INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Návrh a implementace komunikačního protokolu a nadstavbového GUI pro mobilního robota Robotino Festo

Diplomová práce

Bc. Petr Brant

Liberec 2012

Tento materiál vznikl v rámci projektu ESF (CZ.1.07/2.2.00/07.0247)
Reflexe požadavků průmyslu na výuku v oblasti automatického řízení a měření,
KTERÝ JE SPOLUFINANCOVÁN EVROPSKÝM SOCIÁLNÍM FONDEM A STÁTNÍM ROZPOČTEM ČESKÉ REPUBLIKY.

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: N2612 – Elektrotechnika a informatika
Obor: 1802T007 – Informační technologie

Návrh a implementace komunikačního protokolu a nadstavbového GUI pro mobilního robota Robotino Festo

Design and implementation of communication protocol and GUI interface for a mobile robot Robotino Festo

Diplomová práce

Autor: **Bc. Petr Brant**
Vedoucí práce: Ing. Jan Strnad
Konzultant: –

V Liberci 13. května 2012

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: 2011/2012

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Petr Brant**
Osobní číslo:	**M10000218**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Informační technologie**
Název tématu:	**Návrh a implementace komunikačního protokolu a nadstavbového GUI pro mobilního robota Robotino Festo**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Student podá přehled o současném stavu řešené problematiky.

2. Jádrem práce bude návrh obecného komunikačního protokolu pro ovládání mobilního robota, implementace tohoto návrhu pro Robotino Festo a tvorba nadstavbového GUI pro řízení.

3. Využití algoritmu počítačového vidění pro řešení prezentační úlohy.

4. Použité metody řešení budou obsahovat podrobný popis jednotlivých kroků.

5. Student stručně a konkrétně zhodnotí dosažené výsledky, jak byl splněn cíl práce a navrhne využití výsledků v praxi.

Rozsah grafických prací: **dle potřeby dokumentace**

Rozsah pracovní zprávy: **cca 40–50 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] AForge.NET. AForge.NET Framework [online]. 2011 [cit. 2011-10-05]. Dostupné z WWW: <http://www.aforgenet.com/framework/docs/>.

[2] GARAGE, Willow. OpenCV v2.3 documentation [online]. 2011, Last updated on Aug 17, 2011 [cit. 2011-10-05]. Dostupné z WWW: <http://opencv.itseez.com/>.

[3] NAGEL, Christian; EVJEN, Bill. C# 2008 : Programujeme profesionálně. 1. vydání. Praha : Computer Press, 2009. 1904 s. ISBN 978-80-251-2401-7.

[4] FESTO. Robotino : Manual. Germany [CD-ROM] : [s.n.], 11/2009. 100 s.

[5] REC GmbH. Rec::robotino::com API documentation [online]. [s.l.] : [s.n.], 15.2.2011 [cit. 2011-10-05]. Dostupné z WWW: <http://doc.openrobotino.org/documentation/OpenRobotinoAPI/1/doc/rec_robotino_com/>.

Vedoucí diplomové práce: **Ing. Jan Strnad**
Ústav mechatroniky a technické informatiky

Datum zadání diplomové práce: **14. října 2011**

Termín odevzdání diplomové práce: **18. května 2012**

prof. Ing. Václav Kopecký, CSc.
děkan

Technická univerzita v Liberci - 14 -

doc. Ing. Petr Tůma, CSc.
vedoucí ústavu

V Liberci dne 14. října 2011

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum

Podpis

Poděkování

Na tomto místě bych chtěl co nejupřímněji poděkovat panu Ing. Janu Strnadovi za jeho odborné vedení diplomové práce a usměrňování toku mých myšlenek. Chtěl bych mu poděkovat za jeho čas i rady, bez kterých by tato práce nebyla tím, čím je. Dále bych chtěl poděkovat panu doc. Mgr. Ing. Václavu Zádovi, CSc. za umožnění přístupu do robotické laboratoře. V neposlední řadě bych chtěl poděkovat své rodině a přátelům za celkovou podporu v mém úsilí.

Petr Brant

Abstrakt

Obsahem této diplomové práce je seznámení s robotem Robotino od firmy Festo a současným stavem problematiky návrhu univerzálního komunikačního protokolu pro mobilní roboty. V první části této práce seznámím čtenáře s mobilní platformou, která byla použita pro testování navrhovaného řešení. V této části také rozeberu jednotlivé senzory a efektory robota, abych mohl ve druhé části provést analýzu potřeb pro univerzální komunikační protokol a navrhnout tak jeho odpovídající strukturu. Druhá polovina této části je věnována implementaci navrhovaného protokolu v jazyce C#. Ve třetí části je krátce popsána idea a implementace ovládacího programu pro mobilní roboty, který implementuje jednotný komunikační protokol. Poslední část popisuje dvě prezentační úlohy na spojení robotiky a počítačového vidění a jednu úlohu zaměřenou na problematiku plánování.

Klíčová slova

Mobilní robot, Komunikační protokol, PDDL, Počítačové vidění, Plánování

Abstract

The content of this thesis is to introduce the robot Robotino from Festo and the current state of development problems of a universal communication protocol for mobile robots. In the first part of this work readers are acquainted with a mobile platform that was used to test the proposed solution. This section also will discuss the various sensors and effectors of the robot to be able to in the second part of an assessment of needs for universal communication protocol and design the appropriate structure. The second half of this section is devoted to the implementation of the proposed protocol in C#. The third part briefly describes the idea and implementation of the control program for mobile robots, which implements a single communication protocol. The last part describes two presentations on the role of combination of robotics and computer vision, and one focused on the role of planning issues.

Keywords

Mobile robot, Communication protocol, PDDL, Computer vision, Planning

Obsah

Seznam symbolů a zkratek

ASCII American Standard Code for Information Interchange

CF CompactFlash

CMOS Complementary Metal-Oxide-Semiconductor

CMY(K) Cyan-Magenta-Yellow(-Key, Black)

CV Computer Vision

GUI Graphical User Interface

HSL Hue-Saturation-Lightness

HSV Hue-Saturation-Value

HTML HyperText Markup Language

I/O Input/Output

I^2C Inter-Integrated Circuit

LPT Line Print Terminal

PDDL Planning Domain Definition Language

RGB Red-Green-Blue

SDRAM Synchronous Dynamic Random-Access Memory

STRIPS Stanford Research Institute Problem Solver

TCP/IP Transmission Control Protocol / Internet Protocol

UDP User Datagram Protocol

UML Unified Modeling Language

USB Universal Serial Bus

VGA Video Graphics Array

XML Extensible Markup Language

XSL Extensible Stylesheet Language

XSLT Extensible Stylesheet Language Transformations

Seznam obrázků

Seznam tabulek

Seznam ukázek kódu

Kapitola 1

Úvod

Když v roce 1920 v knize R.U.R. poprvé použil Karel Čapek slovo robot, neměl představu, jak mohutný rozvoj bude prodělávat tato oblast v dnešní době. Celý svět je dnes zaplněn nejrůznějšími roboty – pomocníky. Roboti[1] nachází velké uplatnění v širokém množství různých oborů – mohou totiž opakovat jednu a tu samou činnost téměř bezchybně a s velkou rychlostí. Roboti nemají výhrady, když je jim přidělena namáhavá práce, nechodí na dovolené, nejsou nemocní. Jsou to prostě univerzální pomocníci v celém širokém spektru oborů. V armádě likvidují nebezpečnou munici, v automobilovém průmyslu svařují karoserie budoucích automobilů, v lékařství je možné nechat se robotem dokonce operovat.

V dnešní době se výrobou robotů zabývá mnoho firem a každá firma dodává ke svým robotům příslušný ovládací program. Při změně robota je nutno přejít na jiný ovládací software, který je samozřejmě nekompatibilní se softwarem jiných firem (v horším případě je nutno přepracovat všechny napsané algoritmy tak, aby byly kompatibilní s tímto novým ovládacím softwarem).

Cílem této práce je navrhnout a implementovat jednotný komunikační protokol a příslušný ovládací program, který by byl schopen zajistit ovládání různých typů mobilních robotů různých výrobců. Dalším cílem je vytvoření několika prezentačních úloh, které využívají algoritmů pro zpracování obrazu a mohou tak prezentovat životaschopnost komunikačního protokolu. Zároveň tyto úlohy mohou zvednout zájem uchazečů právě o studium robotiky. Populárním tématem je umělá inteligence – *věda o vytváření strojů nebo systémů, které budou při řešení určitého úkolu užívat takového postupu, který – kdyby ho dělal člověk – bychom považovali za projev jeho inteligence* (Marvin Minsky, 1967 – český překlad Roman Barták, 2011). V této práci není možné obsáhnout veškerou problematiku umělé inteligence a proto se zaměřím na podčást umělé inteligence a budu se zabývat plánováním. Posledním cílem této práce je podání stručného vysvětlení základů plánování a jazyka pro

[1] V českém jazyce je možné psát roboti i roboty. Podle Internetové jazykové příručky Ústavu pro jazyk český(2012) bychom pro stroj podobný člověku měli zásadně používat rod mužský životný, pro stroj který není podobný člověku jsou možné obě varianty životnosti.

plánování úloh PDDL, popis problému prostřednictvím tohoto jazyka a za pomoci plánovače zdárné řešení zadané úlohy zaměřené na plánování.

Na počátku práce je důležité se zeptat, zda autor touto prací nebude objevovat již objevené věci a nebude znovu přemýšlet nad již vyřešenými problémy. Po prohledání dostupných databází, které má Technická univerzita v Liberci zakoupeny – především IEEExplore, IEEE Computer Society Digital Library a ACM Digital library a pomocí fulltextového vyhledávání bylo nalezeno několik velmi zajímavých článků o mobilních robotech, žádný z nich se však nezabýval návrhem univerzálního komunikačního protokolu. Výsledkem bylo zjištění, že tato problematika nebyla dosud zpracována.

Kapitola 2

Festo Robotino

Aby mohl být úspěšně navržen a implementován komunikační protokol je nutné seznámit se s robotem – s jeho hardwarem i softwarem. K hlubšímu prostudování poslouží robot od firmy Festo – Robotino, zakoupený Technickou univerzitou v Liberci z prostředků Evropských sociálních fondů v rámci operačního programu Vzdělávání pro konkurenceschopnost.

Obrázek 2.0.1: Festo Robotino – obrázek převzat z oficiální dokumentace k robotovi (Festo Didactic, 2009)

2.1 Hardwarové vybavení

Robotino je po hardwarové stránce vybaveno velmi dobře. Robot se skládá ze dvou součástí – těla (podvozku) tvořeného nerezavějící ocelí a velitelského můstku[1].

Tělo robota slouží pro uložení tří stejnosměrných motorů (navzájem otočených o 120°) se senzorem otáček každého motoru, dvou 12V baterií pro zajištění napájení, nárazového senzoru, devíti senzorů na měření vzdáleností (otočených navzájem o 40°) a barevné kamery. Jako doplňkové senzory je možné přikoupit a na tělo namontovat indukční senzor pro detekci kovových předmětů položených na podlaze (podle nich je možné řídit pohyb robota). Dalším doplňkovým senzorem pro řízení pohybu je dvojice difúzních senzorů pro zjišťování odrazivosti konkrétního materiálu, který se nachází pod senzorem.

Složitějším senzorem je potom takzvaný „Northstar“ senzor (v překladu „Northstar“ znamená „Polárka“). Tento senzor umožňuje v absolutních souřadnicích zjišťovat polohu robota s velkou přesností.

Některé senzory vrací své údaje již v digitální formě (například Northstar senzor se připojuje přes sběrnici USB), jiné vracejí své naměřené hodnoty jako funkci napětí. Pojďme se na všechny tyto senzory trochu blíže podívat a prozkoumat jejich význam i charakteristiky. Čerpáno z (2009).

2.1.1 Senzor otáček motoru

Každý ze tří motorů je vybaven snímačem otáček. Snímač pracuje na principu zachycování světelných pulsů, které projdou děrovaným kolečkem – stejného principu se používá i u kolečka počítačové myši. Uvnitř senzoru jsou umístěny dva takovéto detektory otočeny o 90°, aby bylo možné určit směr otáčení. Na základě znalosti přesných otáček motoru a požadovaných otáček můžeme upravit nastavení kontroléru motoru. Čerpáno z Festo Didactic (2009).

2.1.2 Indukční senzor

Indukční senzor je založen na principu vzájemného působení mezi kovovým vodičem a střídavým elektromagnetickým polem, které je generováno senzorem. Přítomnost kovového materiálu (na nekovové materiály tento senzor nereaguje) se rozpoznává podle úbytku energie elektromagnetického pole, které je způsobeno vznikem vířivých proudů v materiálu. Tento úbytek je možné měřit a pomocí vyhodnocovací logiky určovat přítomnost či nepřítomnost kovového materiálu (indukční senzor ve funkci spínače), nebo vracet analogovou hodnotu (napětí nebo proud), která určuje míru přítomnosti kovového materiálu. Čerpáno z materiálu firmy Balluf CZ (2011). Robotino používá indukční senzor SIEA-M12B-UI-S (zobrazen na obrázku 2.1.1 na následující straně), který vrací napětí v závislosti na vzdá-

[1] V anglické dokumentaci k robotovi je velitelský můstek nazván „Command bridge“.

lenosti kovového předmětu (tento senzor měří v rozsahu 0 mm – 6 mm). Indukční senzor se připojuje k I/O portu velitelského můstku. Čerpáno z Festo Didactic (2009).

Obrázek 2.1.1: Indukční senzor (převzato z Vision-supplies.com)

2.1.3 Difúzní senzor

Difúzní senzor snímá odraz vyzařovaného světla (většinou se používají senzory pro zachycení odrazu viditelného červeného světla nebo infračerveného světla) a převádí míru odrazu na napětí. Odraz je pro každý materiál a pro každou barvu konkrétního materiálu jiný. Na základě změřené míry odrazu je tak možné určit konkrétní materiál, případně jeho barvu. Difúzní senzor se dá snadno použít pro navigování robota pouze na základě odrazu. Robotino používá dva difúzní senzory pracující s červeným světlem, senzory se připojují podobně jako indukční senzor k I/O portu velitelského můstku. Čerpáno z Festo Didactic (2009).

