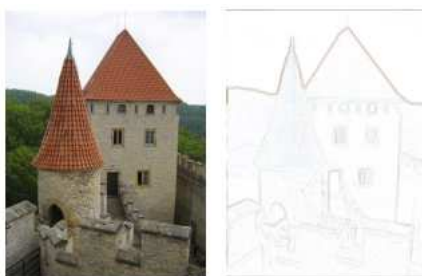
Obrázek H.13: Sobelův konvoluční operátor (výsledek je z důvodu přehlednosti negativ)



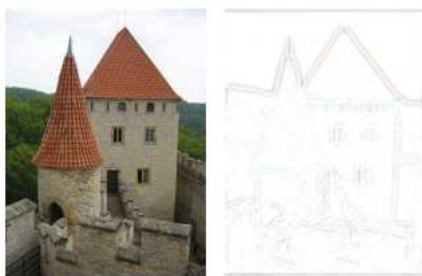
Obrázek H.14: Prewittové konvoluční operátor (výsledek je z důvodu přehlednosti negativ)



Obrázek H.15: Kirschův konvoluční operátor (výsledek je z důvodu přehlednosti negativ)



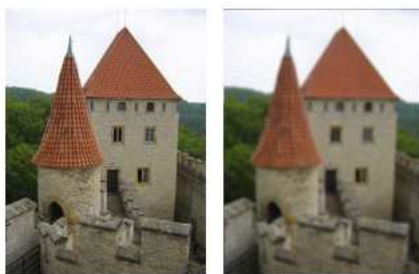
Obrázek H.16: Robinsonův konvoluční operátor (výsledek je z důvodu přehlednosti negativ)



Obrázek H.17: Laplacian of Gaussian s maskou o velikosti 9×9 (výsledek je z důvodu přehlednosti negativ)



Obrázek H.18: Gaussián



Obrázek H.19: Průměrování

Literatura

- [1] Agoston, M. K.: *Computer Graphics and Geometric Modeling: Implementation and Algorithms*. Springer, 2005, ISBN 978-1852338183.
- [2] Black, P. E.: *Algorithms and Theory of Computation Handbook (Dictionary of Algorithms and Data Structures)*. CRC Press, U.S. National Institute of Standards and Technology, 1999.
URL <http://www.nist.gov/dads/HTML/feasiblereg.html>
- [3] Campbell, J.: Robotic Motion Control: How Special Is It? *Robotics Online*, 2 2008.
- [4] Cyberbotics: EPFL education robot e-puck. 2007.
URL <http://www.e-puck.org/>
- [5] Cyberbotics: Robotický simulátor Webots. 2007.
URL <http://www.cyberbotics.com/>
- [6] Dudek, G.; Jenkin, M.: *Computational Principles of Mobile Robotics*. Cambridge University Press, 2000, ISBN 0-521-56876-5.
- [7] Goldberg, D. E.: *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989, ISBN 0-201-15767-5.
- [8] Koza, J. R.: *Genetic Programming: On the Programming of the Computers By Means of Natural Selection*. The MIT Press, 1992, ISBN 0-262-11170-5.
- [9] Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolution Programs, 3rd edition*. Springer, 1999, ISBN 3-540-60676-9.
- [10] Miller, B. L.; Goldberg, D. E.: Genetic Algorithms, Tournament Selection, and the Effects of Noise. *Complex Systems*, ročník 9, 1995: s. 193–212.
URL citeseer.ist.psu.edu/article/miller95genetic.html

- [11] Nelson, A.; Grant, E.; Barlow, G.; aj.: A Colony of Robots Using Vision Sensing and Evolved Nueral Controllers. 2003.
URL citeseer.ist.psu.edu/nelson03colony.html
- [12] Pratt, W. K.: *Digital Image Processing*. John Wiley and Sons, Inc., 2001, ISBN 0-471-37407-5.
- [13] Žára, J.; Beneš, B.; Sochor, J.; aj.: *Moderní počítačová grafika*. Computer Press, 2004, ISBN 80-251-0454-0.
- [14] Selig, J. M.: *Introductory to robotics*. Prentice Hall, 1992, ISBN 978-0134888750.
- [15] Shapiro, L. G.: *Computer Vision*. Prentice Hall, 2001, ISBN 978-0130307965.
- [16] Sonka, M.; Hlavac, V.; Boyle, R.: *Image Processing, Analysis and Machine Vision*. Thomson, 2008, ISBN 978-0-495-08252-1.
- [17] Stanley, K. O.; Miikkulainen, R.: Evolving Nerual Networks with Augmenting Topologies. *MIT Press*, 2002.
- [18] Toub, S.: Sepia Tone, StringLogicalComparer, and More. *MSDN Magazine: The Microsoft Journal for Developers*, January 2005.
URL <http://msdn.microsoft.com/msdnmag/issues/05/01/NETMatters/>
- [19] Wikipedia: Gamma correction, Wikipedia. 2008.
URL http://en.wikipedia.org/wiki/Gamma_correction/
- [20] Xie, M.: *Fundamentals of Robotics: Linking Perception to Action*. World Scientific Pub Co Inc, 2003, ISBN 978-9812383358.