2.1.4 Nárazový senzor

Tento senzor (dlouhý gumový profil) obepíná celé tělo robota kolem dokola a reaguje na své stlačení. Pokud je robot nastaven na takzvaný „bezpečný“ režim, vyvolá náraz nebo najetí do překážky okamžité zastavení pohybu robota a v případě vnitřního programu dojde k úplnému zastavení jeho vykonávání. Pokud má robot „bezpečný“ režim vypnutý, je aktivace senzoru pouze indikována vyšší vrstvě a případné zastavení je ponecháno na ní. Čerpáno z Festo Didactic (2009).

2.1.5 Vzdálenostní senzor

Robot je vybaven celkem devíti vzdálenostními senzory, které jsou v pravidelném intervalu 40° rozmístěny po celém obvodu těla robota. Vzdálenostní senzory pracují na principu odrazu infračerveného světla od předmětu zpět do detektoru. Napětí na výstupu je pak funkcí vzdálenosti překážky od senzoru. Čerpáno z Festo Didactic (2009).

Robotino používá jako vzdálenostní senzor výrobek firmy Sharp, model s označením GP2D120XJ00F. Podle dokumentace firmy Sharp (2005) k tomuto výrobku je dosah tohoto

senzoru od 4 cm do 30 cm a rozdíl napětí při minimální a maximální naměřené vzdálenosti je v typickém případě 2,25 V. Dle dokumentace není téměř rozdíl v naměřených hodnotách mezi materiály s různou hodnotou odrazivosti materiálu. V souboru s dokumentací je také přiložen graf zobrazující funkci napětí ve vztahu ke vzdálenosti (z grafu je vidět, že funkce není lineární), ale nejsou zde uvedeny přesné změřené hodnoty (ani není uvedena aproximační funkce).

Pro odstranění nedostatku dokumentace a získání potřebných hodnot, bylo nutné provést měření. Bylo změřeno 100 hodnot napětí (pro vyloučení možných chyb) pro každou vzdálenost od 4 cm do 46 cm, v kroku po 2 cm. Odraznou deskou byla šedá umělohmotná plocha dostatečně velkého rozměru. Změřené výsledky jsem vyfiltroval (do výsledku nebylo započteno 5 nejmenších a 5 největších změřených hodnot napětí) a ze zbylých 90 jsem spočítal aritmetický průměr. Naměřené, vyfiltrované a zprůměrované výsledky jsou umístěny v tabulce 2.1.1. Surová naměřená data se potom nachází na přiloženém médiu (v jednoduché struktuře formátu XML). Z těchto hodnot jsem také vytvořil graf (zobrazen v příloze na obrázku A.0.1 na straně 68) a nechal jsem propočítat koeficienty polynomu aproximační funkce - pro různé stupně polynomu. Aproximační polynomy jsou vypsány v tabulce 2.1.2 na následující straně, pro praktický přepočet je autorem použit polynom šestého stupně – v tabulce vyznačen žlutou barvou – protože poměrně dobře aproximuje původní data. Vypočtený polynom je potom použit pro výpočet vzdálenosti z naměřených hodnot napětí.

Tabulka 2.1.1: Naměřené výsledky senzoru vzdáleností

Vzdálenost [cm]	Napětí [V]	Vzdálenost [cm]	Napětí [V]
4	2,548	26	0,422
6	1,909	28	0,381
8	1,456	30	0,343
10	1,155	32	0,321
12	0,970	34	0,301
14	0,824	36	0,281
16	0,724	38	0,261
18	0,643	40	0,241
20	0,582	42	0,222
22	0,513	44	0,206
24	0,469	46	0,193

Tabulka 2.1.2: Aproximační polynomy pro výpočet vzdálenosti z napětí vzdálenostního senzoru

Stupeň polynomu	Polynom
0.	25
1.	$-17,965x + 37,22$
2.	$14,484x^2 - 53,04x + 49,269$
3.	$-12,362x^3 + 62,03x^2 - 100,07x + 59,902$
4.	$11,225x^4 - 70,366x^3 + 157,78x^2 - 156,46x + 69,335$
5.	$-12,189x^5 + 88,073x^4 - 242,68x^3 + 324,63x^2 - 223,86x + 78,16$
6.	$16,331x^6 - 131,07x^5 + 416,18x^4 - 675,07x^3 + 606,04x^2 - 307,92x + 87,043$
7.	$-17,883x^7 + 161,70x^6 - 598,46x^5 + 1179,4x^4 - 1350,9x^3 + 926,48x^2 - 382,37x + 93,546$

Tabulka 2.1.3: Rozlišení kamery pro snímání videa

160×120, 30 snímků/s	176×144, 30 snímků/s	320×240, 30 snímků/s
352×288, 30 snímků/s	640×480, 15 snímků/s	

2.1.6 Kamera

Robotino je vybaveno barevnou CMOS kamerou, díky které je možné snímat snímky reálného světa a za pomoci algoritmů počítačového vidění vykonávat různé činnosti. Kamera má nastavitelnou výšku i sklon a může snímat v několika různých rozlišeních, při různé snímkovací frekvenci. K Robotinu se kamera připojuje prostřednictvím USB portu (zde aktuálně ve verzi 1.1). Tabulka 2.1.3 uvádí přehled možných rozlišení kamery pro snímání videa. Preferované rozlišení je v tabulce podbarveno žlutou barvou. Na obrázku 2.1.2 na následující straně je zobrazen reálný snímek z kamery. Zpracováno podle Festo Didactic (2009).

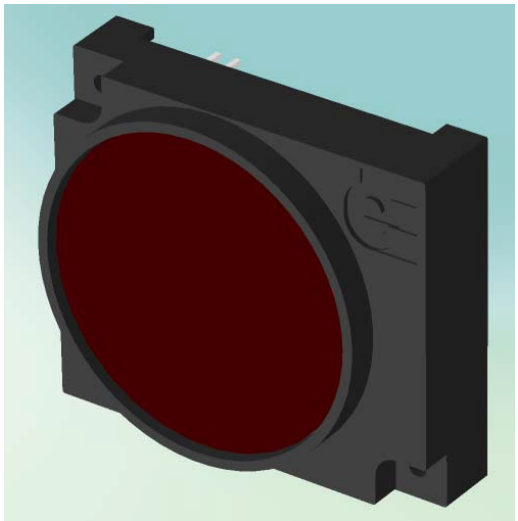
2.1.7 Northstar senzor

Pomocí senzoru otáček na každém motoru je možné určit polohu robota – ovšem pouze polohu relativní. K určení absolutní polohy je nutno znát počátek souřadnic, a to jen pomocí senzoru otáček motoru není možné. K určení absolutní polohy je zapotřebí použít Northstar senzor. Tento senzor pracuje na principu statického vysílače infračervených paprsků, které tvoří na stropě speciální vzor a pohyblivého přijímače, který umí na základě odrazu infračervených paprsků od stropu určit svou polohu.

Obrázek 2.1.2: Reálný snímek z kamery (na obrázku se nachází robot iRobot Create)

Občas bývá zapotřebí určit nejen polohu v aktuální místnosti, ale i konkrétní místnost, ve které se senzor nachází. Vysílač je proto vybaven přepínačem místností a je tak možné rozpoznat až 9 různých místností. Senzor rozpoznává konkrétní místnost podle frekvence „blikání“ rozpoznávacího vzoru. Frekvence je přesně dána tabulkou, která se nachází v dokumentaci Evolution Robotics (2008a). Zjednodušené vyobrazení projektoru i detektoru se nachází na obrázku 2.1.3 a 2.1.4.

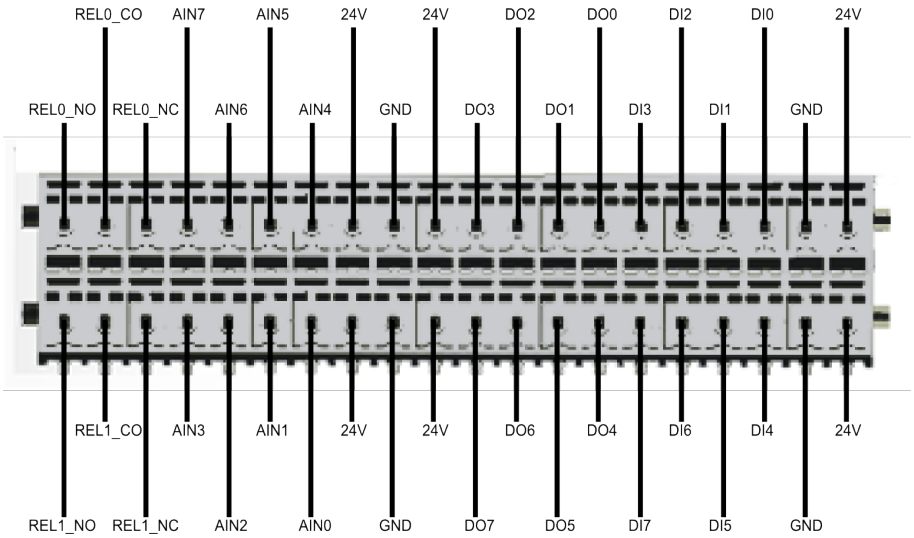
Obrázek 2.1.3: Northstar projektor – převzato z Evolution Robotics (2008a)



Obrázek 2.1.4: Northstar detektor – převzato z Evolution Robotics (2008b)

2.1.8 I/O port

Velitelský můstek má na sobě vstupně-výstupní port, kam je možné připojit další senzory nebo i efektory (například pohyblivé rameno). Standardně je k dispozici 8 analogových vstupů (0 V – 10 V), 8 digitálních vstupů (logická 1 má úroveň 8,5 V – 24 V, logická 0 má úroveň 0 V – 5,7 V), 8 digitálních výstupů (logická 1 má úroveň 24 V, logická 0 má úroveň 0 V, maximální povolený odběr je 0,3 A) a 2 reléové spínače (výstupní napětí je mezi 24 V – 30 V (dle nabití baterie), maximální odběr je 0,5 A - výstup není chráněn proti zkratu) pro napájení přídavných efektorů. Vstupy a výstupy je možno každý zvlášť softwarově ovládat a rozšířit tak standardní nabídku senzorů a efektorů o nové prvky. Rozložení vstupů a výstupů je možné nalézt na obrázku 2.1.5.



Obrázek 2.1.5: I/O port - přepracováno podle Festo Didactic (2009)

2.1.9 Velitelský můstek

Hlavní součástí velitelského můstku je řídící jednotka. Tato jednotka se skládá z jednodeskového počítače, compact flash (CF) karty o velikosti minimálně 1GB a bezdrátové síťové karty.

Jednodeskový počítač je typu MOPSlcdVE (výrobcem je firma Kontron), je vybaven 300MHz procesorem VIA Eden, 128MB SDRAM, dvěma porty USB verze 1.1 (tyto porty jsou vyvedeny ven pro připojení senzorů – například kamery), 10/100BaseT ethernetovým připojením (konektor je rovněž vyveden ven), VGA konektorem pro připojení monitoru (také vyveden ven). Deska dále disponuje dvěma porty RS-232, jedním LPT portem, portem pro připojení ke sběrnici I^2C a portem pro připojení disketové mechaniky, které však nejsou vyvedeny ven pro použití.

CF karta obsahuje upravený operační systém Linux (použitým základem je Ubuntu). Operační systém je dovybaven knihovnami, které umožňují vzdálené řízení robota přes

bezdrátovou síť nebo přímým připojením klávesnice a monitoru k robotovi. Zpracováno podle Festo Didactic (2009); Kontron (2004).

2.2 Softwarové vybavení

Robotino kromě základního operačního systému – viz část 2.1.9 na předchozí straně – obsahuje i několik předpřipravených demo programů. Z těchto programů jsem se inspiroval pro napsání vlastních algoritmů, které budou využívat navrhovaného komunikačního protokolu. Předpřipravené programy jsou shrnuty v tabulce 2.2.1. Údaje pro tabulku čerpány z Festo Didactic (2009).

Tabulka 2.2.1: Demonstrační programy

Circle	Robotino se bude po dobu 10 vteřin pohybovat po kruhové dráze o průměru 1 m.
Forward	Robotino se bude po dobu 10 vteřin pohybovat rovně a ujede dráhu 1 m.
Quadrangle	Robotino se bude neustále pohybovat po čtverci o hraně 1 m.
Roaming	Robotino se bude pohybovat rovně a bude se vyhýbat překážkám.
Follow line	Robotino se bude pohybovat po čáře, kterou detekuje pomocí kamery. Čára by měla mít černou, červenou nebo modrou barvu a měla by být alespoň 5cm široká.

Kapitola 3

Návrh komunikačního protokolu

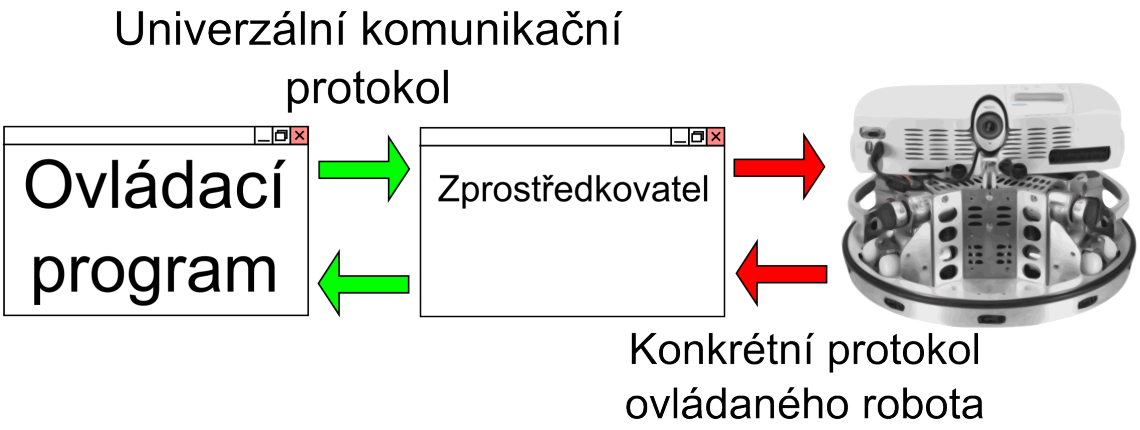
Z přehledu senzorů a efektorů předchozí kapitoly plyne velká část požadavků, které bude muset komunikační protokol splňovat. Představený robot je však jen jedním z mnoha různých typů mobilních robotů. Někteří roboti nemusí pro pohyb využívat kola, ale mohou místo nich mít pásy, vrtule nebo nohy. Také mohou používat odlišné senzory a efektory – někteří roboti mohou mít například několik kamer, nebo naopak mohou být bez optické kamery. Všechny tyto eventuality v rozdílu robotů je nutno zpracovat a vyvodit z nich nějaké důsledky pro návrh obecného komunikačního protokolu.

Proto při tvorbě této práce spolupracuji se studenty Tomášem Němečkem a Martinem Sobotkou (oba studenti Technické univerzity v Liberci), kteří mají jako svou bakalářskou práci ovládání jiného robota - iRobot Create. Shrnul jsem jejich připomínky a přání, co by komunikační protokol měl umět, aby mohli plně ovládat svého robota a zahrnul jsem toto do návrhu.

3.1 Základní idea návrhu

Základní ideou návrhu bylo dokonalé odstínění ovládacího programu od ovládaného robota. Robot by neměl vědět čím je ovládán, stejně jako ovládací program by neměl vědět, jakého robota ovládá. Veškerá komunikace musí probíhat pomocí konkrétních zpráv a není tak možné si „cokoli“ dovozovat ze znalosti konkrétního robota nebo ovládacího programu. Tento přístup na jednu stranu ztěžuje implementaci komunikačního protokolu, ale na druhou stranu nabízí největší míru svobody ve výběru ovládaného robota i ovládacího programu. Základní schéma komunikace ovládacího programu je zobrazeno na obrázku 3.1.1 na následující straně. Na tomto obrázku můžeme vidět ovládací program, který přímo neovládá robota, ale zasílá své požadavky zprostředkujícímu programu, který překládá zprávy univerzálního protokolu do ovládacího protokolu cílového robota. Aby mohl být ovládán jiný robot, je zapotřebí pouze napsat právě takový program, který by prováděl tento překlad. Ovládací program i komunikační protokol může zůstat naprosto stejný.

V předchozím textu jsem použil výraz „zpráva“. Takovou zprávu si můžeme představit jako obecný příkaz robotovi - například „Posouvej se rychlostí 100 mm/s rovně“. A záleží na každém konkrétním robotovi, jak na tuto zprávu zareaguje. Robot vybavený kolečky zapne motor na určitých otáčkách a začne popojíždět. Chodící robot bude rytmicky zvedat, posouvat a spouštět nohy.



Obrázek 3.1.1: Základní idea komunikace

3.2 Zpráva

Zpráva je základním objektem komunikačního protokolu. V počítačovém světě mohou být zprávy dvojího druhu – textové (čitelné i pro běžného člověka), nebo binární (pro běžného člověka nečitelné). Návrh umožňuje si z těchto dvou variant vybrat a použít tu, která je pro danou situaci vhodnější. Tabulka 3.2.1 na následující straně porovnává obě tyto možnosti na základě různých kritérií. Zprávy v textové formě jsou vytvořeny pomocí vlastního jazyka postaveného nad XML, kde jazyk je definován pomocí XML schématu (XML schéma je přiloženo na médiu). Zprávy v binární podobě si zachovávají stromovou strukturu (protože je to výhodné při zachovávání zpětné kompatibility verzí). Podrobný popis převodu zprávy na binární formu naleznete v kapitole 3.3.2 na straně 34, která se věnuje implementační části komunikačního protokolu. Každá zpráva se skládá z hlavičky zprávy, příkazu zprávy a parametrů příkazu. Obecné schéma textové formy zprávy je zobrazeno v ukázce kódu 3.2.1 na následující straně.

Tabulka 3.2.1: Porovnání textové a binární formy zpráv

Textová forma	+ zprávy jsou snadno čitelné i upravovatelné
	+ výhodnější pro ladění
	+ má samopopisnou strukturu
	— zprávy jsou delší a budou se déle přenášet
Binární forma	+ zprávy jsou kratší, jejich přenos je rychlejší
	— nelze je běžně číst, ani upravovat
	— nemá samopopisnou strukturu a je nutné dodržovat přesnou formu

```
<RMessage>
  <header>...</header>
  <message>
    <command>...</command>
    <params>...</params>
  </message>
</RMessage>
```

Ukázka kódu 3.2.1: Obecné schéma zprávy v textovém formátu

3.2.1 Hlavička zprávy

Každá zpráva začíná hlavičkou, která obsahuje základní údaje o zprávě. Jednotlivá pole se v hlavičce vyskytují v totožném pořadí, jako v tabulce 3.2.2 na následující straně. Pokud má být zpráva zpracovávaná v nějakém daném intervalu, je možné doplnit hlavičku o část, která je uvedena v tabulce 3.2.3 na následující straně. Zprávu tak není nutné zasílat při každé opakované žádosti, ale zpráva je automaticky na přijímači zpracována a případná odpověď je automaticky vyslána. Ukázka správně sestavené hlavičky zprávy v textovém formátu je ve výpisu kódu 3.2.2 na následující straně.

Tabulka 3.2.2: Datová pole v hlavičce zprávy

Název identifikátoru	Datový typ	Význam
id	4B integer	Identifikátor zprávy. Každou zprávu je tak možno v systému identifikovat a určit zda nedošlo například k výpadku nějaké zprávy.
protocolVersion	4B integer	Udává verzi protokolu, podle které je zpráva vytvořena. Přijímač, který zachytí zprávu s vyšším číslem protokolu, nemusí tuto zprávu zpracovávat.
priority	4B integer	Udává prioritu zprávy. Zprávy, které mají nastavenou vyšší prioritu, by měly být ve zpracovávání upřednostněny před zprávami s prioritou nižší.
type	textový řetězec	Udává typ zprávy. Na výběr jsou 3 typy zpráv: • **request** – zpráva obsahuje požadavek na zpracování • **answer** – zpráva obsahuje odpověď na zpracování • **error** – zpráva obsahuje nějakou chybovou informaci

Tabulka 3.2.3: Nastavení opakování zprávy

Název identifikátoru	Datový typ	Význam
interval	4B integer	Nastavení hodnoty intervalu. Číslo udává, po kolika milisekundách se má zpráva zopakovat.
numberOfRepeats	4B integer	Hodnota udává kolikrát má být zpráva opakována. Pokud je uvedeno záporné číslo, pak je zpráva opakována nekonečně-krát.

```
...
<header>
  <id>485</id>
  <protocolVersion>2</protocolVersion>
  <priority>15</priority>
  <type>request</type>
  <repeatControl>
    <interval>1500</interval>
    <numberOfRepeats>5</numberOfRepeats>
  </repeatControl>
</header>
...
```

Ukázka kódu 3.2.2: Příklad hlavičky zprávy v textovém formátu

3.2.2 Příkaz zprávy

V příkazu zprávy definujeme co má robot vykonat. V případě textového režimu je příkaz zabalen uvnitř tagu <command>. Název příkazu je vždy textový a je to výběr jednoho příkazu z tabulky 3.2.4.

Tabulka 3.2.4: Přehled příkazů zprávy

Název příkazu	Význam
init	Zpráva s příkazem init slouží pro inicializaci připojení k robotovi.
move	Zpráva s příkazem move říká robotovi, že se bude nějakým způsobem pohybovat.
sensorValue	Zpráva s příkazem sensorValue dává pokyn pro zjištění aktuálního stavu senzorů (v případě, že se jedná o request), nebo obsahuje změřené hodnoty ze senzorů (v případě, že se jedná o answer).
image	Zpráva obsahuje snímek z kamery.
startInnerFunction	Spustí nějakou předpřipravenou funkci přímo na robotovi. Používáním této zprávy do jisté míry klesá nezávislost na konkrétním robotovi, bohužel je nutné ji občas použít. Jedná se většinou o velmi specializované funkce – například iRobot umožňuje přehrávat melodie.
digitalOutput	Zpráva obsahuje pokyn pro zapnutí nebo vypnutí digitálního výstupu robota.
property	Speciální příkaz - více v kapitole 3.2.4 na straně 32.

3.2.3 Parametry příkazu

Příkazy mohou mít parametry, který jej blíže specifikují. Parametry například určují rychlost a směr pohybu nebo udávají adresu robota. Každý příkaz – viz kapitola o příkazech zprávy 3.2.2 – má trochu jiné parametry (i v závislosti na tom, zda se jedná o požadavek, nebo odpověď), kterými je možné jej přesněji specifikovat. V textovém režimu jsou parametry uzavřeny v tagu <params>. V následujících několika podsekcích jsou vypsány příkazy a jejich parametry.

Příkaz request init

Tabulka 3.2.5: Parametry příkazu request init

Název identifikátoru	Datový typ	Význam
address	textový řetězec	Adresa, na kterou se má zprostředkovávající program připojit. Adresa může být v různém tvaru v závislosti na typu adresy, který ji specifikuje.
addressType	textový řetězec	Specifikuje adresu. Adresa může být v některém z následujících formátů: • **Socket** – Adresa udává IP adresu robota. Připojení je realizováno socketem • **COMPort** – Adresa udává číslo COM portu, ke kterému je připojen robot • **SharedMemory** – Adresa udává oblast paměti, která slouží pro výměnu zpráv • **File** – Adresa udává název souboru, který slouží pro výměnu zpráv • **NamedPipe** – Využití nalezne především v systémech UNIXového typu

Příkaz answer init

Tabulka 3.2.6: Parametry příkazu answer init

Název identifikátoru	Datový typ	Význam
status	pravdivostní hodnota	Určuje, zda se podařilo připojit k robotovi. Hodnota **pravda** znamená úspěšné připojení k robotovi. Hodnota **nepravda** znamená, že se k robotovi nepodařilo připojit.
errorMessage	textový řetězec	Volitelná část – v případě chyby připojování zde může být chybová odpověď.

Příkaz request move

Tabulka 3.2.7: Parametry příkazu request move

Název identifikátoru	Datový typ	Význam
direction	4B integer	Udává směr pohybu. Hodnota směru musí být mezi 0° a 359°. Hodnota je vždy relativní k aktuálnímu směru pohybu robota.
speed	4B integer	Hodnota určuje rychlost pohybu robota. Rychlost se uvádí v milimetrech za sekundu.
angularSpeed	4B integer	Hodnota určuje rychlost otáčení robota. Rychlost se uvádí ve stupních za sekundu.

Příkaz request sensorValue

Nejprve je nutné si definovat přehled jednotlivých typů senzorů s jejich standardním typovým pojmenováním a standardní jednotkou, kterou budeme považovat pro daný typ senzoru za výchozí. Tato přesná specifikace je nutná proto, abychom mohli přistupovat k různým senzorům různých výrobců jednoho typu stejně. Převod na standardní jednotku pak bude v kompetenci programu zprostředkovávajícího komunikaci mezi robotem a ovládacím programem. Přehled je zobrazen v tabulce 3.2.8. Požadavek na čtení senzoru je pak ve tvaru dle ukázky kódu 3.2.3 na následující straně.

Tabulka 3.2.8: Typy senzorů a jejich standardní jednotky

Název	Typ senzoru	Standardní jednotka
Vzdálenostní senzor	**distancemeter**	centimetr (cm)
Nárazový senzor	**bumper**	bezrozměrné číslo[1], 0 znamená není aktivován, >0 znamená je aktivován + určení zóny aktivace
Rychloměr	**speedometer**	milimetr za sekundu (mm/s)
Napájení	**voltage**	nabití baterie vyjádřené v napětí (V)
Northstar	**northstar**	2×milimetr (mm) + 1×úhel ve stupních
IR senzor	**irsensor**	bezrozměrné číslo
Indukční senzor	**inductive**	bezrozměrné číslo, míra přítomnosti vyjádřená v procentech
Difúzní senzor	**diffuse**	bezrozměrné číslo, míra odrazu vyjádřená v procentech
Analogové vstup	**anologInput**	napětí ve voltech (V)
Digitální vstup	**digitalInput**	logická hodnota – **logická pravda** je reprezentována hodnotou 1, **logická nepravda** je reprezentována hodnotou 0.
Kamera	**camera**	Standardní jednotkou je obraz. Kamera má trochu zvláštní postavení, snímky z kamer jsou zasílány automaticky, není třeba o ně žádat tímto způsobem. V této tabulce je uvedena pouze pro sumarizaci senzorů.

```
...
<sensors>
  <sensor>
    <sensorType>Typ senzoru</sensorType><!-- typ senzoru -->
    <num>1</num><!-- cislo senzoru -->
    <num>3</num>
  </sensor>
  <sensor>
    <sensorType>bumper</sensorType>
    <num>1</num>
  </sensor>
  ...
</sensors>
...
```

Ukázka kódu 3.2.3: Požadavek na data ze senzorů

Příkaz answer sensorValue

Syntax odpovědi bude nejlépe vidět na příkladu kódu 3.2.4. Typ senzoru je stejný jako v tabulce 3.2.8 na předchozí straně. Hodnoty jsou uvedeny ve standardních jednotkách.

```
...
<sensorsVal>
  <sensor>
    <sensorType>distancemeter</sensorType>
    <sensorVal>
      <num>1</num>
      <value>150</value>
    </sensorVal>
    <sensorVal>
      <num>3</num>
      <value>400</value>
    </sensorVal>
  </sensor>
  <sensor>
    <sensorType>bumper</sensorType>
    <sensorVal>
      <num>1</num>
      <value>0</value>
    </sensorVal>
  </sensor>
  ...
</sensorsVal>
...
```

Ukázka kódu 3.2.4: Vrácená data ze senzorů

Příkaz answer image

Parametry odpovědi jsou zabaleny uvnitř tagu <image> (v případě textového režimu).

Tabulka 3.2.9: Parametry příkazu answer image

Název identifikátoru	Datový typ	Význam
cameraNumber	4B integer	Udává číslo kamery, ze které jsou obrazová data. Vhodné pro roboty s více kamerami.
dataType	textový řetězec	Udává v jakém datovém formátu jsou uložena obrazová data. Povolené formáty jsou **bmp**, **jpeg**, **png.**
width	4B integer	Hodnota určuje šířku obrázku.
height	4B integer	Hodnota určuje výšku obrázku.
imageType	textový řetězec	Udává zda jde o barevný obrázek (**colour**), nebo o hloubkovou mapu (**depth**).
imageData	pole bytů	Vlastní obrazová data. V textovém režimu jsou vstupní data zakódována pomocí Base64 kódování.

Příkaz request digitalOutput

Parametry jsou zabaleny uvnitř tagu <digitalOutput> (v případě textového režimu).

Tabulka 3.2.10: Parametry příkazu request digitalOutput

Název identifikátoru	Datový typ	Význam
outputNum	4B integer	Udává číslo digitálního výstupu.
funcParam	pravdivostní hodnota	**Pravda** znamená zapnutý výstup (maximální napětí), **nepravda** znamená vypnutý výstup (bez napětí).

Příkaz request startInnerFunction

Parametry jsou zabaleny uvnitř tagu <startInnerFunction> (v případě textového režimu).

Tabulka 3.2.11: Parametry příkazu request startInnerFunction

Název identifikátoru	Datový typ	Význam
funcName	textový řetězec	Udává název vnitřně volané funkce.
funcParam	pole bytů	Předává parametry vnitřně volané funkci (například melodii, která má být přehrána).

3.2.4 Zpráva vlastností

Z důvodů dříve psaných (kapitola 3.1 na straně 23) není ovládací program informován, jakého ovládá robota a nedisponuje tedy žádnou informací o senzorech a efektorech, kterými je daný robot vybaven. Aby byl ovládací program schopen zjistit vybavení robota (například počet dálkoměrů nebo kamer), zašle zprostředkujícímu programu zprávu s typem **request** s příkazem **property** a prázdnými parametry. Zprostředkující program odpoví zprávou typu **answer** s příkazem **property** a v parametrech budou uvedeny typy a počty jednotlivých senzorů. V odpovědní zprávě bude také výrobce a jméno robota (čistě pro identifikační účely – například pro zobrazení loga) a výpis všech vnitřních funkcí, které robot podporuje. Příklad parametrů odpovědní zprávy je v ukázce kódu 3.2.5. Po přijetí této zprávy ovládacím programem má tak ovládací program k dispozici informace o robotovi.

```
...
<params>
  <robotProperty>
  <robotProducer>Festo</robotProducer>
  <robotName>Robotino</robotName>
  <sensorsNum>
    <sensor>
      <sensorType>digitalInput</sensorType>
      <count>8</count>
    </sensor>
    ...
  </sensorsNum>
  <innerFuncNames>
    <innerFuncName>Beep</innerFuncName>
    <innerFuncName>PlayMelody</innerFuncName>
    ...
  </innerFuncNames>
  </robotProperty>
</params>
...
```

Ukázka kódu 3.2.5: Parametry zprávy vlastností

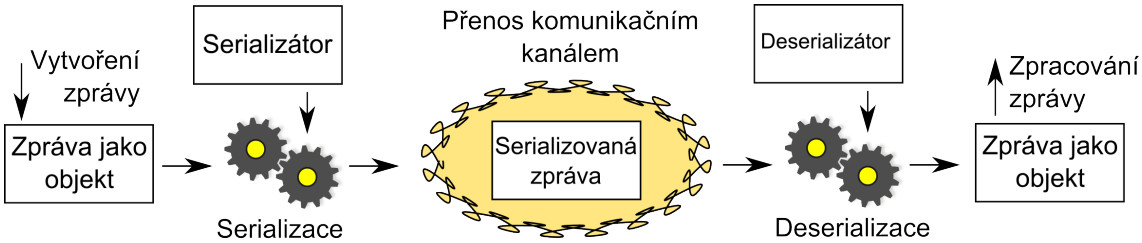
3.3 Implementace komunikačního protokolu

Pro implementaci výše uvedeného protokolu jsem zvolil jazyk C#. Jazyk C# je vysokoúrovňový objektově orientovaný programovací jazyk. Tento jazyk vyvinula firma Microsoft a společně se svou platformou .NET. Práce v jazyce C# je jednodušší oproti starším jazykům. Zjednodušení spočívá v automatické správě paměti (garbage collector) – není tak nutné manuálně odstraňovat objekty z paměti – ty se automaticky odstraňují až je to „vhodné“, C# je také typově bezpečnější než například jazyk C++ (nejsou zde automatické konverze například mezi integerem a booleanem). Dalším velkým přínosem je také

zavedení property (vlastností), protože dříve bylo nutné (a například v jazyce Java stále je) napsat velké množství metod Get a Set pro nastavení případně získání hodnoty nějaké vnitřní proměnné. Podstatným důvodem pro výběr jazyka byla dřívější zkušenost autora s programováním v tomto jazyce a celém frameworku .NET.

V implementaci je dodržen správný objektový návrh – třídy jsou důsledně zapouzdřené, ve velkém množství je použito skládání objektů a dědičnost. Programováno je proti rozhraní, nikoli proti implementaci (to znamená, že při změně implementace není potřeba předělávat kód na mnoha místech, ale jen tam, kde se vytváří nová instance). Na několika místech jsou použity návrhové vzory (abstract factory, singleton). Součástí implementace je také možnost konfigurace bez překládání kódu. K této konfiguraci slouží speciální soubor *app.config,* který tuto konfiguraci umožňuje. Asi nejdůležitější možností nastavení je volba režimu přenosu zpráv přes médium – ke změně stačí v nastavení přenosového módu (**TransferMode**) změnit nastavení na **XML1** (přenos v textovém režimu), nebo **BIN1** (přenos v binárním režimu). Ostatní parametry jsou dokumentovány v přiložené dokumentaci ke komunikační knihovně.

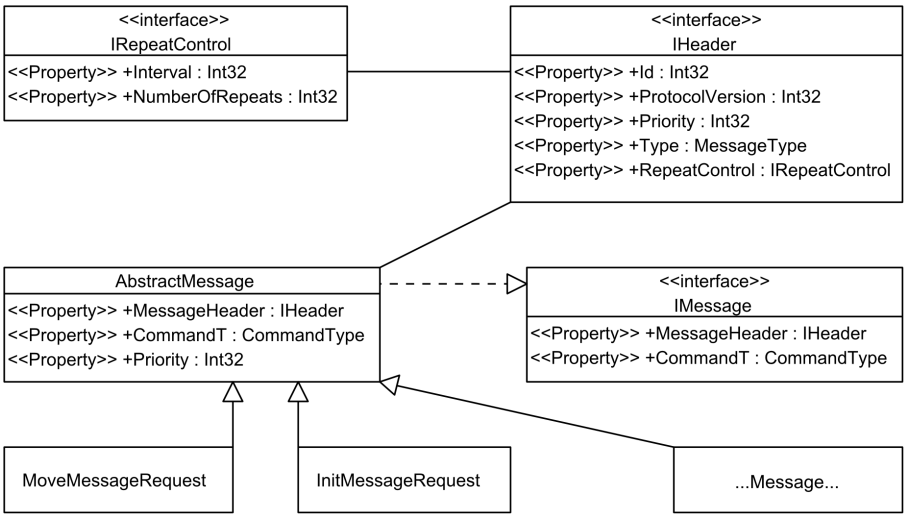
Schéma postupu při přenosu zprávy s využitím navrhovaného komunikačního protokolu je zobrazeno na obrázku 3.3.1. Objekt, který je na obrázku vyznačen jako „serializovaná zpráva“ je zpráva vytvořená v souladu s komunikačním protokolem.



Obrázek 3.3.1: Přenos zprávy komunikačním kanálem

3.3.1 Zpráva jako objekt

Objektově orientovaný jazyk nepracuje s textovými nebo binárními entitami, ale s objekty. Objekt je základní součástí objektově orientovaných jazyků a můžeme si jej představit jako data a operace s těmito daty prostřednictvím metod. Zpráva z tohoto pohledu je objektem, který obsahuje nějaká data a nějaké povolené operace. V systému komunikace s robotem existuje právě 10 různých zpráv, které říkají robotu, co má konat. Objekt zprávy je do jisté míry vytvořen podle kapitoly 3.2, která navrhuje komunikační protokol. Na obrázku 3.3.2 na následující straně je zobrazeno UML schéma zprávy jako objektu a navíc je zachycen vztah mezi abstraktní zprávou a konkrétními objekty zpráv. Na obrázku nejsou zachyceny veškeré poděděné třídy, protože pak by došlo k znepřehlednění schématu (princip přidání libovolné další zprávy je z obrázku patrný). Ke zjednodušení vytváření zpráv je připravena tovární třída. Více v ukázce kódu 3.3.1 na straně 35.



Obrázek 3.3.2: UML diagram tříd zpráv

3.3.2 Serializace a deserializace

Aby bylo možné zprávu přenést skrz komunikační kanál, je třeba ji serializovat. Serializaci si můžeme představit jako proces, při kterém z objektu zprávy uděláme nějakou vhodnou posloupnost bytů, ty následně přeneseme komunikačním kanálem a na druhé straně je opět složíme – deserializujeme. Pokud chceme naprogramovat serializaci a deserializaci můžeme se spolehnout na dostupné možnosti programovacího jazyka (v našem případě schopnosti frameworku .NET) a nechat celý proces na něm (k tomuto účelu slouží třídy jmenného prostoru **System.Runtime.Serialization** a třídy jeho podprostorů), nebo si můžeme naprogramovat serializaci vlastní.

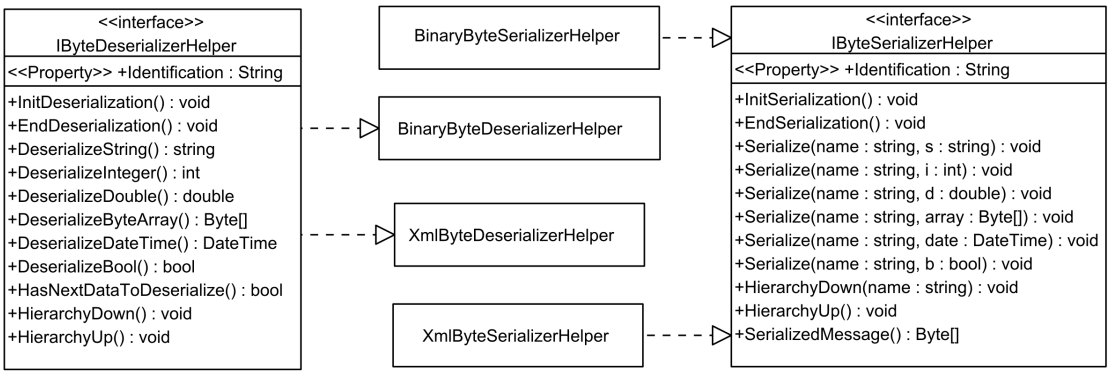
Pokud píšeme vlastní serialiazaci (a deserializaci) existují minimálně dva různé způsoby její implementace :

1. Máme vytvořen serializátor, kterému předáme objekt zprávy a on ji sám převede do požadovaného tvaru. Tento způsob je výhodný v tom, že zpráva nemusí mít implementovánu žádnou serializační (a deserializační) část. Nevýhodou tohoto způsobu je rozsáhlá implementace serializátoru, protože nutně musí mít informace o objektech zpráv. V případě, že potřebujeme několik různě se chovajících serializátorů, může se vyskytnout podobný kód, který je nutné psát znovu.

2. Zpráva bude při serializaci a následné deserializaci poskytovat součinnost. Serializátor poskytuje pouze několik základních metod pro serializaci jednoduchých datových typů (číslo, řetězec, pole bytů,...) a zpráva se sama voláním těchto metod postupně serializuje, případně deserializuje. Tento způsob je výhodný v tom, že sama zpráva zná konkrétního kroky pro svou serializaci a deserializaci. Zodpovědností serializujícího (deserializujícího) objektu je pouze uložit (poskytnout) korektní data do vytvářené (z předané) posloupnosti bytů.

Pro implementaci jsem (po diskuzi s vedoucím práce i kolegy) zvolil druhou možnost, jak zprávy serializovat a deserializovat. Na obrázku 3.3.3 je zobrazen UML diagram tříd pro serializaci a deserializaci. V ukázce kódu 3.3.1 je jednoduchý příklad na vytvoření nové zprávy, její serializaci a deserializaci.

V rámci této práce jsou implementovány dva rozdílné serializátory a deserializátory - textový (XML) a binární.



Obrázek 3.3.3: UML diagram tříd serializátoru a deserializátoru

```
ISerializerDeserializerFactory sdFct =
  SerializerDeserializerAbstractFactory.GetFactory(
  Settings.Default.TransferMode);
IMessageFactory mFct = MessageConfigFactory.GetFactory();
IMessage msg = factory.GetMessageFromParams(MessageType.request,
  CommandType.move, 10);
byte[] data = msg.Serialize(sdFactory.GetSerializer())
  .SerializedMessage(); //serializace
IMessage msg2 = factory.GetMessageFromParams(MessageType.request,
  CommandType.move, 10); //deserializace
msg2.Deserialize(sdFactory.GetDeserializer(data));
```

Ukázka kódu 3.3.1: Vytvoření zprávy, její serializace a deserializace

Textový serializátor

Serializuje zprávy do textové podoby dané komunikačním protokolem. Výstupem je text, který obsahuje validní XML – je možné jej ověřit oproti XML schématu (toto schéma je na přiloženém médiu).

Binární serializátor

Binární serializátor serializuje podobným způsobem jako serializátor textový, ale snaží se minimalizovat nadbytečné informace – nepřenáší názvy tagů a pro naznačení zanoření do struktury (binárně serializovaná zpráva si uchovává stromovou strukturu) používá znak

s ASCII hodnotou – 209 v případě zanoření do struktury, nebo 207 v případě vynořování ze struktury. Primitivní datové typy jsou kódovány pomocí metod třídy **BitConverter**. Všechny vícebytové hodnoty (čísla, řetězce) jsou uloženy jako little-endian (předpokládám, že komunikační protokol implementovaný v .NET bude provozován především na systémech, které jsou little-endian).

Endianita

Endianita způsobuje jednu ze základních nekompatibilit mezi různými systémy a proto důležité při výměně dat zajistit jednotnou endianitu. Endianita určuje způsob uložení vícebytových hodnot. Podle způsobu, jakým jsou tyto hodnoty ukládány, rozlišujeme v dnešní době dva druhy (takzvaný middle-endian se již nepoužívá):

1. **Little-endian** – data jsou ukládána od nejméně významného bytu k nejvýznamnějšímu. Například chceme-li zapsat číslo 16874 (což je 000041EA hexadecimálně), zapíšeme jej pomocí little-endianu jako |0xEA|0x41|0x00|0x00|. Pokud by tuto hodnotu přečetl systém s big-endianem (a chtěli bychom znaménkový typ) dostal by číslo -364838912. Tento systém používají procesory kompatibilní s Intel x86 a x86-64.

2. **Big-endian** – data jsou ukládána od nejvýznamnějšího bytu k nejméně významnému. Například chceme-li zapsat číslo 16874 (což je 000041EA hexadecimálně), zapíšeme jej pomocí big-endianu jako |0x00|0x00|0x41|0xEA|. Pokud by tuto hodnotu přečetl systém s little-endianem (a chtěli bychom znaménkový typ) dostal by číslo -364838912. Tento systém používají procesory ARM, PowerPC, SPARC.

Čerpáno z Wikipedia contributors (2012).

3.3.3 Přenos komunikačním kanálem

Pro přenos serializované zprávy můžeme použít libovolný komunikační kanál (od sdílené paměti, přes sockety až například po poštovní holuby). Pro svou implementaci jsem zvolil použití TCP/IP socketů hned z několika důvodů:

- Přenos přes TCP je spolehlivý – TCP zaručuje kompletní doručení paketů ve správném pořadí, na rozdíl od UDP, které tuto spolehlivost nezaručuje. Při využití TCP není potřeba další kontrolní mechanismus implementovaný na úrovni serveru či klienta.

- Není rozdíl mezi komunikací (až na rychlost) na jednom počítači a na síti (ovládací software může fyzicky běžet na jiném počítači, než zprostředkovatelský program).

- Daň za tyto výborné vlastnosti je poměrně malá - na začátku spojení musí dojít k takzvanému handshaku (třífázové ustavení spojení). Více o ustavování spojení například v článku, který napsal Odvárka(2000).

Přenos každé serializované zprávy začíná odesláním její velikosti (4B integer), nyní ovšem v takzvaném síťovém pořadí bytů (Network byte order) nebo-li big-endianu. Za touto velikostí je pak odeslána vlastní serializovaná zpráva. Příjemce naopak přečte velikost přijímané zprávy a dále zachytí správný počet bytů, které odpovídají přijaté velikosti této zprávy, kterou poté může zpracovat.

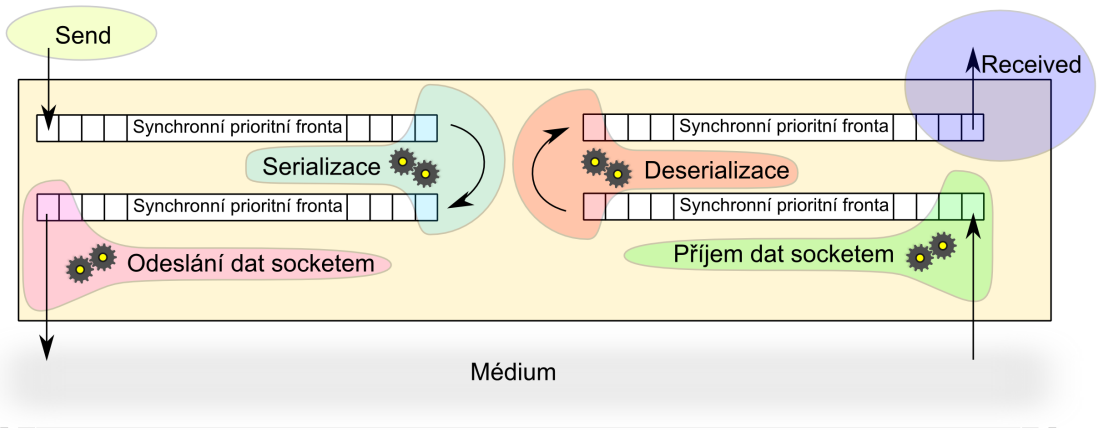
3.3.4 Přehled tříd implementace komunikačního protokolu

Všechny třídy použité pro implementaci protokolu je možné si prostudovat z dodaných zdrojových kódů (projekt **ComLib**), případně si prohlédnout vygenerovanou dokumentaci (oboje je dostupné na přiloženém médiu). K provedení celého řetězce naznačeného na obrázku 3.3.1 na straně 33 jsou pro jednoduchost vlastního použití napsány dvě třídy:

1. **TcpIPCommunicationClient** – třída, která zajišťuje funkcionalitu klienta (ovládací program). Objekty z této třídy se umí připojit na server, serializovat a deserializovat zprávy a předávat přijaté zprávy k dalšímu zpracování.

2. **TcpIPCommunicationServer** – třída, která zajišťuje funkcionalitu serveru (zprostředkovatelský program). Objekty z této třídy poskytují bod pro připojení, umí serializovat a deserializovat zprávy a přijaté zprávy předávat k dalšímu zpracování.

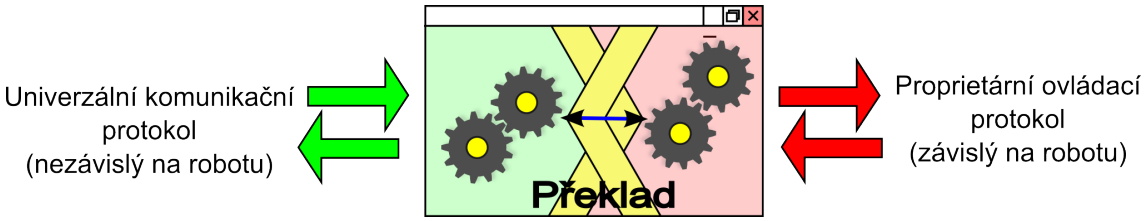
Obě třídy jsou ve své podstatě kompozity složené z již představených částí. Každá větší součást těchto tříd běží ve svém vlastním vlákně a je tak co nejméně závislá na ostatních součástech. Výměna dat mezi jednotlivými vlákny probíhá na základě synchronní prioritní fronty. Na obrázku 3.3.4 je vyznačena struktura těchto zjednodušujících tříd, vlastní třída je orámována čtvercem, součásti běžící v jednom vlákně jsou vyznačeny a odlišeny barvou.



Obrázek 3.3.4: TcpIPCommunicationClient – schéma činnosti

3.4 Zprostředkující aplikace

Zprostředkující aplikace spojuje aplikaci pro řízení robota a ovládaného robota. Na jedné straně je ovládána obecným komunikačním protokolem, na straně druhé pak protokolem konkrétního robota. Uvnitř aplikace dochází k překladu z jednoho protokolu do druhého (překladu zpráv na jednotlivá volání, konverze naměřených dat ze senzorů, ...). Schéma činnosti zprostředkující aplikace je na obrázku 3.4.1.



Obrázek 3.4.1: Schéma činnosti zprostředkující aplikace

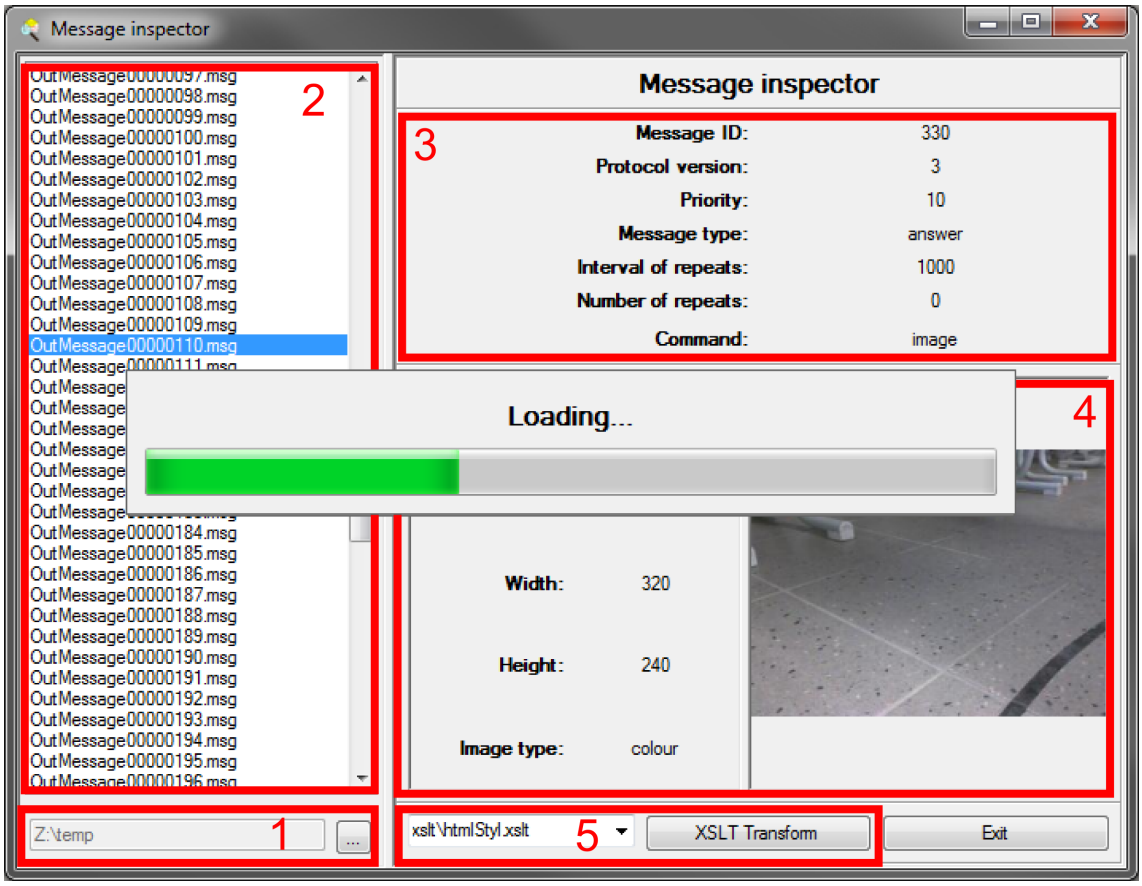
Zprostředkující aplikace nemusí být fyzicky aplikací (tak jak si aplikaci představujeme, to jest samostatně běžící aplikace), ale může být součástí robota a zdánlivě se tak zdá, že ovládací program ovládá přímo samotného robota. Implementace přímo na robotu není součástí této práce, ale tato implementace může být zajímavým rozšířením této práce do budoucna. Součástí této práce je zprostředkovatelská aplikace pro robota Robotino firmy Festo.

3.5 Ladění komunikačního protokolu

Pokud všechna komunikace probíhá korektně a robot se přesně podle zamýšlených pokynů chová, není důvod pro ladění komunikačního protokolu. Pokud však reaguje robot neočekávaně, případně na zprávy nereaguje vůbec, je zapotřebí program odladit. Pro účely ladění zpráv komunikačního protokolu – ověření zda dané zprávy obsahují správné příkazy a parametry – je v prvé řadě možnost zaznamenávání veškerých zpráv, které klient, či server odeslal nebo přijal, případně zapisování do log souboru. K logování událostí je použita knihovna **log4net**, která je klonem pokročilého logovacího systému pro programovací jazyk Java **log4j**. Více informací k tomuto logovacímu systému je k nalezení na oficiálních stránkách Apache Software Foundation (2011).

K různým nastavení logování a ladění slouží soubor s konfigurací *app.config*. Pokud chcete přenastavit logování, pak stojí za pozornost dokumentace, která je k nalezení na oficiálních stránkách **log4net** (Apache Software Foundation, 2011). K nastavení ukládání zpráv (v textovém nebo binárním režimu) slouží v nastavení dva přepínače **StoreIncomingMessage** (pokud je nastaven na **true,** ukládá všechny příchozí zprávy do adresáře *INCOMING_MESSAGES*) a **StoreOutgoingMessage** (pokud je nastaven na **true,** ukládá všechny odchozí zprávy do adresáře *OUTCOMING_MESSAGES*).

Pokud máme zprávy uloženy, můžeme se na ně podívat prostřednictvím programu **Message inspector** (program a zdrojový kód přiložen na médiu). Program umí přehledně zobrazit všechny podporované zprávy (hlavičku i parametry) a jeho rozhraní je snadno pochopitelné a ovladatelné (ukázka hlavní obrazovky programu s popisem ovládacích prvků je na obrázku 3.5.1). Program umožňuje exportovat danou zprávu v libovolném formátu, který je určen vytvořeným XSLT stylem (v této práci se XSLT styly a XSL transformací nezabývám – pro bližší seznámení s tvorbou vlastních šablon doporučuji prakticky zaměřené webové stránky inženýra Koska (2011), nebo W3Schools). Pro ukázku je přiložen styl, který zprávu převede na HTML stránku.



Obrázek 3.5.1: Popis uživatelského prostředí programu Message inspector

1. Cesta k načtenému adresáři. Tlačítkem s označením „...“ se vyvolá okno pro výběr adresáře.
2. Jména jednotlivých souborů v adresáři.
3. Základní informace z hlavičky zprávy.
4. Parametry a další údaje (například obrázek) zprávy.
5. Výběr XSLT šablony a export za pomoci této šablony.

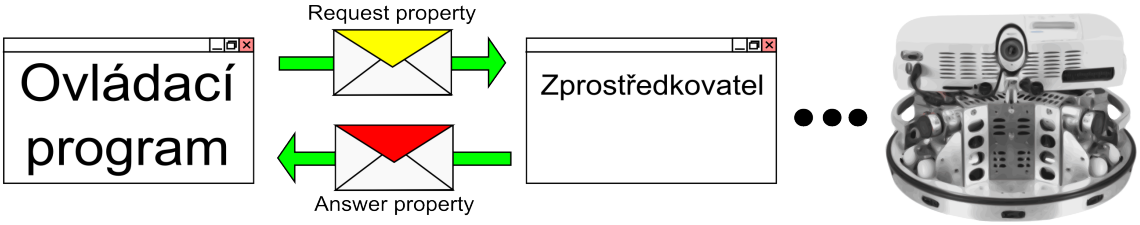
Kapitola 4

Aplikace pro řízení robota

Komunikační protokol je navržený a implementovaný, to však pro řízení robota nestačí. Je zapotřebí navrhnout a implementovat aplikaci, která bude využívat navržený komunikační protokol a bude umožňovat ovládat robota (číst vstupní data ze senzorů, pohybovat robotem, ...). Podíváme-li se na návrh protokolu, je snadné si představit budoucí strukturu aplikace, která se bude skládat z obrazu kamery, dat vstupních senzorů, vlastního ovládání pohybu robota (protože to bývá obvykle nejdůležitější) a ovládání výstupních prvků – digitálních výstupů, nebo volání vnitřních funkcí.

Aplikace je nezávislá na konkrétním použitém robotu – tato skutečnost vychází z představené ideologie návrhu (ideologie návrhu je popsána v kapitole 3.1 na straně 23) a může tak ovládat různé roboty, kteří využívají jednotný komunikační protokol, případně ty, pro které je napsán zprostředkující program pro převod jejich nativního komunikačního protokolu do protokolu navrhovaného.

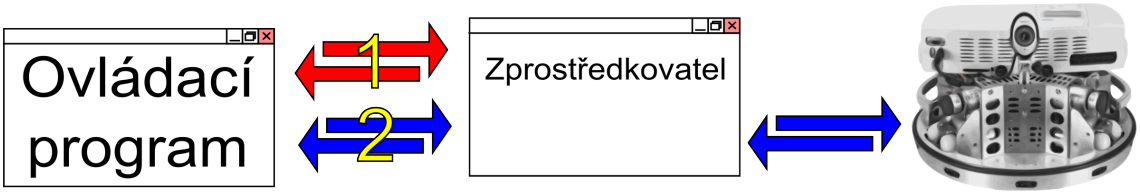
Ačkoli je aplikace nezávislá na použitém robotu, umožňuje dynamicky měnit své prostředí tak, že se vždy přizpůsobí konkrétnímu robotu. Pokud například robot nemá dálkoměry, je zbytečné zobrazovat panel s přehledem naměřených hodnot, naopak pokud má robot například 4 kamery, zobrazí se panel s výběrem aktuálního zdroje dat pro zobrazení v aplikaci. Pro zjištění počtu jednotlivých senzorů program využívá zprávu vlastností (více o zprávě vlastností v části 3.2.4 na straně 32) – která obsahuje všechny tyto nezbytné informace. Schéma zjištění počtu jednotlivých typů senzorů je zobrazeno na diagramu 4.0.1.



Obrázek 4.0.1: Diagram zjištění vlastnosti robota

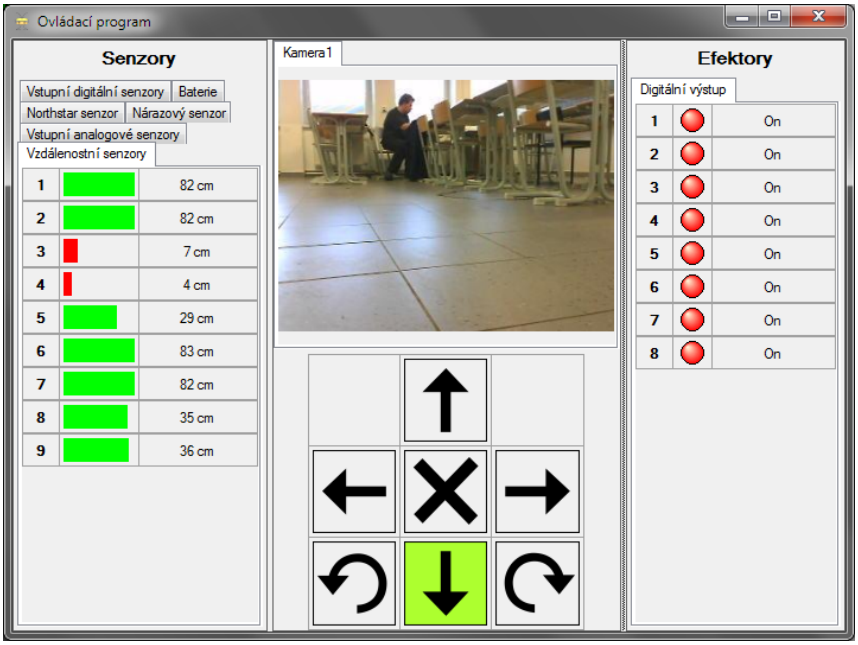
Prostředí aplikace je – alespoň v to autor doufá – uživatelsky příjemné a přehledné. Aplikace není vytvořena pro odborníky a neobsahuje mnoho nastavení, ale běžným uživatelům bude vyhovovat jednoduchost použití. Aplikace je lokalizována do českého a anglického jazyka, k přepnutí jazyka dochází automaticky v závislosti na nastavení operačního systému (výchozím jazykem – v případě nepřítomnosti lokalizačního balíčku – je angličtina tak aby i nečesky mluvící uživatel mohl tuto aplikaci používat).

Po spuštění aplikace je nutné se nejprve připojit ke zprostředkovateli (a od zprostředkovatele obdržet informaci o robotu) a poté k vlastnímu robotu. Připojování tak probíhá dvoufázově a je přehledně znázorněno na diagramu 4.0.2.

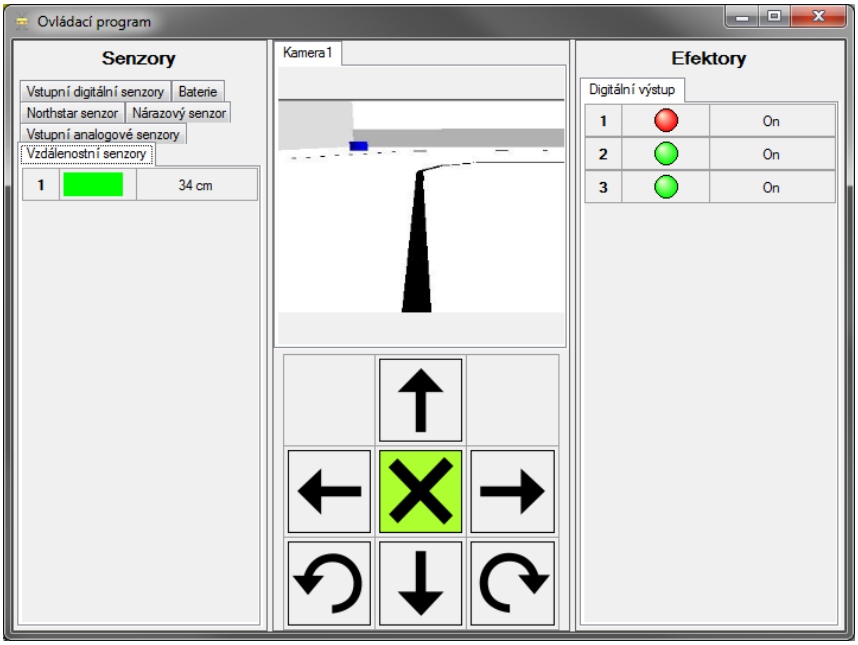


Obrázek 4.0.2: Diagram dvoufázového připojení

Na obrázku 4.0.3 je zobrazeno hlavní okno ovládací aplikace po připojení k robotu Robotino Festo, na obrázku 4.0.4 na následující straně je zobrazeno to samé hlavní okno ovládací aplikace po připojení k robotu iRobot Create. Je možné si povšimnout různého počtu dálkoměrných senzorů a digitálních výstupů.



Obrázek 4.0.3: Hlavní okno aplikace pro řízení robota po připojení k Festo Robotino



Obrázek 4.0.4: Hlavní okno aplikace pro řízení robota po připojení k iRobot Create

Kapitola 5

Prezentační úlohy s využitím počítačového vidění

Navrhnout a implementovat komunikační protokol je z hlediska tohoto návrhu a implementace zajímavé, ale pro nepoučeného pozorovatele málo efektní. Ukázat ovládací program a předvést, jak robot poslouchá, je z diváckého pohledu nepříliš zajímavé, protože divák ví, že je robot ovládán člověkem. Mnohem zajímavější je, pokud se robot pohybuje a něco samostatně vykonává (nesmíme zapomenout, že inteligence robota vzniká až v očích pozorovatele – jak říká doktorka Volná (2005) ve své práci). Na dnech otevřených dveří (a nejen na nich) je vhodné prezentovat výsledky práce a předvést, jak roboti sami pracují, nejlépe na úlohách, které na první pohled zaujmou.

V rámci této práce jsou vytvořeny dvě prezentační úlohy, které mají společný základ ve využití algoritmů pro počítačové vidění. Než přistoupím k zadání úloh a způsobu jejich řešení (pomocí algoritmů počítačového vidění), je nutné se podívat na některé základní pojmy a operace využívané pro zpracování obrazu.

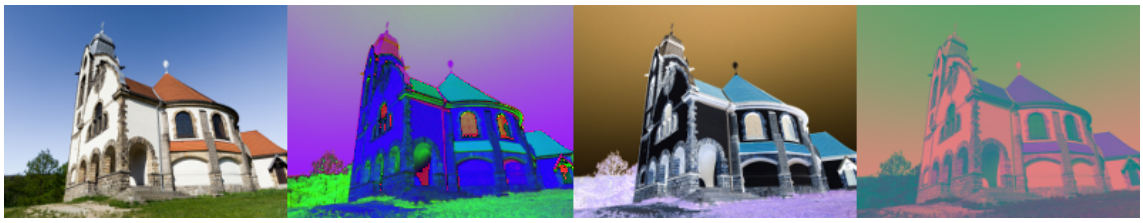
5.1 Zpracování obrazu

Podrobný výklad všech aspektů získání (různé objektivy, kamerové čipy, případná digitalizace, ...) a zpracování obrazu je mimo rozsah a způsob zaměření této práce, proto je zde podán jen velmi stručný úvod do některých vybraných (později použitých) pojmů a algoritmů. Zájemce o hlubší studium počítačového vidění odkazuji na podle mého názoru výbornou knihu od autora Richarda Szeliského (Szeliski, 2010) věnující se této tématice.

5.1.1 Barevné prostory

Obrázek je posloupnost bytů určitého významu. Pokud chceme zobrazit barevný obrázek, opakuje se vždy následující situace – z posloupnosti bytů získám n souřadnic, které odkazují na místo n-rozměrného prostoru a teprve volba barevného prostoru určuje, jaká konkrétní

barva odpovídá těmto souřadnicím. Dalo by se říci, že barevný prostor je zobrazením mezi souřadnicemi (většinou třemi, jde tedy o zobrazení třírozměrného prostoru) a barvou. Při nevhodně zvoleném barevném prostoru mohou být barvy nesprávně interpretovány (a obrázek nevypadá tak, jak autor původně zamýšlel). Příklad, jak může vypadat tentýž obrázek, pokud je použit nesprávný barevný prostor, naleznete na obrázku 5.1.1. Některé barevné prostory jsou vhodné pro pořizování a zobrazování fotografií, jiné pro jejich úpravu a další pro tisk. Následující popis není úplným přehledem všech možných barevných prostorů. Zpracováno podle materiálů Radky Tezaur(2003).



Obrázek 5.1.1: Nesprávně interpretovaný barevný prostor – zleva doprava správný obrázek v RGB, interpretace v HSV, interpretace v CMY, interpretace v YCbCr

Barevný prostor RGB

Barevný prostor RGB využívá aditivního míchání barev, kdy se jednotlivé barevné složky sčítají – vyzařují intenzivněji. Většina dnešních zobrazovacích i snímacích zařízení používá barevný prostor RGB (RGB je zkratkou pro Red – červená, Green – zelená a Blue – modrá). Používání RGB je dáno výrobní strukturou, kdy LCD displeje, stejně jako čipy pro snímání barevného obrazu mají právě 3 barevné filtry odpovídající tomuto barevnému prostoru – pominu-li takové případy jako je technologie Quattron firmy Sharp přidávající žlutou barvu do televizních obrazovek (podle článku pana Říčného (2010) však tato technologie není nijak revoluční). Různých barevných prostorů RGB, které se od sebe liší jen nepatrně, existuje více druhů – Adobe RGB, Apple RGB, CIE RGB, sRGB, ... [Tezaur] (2003).

Barevný prostor CMY(K)

Tento barevný prostor používají tiskárny pro barevný výstup. Barva na papíře se totiž vytváří jiným způsobem než při tvorbě pomocí světla. Tento barevný prostor využívá subtraktivního míchání barev, kdy je světlo jednotlivými barvami pohlcováno a výsledná barva je tak rovna pouze odražené (nepohlcené) části spektra. Tento barevný prostor je tvořen barvami C – Cyan – Azurová, M – Magenta – purpurová a Y – Yellow – žlutá. Z ekonomických (a vzhledových) důvodů se velmi často přidává černá barva – blacK, která však nemá žádné jiné opodstatnění.

Barevný prostor YUV, YCbCr

Tyto barevné prostory pracují velice podobně. YUV se používá především v analogové technologii a YCbCr se používá v technologii digitální. V obou barevných prostorech značí Y jasovou složku (člověk je nejcitlivější na jasový kanál) a složky U,V (respektive Cb, Cr) zde značí složky barvové. Pro přepočet se dá použít následující maticový vzorec (vzorec pro převod do YCbCr je podobný a snadno dohledatelný). Vzorec dle autora Feipeng (2011).

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} 0,299 & 0,587 & 0,114 \\ -0,147 & -0,289 & 0,436 \\ 0,615 & -0,515 & -0,100 \end{pmatrix} * \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Barevný prostor HSV, HSL

Barevné prostory HSV a HSL jsou si velmi podobné a proto budu popisovat jen barevný prostor HSL, který je nejzajímavějším (protože převodem do tohoto barevného prostoru je výrazně ulehčena druhá prezentační úloha) barevným prostorem co se týká této práce. Prostor HSL má tři složky: H – Hue – odstín, S – Saturation – saturace, nasycení a L – Lightness – světlost. Tento barevný prostor nejvíce odpovídá lidskému vnímání barev. Barevný prostor HSL je také velmi vhodný pro úlohy počítačového vidění, protože různě saturované a světlé odstíny jedné barvy jsou si rovny ve složce H, jak tvrdí Cheng et al. (2001).

Na obrázku 5.1.2 je zobrazeno několik různých grafických objektů s různými stupni zelené barvy (na první pohled snadno odlišitelnými). Vedle je vyobrazena složka H tohoto obrázku a z ní je vidět, že vše je jen jeden zelený odstín s různým nastavením saturace a světlosti.

Obrázek 5.1.2: Barevný prostor HSL

5.1.2 Segmentace obrazu

Segmentace obrazu je odborný pojem z oblasti počítačového vidění, kterým se vyjadřuje proces, jakým rozdělíme obraz na několik různých skupin (segmentů) podle nějaké společné vlastnosti. Touto vlastností může být například jas, odstín, hodnoty okolních bodů,... Po segmentaci tak máme několik různých segmentů (a každý segment by měl vyjadřovat jeden zájmový prvek obrazu), se kterými můžeme provádět další operace. Nejčastěji se používá segmentace prahováním.

Prahování můžeme rozdělit do dvou skupin:

- Prahování s pevným prahem – práh je pevně dán a je pro daný obraz neměnný
- Adaptivní prahování – práh není pevně dán, prahování probíhá podle situace v daném konkrétním obrazu

Prahování s pevným prahem

Představme si obraz jako matici bodů, jejichž hodnota vyjadřuje míru vlastnosti, podle které budeme prahovat. Pak výsledný obraz dostaneme podle následujícího předpisu,

$$g(i,j) = \begin{cases} 1 & f(i,j) \geq T \\ 0 & f(i,j) < T \end{cases}$$

kde f je matice bodů vstupního obrazu, g je výsledný prahovaný obraz a T je hodnota prahu.

Drobnou úpravou předchozího algoritmu můžeme získat prahování v intervalu hodnot vlastnosti, případně prahování s více prahy. Rozhodovací funkce pro prahování v intervalu vlastnosti pak vypadá,

$$g(i,j) = \begin{cases} 1 & f(i,j) \in I \\ 0 & f(i,j) \notin I \end{cases}$$

kde f je matice bodů vstupního obrazu, g je výsledný prahovaný obraz a I je interval míry vlastnosti. Zpracováno podle přednášek docenta Chaloupky (2010).

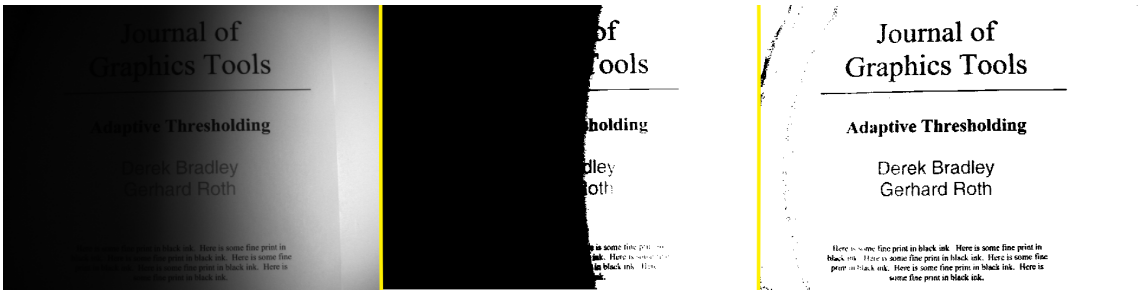
Adaptivní prahování

Výhodou adaptivního prahování je skutečnost, že se dokáže vypořádat s proměnnou hodnotou vlastnosti (například jasu) v rámci jednoho obrázku stejně dobře jako s proměnou hodnotou vlastnosti v průběhu času (například proud obrazových dat) a přitom dokáže zachytit významné změny ve vlastnosti.

Jedním z algoritmů pro adaptivní prahování je Bradleyho algoritmus nazvaný na počest svého tvůrce. Tento algoritmus je založen na myšlence, že pozadí se v malém okolí (nazvaném okno) příliš nemění. Celý algoritmus je dvoufázový, jeho první částí je výpočet takzvaného integrálního obrazu (tento výpočet výrazně urychluje celý algoritmus) – což není nic jiného než, že ke každému pixelu $[x, y]$ je přiřazeno číslo, které odpovídá součtu všech hodnot vlastnosti v obdélníku s levým horním rohem na souřadnici $[0, 0]$ a pravým dolním rohem na souřadnici $[x, y]$. Ve druhé fázi je každý pixel porovnán s průměrnou hodnotou okna (díky integrálnímu obrazu je výpočet průměrné hodnoty proveden v konstantním čase) dané velikosti a pokud je rozdíl pixelu a průměrné hodnoty okna malý (je možné nastavit parametrem), přiřadíme barvu pozadí. Pokud je rozdíl pixelu od průměrné

hodnoty okna dostatečně velký (je možné nastavit parametrem), pak jde pravděpodobně o nějaký významný bod a tomuto pixelu přiřadíme barvu popředí. Zpracováno podle příspěvku v časopise od Bradleyho a Rotha (2007), kde v případě zájmu naleznete přesný popis a další informace včetně implementace v pseudokódu.

Na obrázku 5.1.3 naleznete ukázku prahování s pevným prahem v porovnání s adaptivním prahováním Bradleyho algoritmem. Segmentace prahováním není jedinou možností a existují i další metody segmentace obrazu (segmentace založená na detekci hran, algoritmus Watershed,. . .), ale pro tuto práci je to metoda (společně s binární morfologií) dostačující. Ostatní metody jsou podrobně vysvětleny v přednáškách docenta Chaloupky (2010).



Obrázek 5.1.3: Prahování s pevným prahem a adaptivní prahování. Na prvním obrázku zleva je původní obrázek, na druhém je tento obrázek segmentován prahováním s pevným prahem a poslední obrázek ilustruje použití adaptivního prahování – zde zobrazen výsledek po použití Bradleyho algoritmu. Obrázky převzaty z příspěvku autorské dvojice Bradley – Roth (2007).

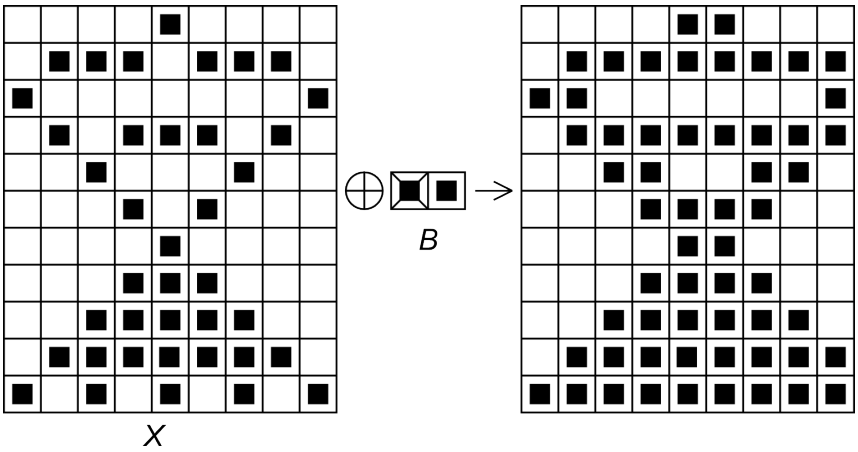
5.1.3 Matematická a binární morfologie

V digitálním zpracování obrazu jde o matematický nástroj pro předzpracování i segmentaci obrazu. Obrázky lze modelovat pomocí bodových množin libovolné dimenze – binární morfologie používá množinu dvojic celých čísel (jde tak o prostor $\mathbb{Z}^2$). Morfologická transformace Ψ je dána relací mezi bodovou množinou X a typicky menší bodovou množinou B, která se obvykle nazývá strukturní element. Strukturní element má „lokální“ počátek – pokud je strukturní element zadán graficky, bývá tento počátek znázorněn křížkem. Binární matematická morfologie definuje dvě základní operace a dvě operace složené. Tato sekce a podsekce je zpracována dle přednášek docenta Chaloupky (2010).

Binární dilatace

Binární dilatace se používá pro zaplnění malých děr a zálivů v binárním obraze. Maximální velikost a strukturu zaplňovaných děr určuje tvar a velikost strukturního elementu. Binární dilatace sčítá dvě bodové množiny a je dána následujícím vztahem. Pro zlepšení představy je binární dilatace zobrazena na obrázku 5.1.4 na následující straně.

$$X \oplus B = \left\{ p \in \mathbb{E}^2 : p = x + b,\, x \in X \wedge b \in B \right\}$$

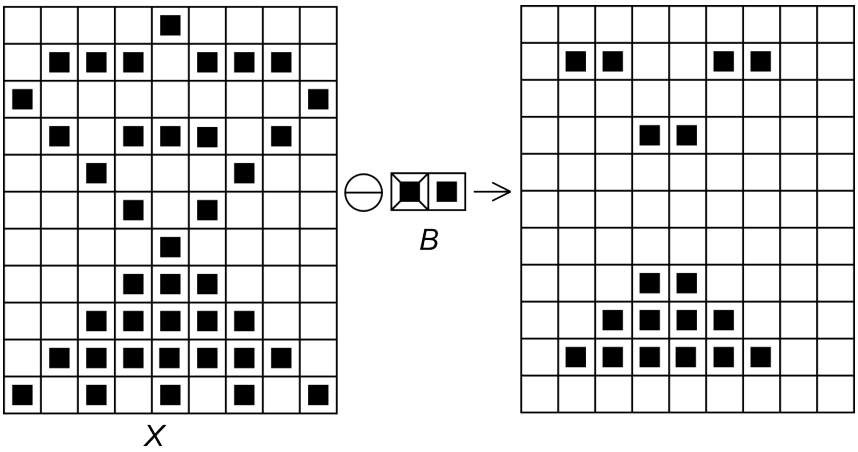


Obrázek 5.1.4: Binární dilatace

Binární eroze

Binární eroze se používá pro odstranění malých objektů v obraze. Maximální velikost a strukturu odstraňovaných objektů určuje tvar a velikost strukturního elementu. Binární eroze je dána následujícím vztahem. Pro zlepšení představy je binární eroze zobrazena na obrázku 5.1.5.

$$X \ominus B = \left\{ p \in \mathbb{E}^2 : p = x + b \in X \,\text{pro každé}\, b \in B \right\}$$

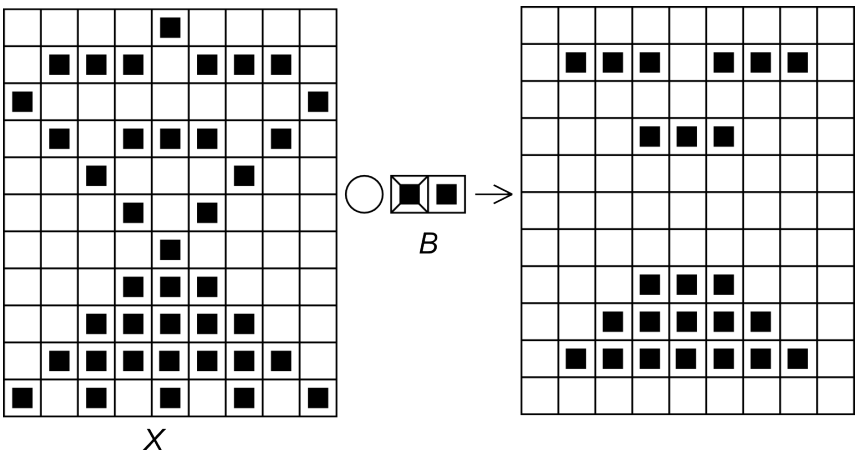


Obrázek 5.1.5: Binární eroze

Binární otevření

Binární otevření se používá pro rozdělení objektů, které si jsou blízko – jsou spojeny pouze úzkou spojnicí. Binární otevření není základní operací, ale jedná se o binární erozi následovanou binární dilatací. Binární otevření je dáno následujícím vztahem. Pro zlepšení představy je binární otevření zobrazeno na obrázku 5.1.6.

$$X \circ B = (X \ominus B) \oplus B$$



Obrázek 5.1.6: Binární otevření

Binární uzavření

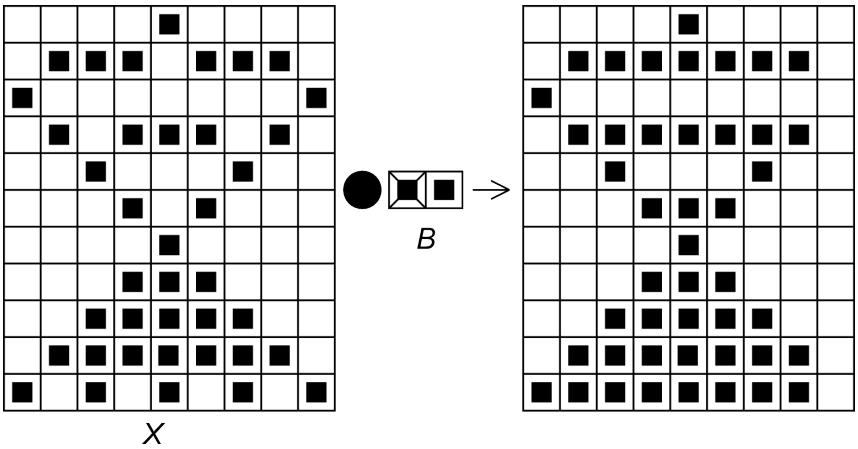
Binární uzavření se používá pro spojení objektů, které jsou si blízko – jsou odděleny pouze úzkým prostorem. Binární uzavření není základní operací, ale jedná se o binární dilataci následovanou binární erozí. Binární uzavření je dáno následujícím vztahem. Pro zlepšení představy je binární uzavření zobrazeno na obrázku 5.1.7 na následující straně.

$$X \bullet B = (X \oplus B) \ominus B$$

5.1.4 Nalezení spojité oblasti a výpočet těžiště oblasti

Abychom mohli spočítat plochu nebo například těžiště nějaké oblasti, je zapotřebí nejprve najít spojité oblasti obrazu. Jedním z algoritmů, který se k tomuto účelu používá, je algoritmus barvení oblastí. Algoritmus barvení oblastí je dvoufázový a předpokládá jako vstup binární obraz. Popis algoritmu a teorie pro výpočet těžiště spojité oblasti pochází ze skript autorského týmu doktora Horáka (Horák et al., 2008).

1. Algoritmus postupně prochází obraz zleva doprava a shora dolů a každému jedničkovému (nenulovému) obrazovému elementu přiřadí hodnotu podle hodnoty všech jeho



Obrázek 5.1.7: Binární uzavření

již obarvených sousedů – již obarvení sousedi se nachází v řadě nad ním (tři) a jeden vedle něj. Při přiřazování hodnoty postupuje podle následujících pravidel:

- jsou-li všichni sousedi nuloví, pak přiřadíme bodu dosud nepřidělenou barvu
- pokud je jeden nenulový nebo je více nenulových, ale se stejnou barvou, přiřadíme bodu tuto barvu
- pokud je více nenulových sousedů, ale s různou barvou dochází ke kolizi barev – bodu přiřadíme jednu z těchto hodnot a do tabulky ekvivalencí si algoritmus poznamená, že tyto různé barvy jsou ekvivalentní

2. Algoritmus projde znovu celý obraz a podle vytvořené tabulky ekvivalencí přebarví všechny ekvivalentní oblasti na jednu barvu. Výsledkem je nalezení (a očíslování) všech souvislých oblastí v obraze.

Geometrické momenty jsou fotometrické příznaky založené na regionech (oblastech) a jejich výpočet tedy vychází z tvaru objektu a jasových hodnot jeho pixelů (v případě binárního obrazu jsou hodnoty 0 a 1). Pro každou oblast můžeme sestavit nekonečně mnoho různých geometrických momentů. Základní geometrický moment řádu $p + q$ se stanoví podle následujícího vzorce (kde $s(x, y)$ vyjadřuje hodnotu pixelu v daném bodě o souřadnici x a y):

$$m_{pq} = \sum_{Y} \sum_{X} x^p y^q s(x, y)$$

Těžiště objektu lze přímo vypočítat z obrazových dat za použití základního geometrického momentu nultého a prvního řádu. Pro souřadnici x_t platí následující vztah:

$$x_t = \frac{\sum_Y \sum_X x \cdot s(x, y)}{\sum_Y \sum_X s(x, y)} = \frac{m_{10}}{m_{00}}$$

Pro souřadnici y_t platí analogický vztah:

$$y_t = \frac{\sum_Y \sum_X y \cdot s(x,y)}{\sum_Y \sum_X s(x,y)} = \frac{m_{01}}{m_{00}}$$

5.2 Knihovny pro počítačové vidění

Veškeré algoritmy počítačového vidění (v anglické terminologii označované jako algoritmy CV – Computer Vision) je možné naprogramovat ve vlastní režii, ovšem toto programování v sobě nese riziko nízké optimalizace daných algoritmů způsobené nedostatkem času, který by bylo nutné této optimalizaci věnovat. Na trhu existují komerční knihovny které implementují tyto algoritmy – například Imaging Library od firmy Matrox (http://-www.matrox.com). Kromě těchto placených knihoven je možné nalézt i mnoho knihoven se svobodnou licencí.

5.2.1 OpenCV

OpenCV je pravděpodobně nejznámější a nejrozšířenější knihovna pro počítačové vidění. Tato knihovna je šířena pod licencí BSD a je zdarma dostupná pro osobní potřebu, výuku, ale i komerční nasazení. Knihovna obsahuje více než 2500 optimalizovaných algoritmů a ke stažení je na oficiálních stránkách tohoto projektu. V knihovně nalezneme funkce pro zpracování obrazu, videa, detekci pohybu, rekonstrukci 3D scény, funkce pro strojové učení (neuronové sítě a podobně) i algoritmy akcelerované grafickou kartou (zpracováno podle webových stránek projektu OpenCV, 2012). Tuto knihovnu ve své práci nepoužívám, protože je psaná v jazyce C++ a neexistuje žádný svobodný port této knihovny pro jazyky platformy .NET – jistou možností je použití Emgu CV, které vyžaduje při nesvobodném nasazení (bez zdrojových kódů) aplikace zaplatit licenci, jak uvádí stránky projektu Emgu CV (2012).

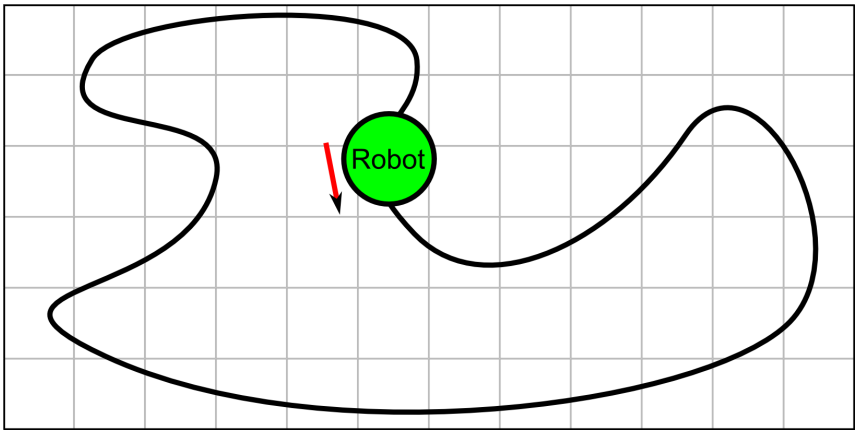
5.2.2 AForge.NET

Je framework pro počítačové vidění šířený pod otevřenou licencí vhodný pro vývojáře na platformě .NET (framework sám je napsán v jazyce C#). AForge.NET obsahuje algoritmy pro zpracování obrazu, strojové učení, neuronové sítě, práci s videem, třídy pro práci s genetickými algoritmy i ovládací knihovny pro některé roboty (například Lego Minstorm RCX). Rozsahem se nevyrovná knihovně OpenCV, ale obsahuje všechnu potřebnou funkčnost, kterou potřebuji (zpracováno podle webových stránek projektu AForge.NET, 2012). Práce s touto knihovnou se mi zdá jednodušší a přímočařejší než v případě OpenCV, navíc je tato knihovna primárně určena pro platformu .NET, kterou ve své práci využívám a proto tuto knihovnu preferuji.

5.3 Jízda po čáře

Zadání úlohy

Na ploše je vyznačena černá čára o minimální šířce 1 cm. Robot jede po této čáře a tuto vyznačenou dráhu neopouští. Na konci čáry robot zastaví (čára samozřejmě může být nekonečná, pak robot pokračuje stále v pohybu). Na obrázku 5.3.1 je zobrazeno schéma (pohled shora) této úlohy.

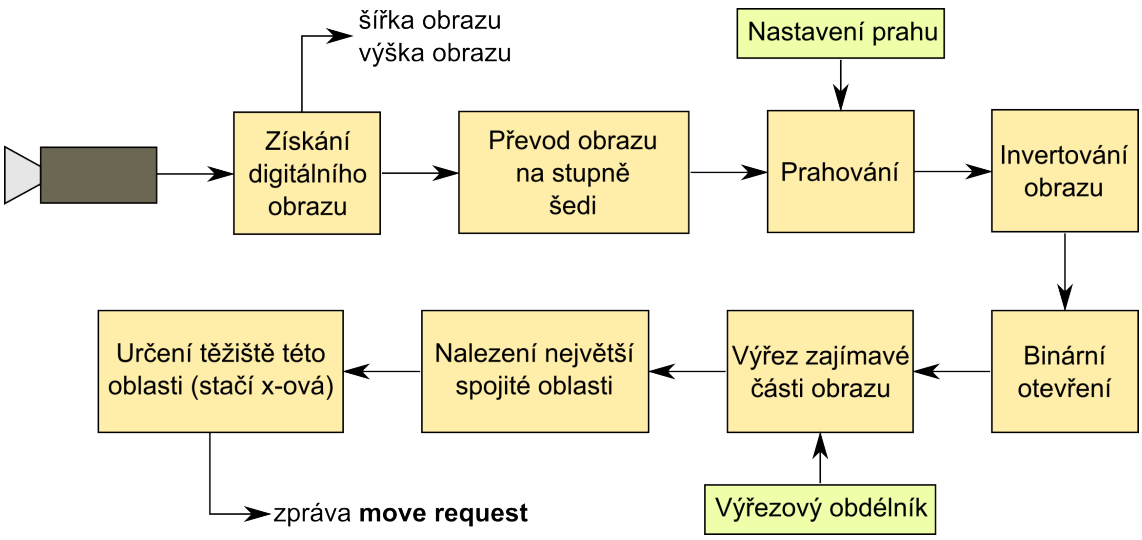


Obrázek 5.3.1: Schéma prezentační úlohy – „Jízda po čáře“

Řešení

Pro řešení této úlohy se dají využít různé senzory robota. Je možné využít difúzní senzory, které umí rozpoznat odrazivost materiálu a podle toho řídit směr pohybu nebo se dá použít optická kamera. Osobně jsem si zvolil optickou kameru v kombinaci s počítačovým viděním, protože kombinace jejich možností jsou lepší než možnosti difúzního senzoru – plocha s čarou není jednobarevná, ale dlaždice obsahují příměsi dalších barev (černé, bílé a dalších odstínů šedi), kde by difúzní senzor mohl mít problém. Algoritmus řešení je zobrazen na obrázkovém diagramu 5.3.2 na následující straně. V postupu je použito prahování s pevným prahem, může však být nahrazeno adaptivním prahováním (záleží na konkrétní situaci). Na obrázku 5.3.3 na následující straně je zobrazeno postupné zpracování obrazu podle diagramu 5.3.2 na následující straně.

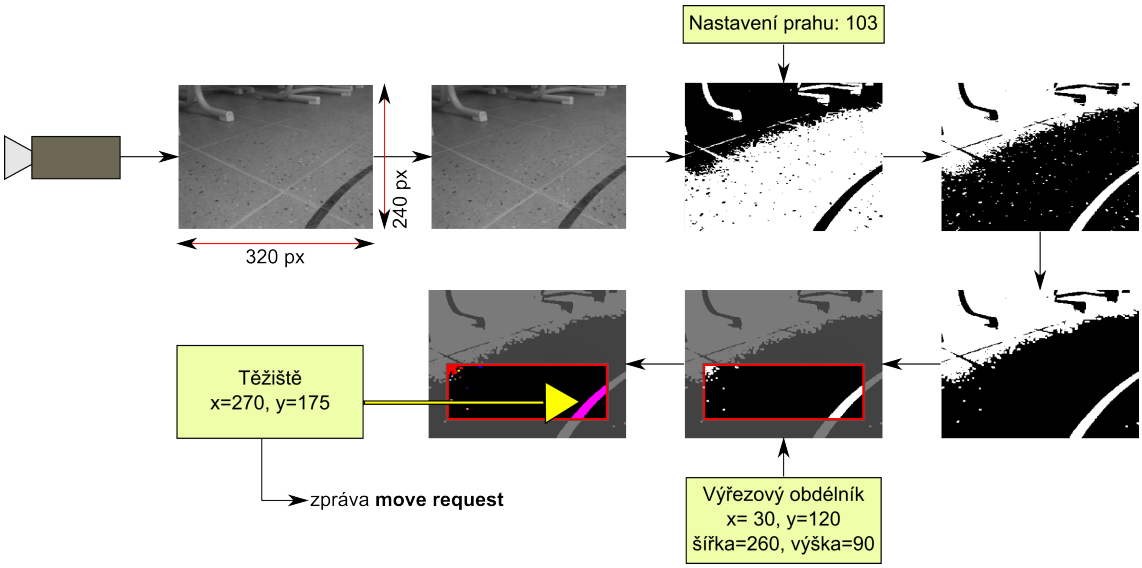
Parametry výsledné zprávy move request (podrobný popis naleznete v sekci 3.2.3 na straně 28) jsou uvedeny v tabulce 5.3.1 na následující straně.



Obrázek 5.3.2: Schéma algoritmu úlohy „Jízda po čáře“

Tabulka 5.3.1: Parametry zprávy v úloze „Jízda po čáře“

Název identifikátoru	Hodnota
direction	0
speed	podle nastavení uživatele
angularSpeed	$\left(\frac{\text{šířka obrázku}}{2}\right) - \text{x souřadnice těžiště}$



Obrázek 5.3.3: Zpracování obrazu – algoritmus úlohy „Jízda po čáře“

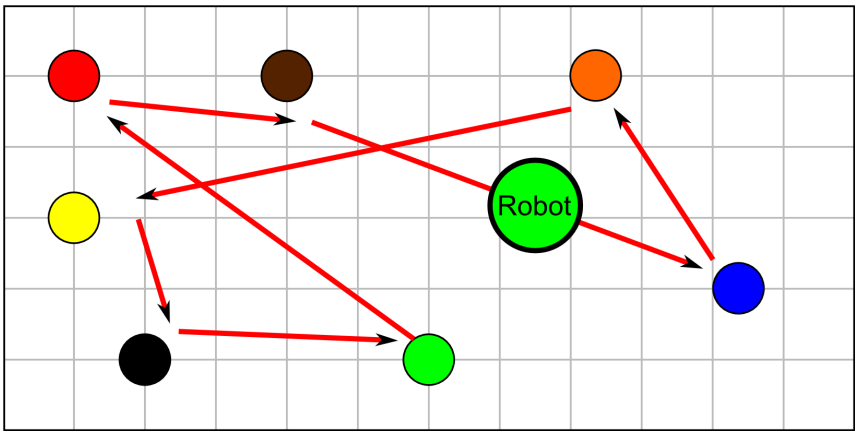
Problém samozřejmě mohl být řešen mnohem komplexněji, za využití různých pokročilých algoritmů, ale jednoduchá řešení obvykle přinášejí vyšší efektivitu, spolehlivost i robustnost.

Algoritmus je implementován v aplikaci **LineFollow**, pro zpracování obrazu je použit framework AForge.NET (více o tomto frameworku naleznete v kapitolce 5.2.2). Na přiloženém médiu je kromě této aplikace včetně zdrojových kódů dostupné video této prezentační úlohy, včetně zobrazení jednotlivých kroků algoritmu.

5.4 Barevné terčíky

Zadání úlohy

Na ploše jsou postaveny různobarevné válce přibližně o výšce 21 cm a poloměru 4 cm. Válce jsou vyrobeny z papírových tubusů, které jsou polepeny barevným papírem. Každý válec je v dolní části zatížen – pro dosažení vyšší stability. Úkolem robota je vyhledat správně barevné válce a přijet k nim na vzdálenost menší než 10 cm. Program definuje pořadí barev, ve kterém mají být tyto barvy hledány. V programu by měla být možnost doplnění dalších barev a kalibrace barev stávajících. Na obrázku 5.4.1 je zobrazeno schéma (pohled shora) této úlohy.



Obrázek 5.4.1: Schéma prezentační úlohy – „Barevné terčíky“

Řešení

Pro řešení této úlohy si již není možno zvolit různé senzory robota, ale je zapotřebí využít optickou kameru a dálkoměr. Algoritmus této úlohy je podobný jako v předchozí řešené úloze, jen je zkomplikovaný nutností rozlišování různých barev. Po získání barevného obrazu je tento obraz převeden do barevného prostoru HSL (více o tomto barevném prostoru v 5.1.1 na straně 45). Na takto převedený obraz je následně aplikován filtr, který ponechá jen určité rozmezí barev – provede filtraci podle složky H, S a L. Výsledkem po této filtraci je barevný obraz, ve kterém jsou barevně ponechány jen pixely, které odpovídaly aplikovanému filtru, zbytek obrazu je nastaven na barvu pozadí. Barva pozadí nemusí být vždy černá – proto abychom mohli rozpoznávat i úplně černé objekty. Úplně černé objekty jsou

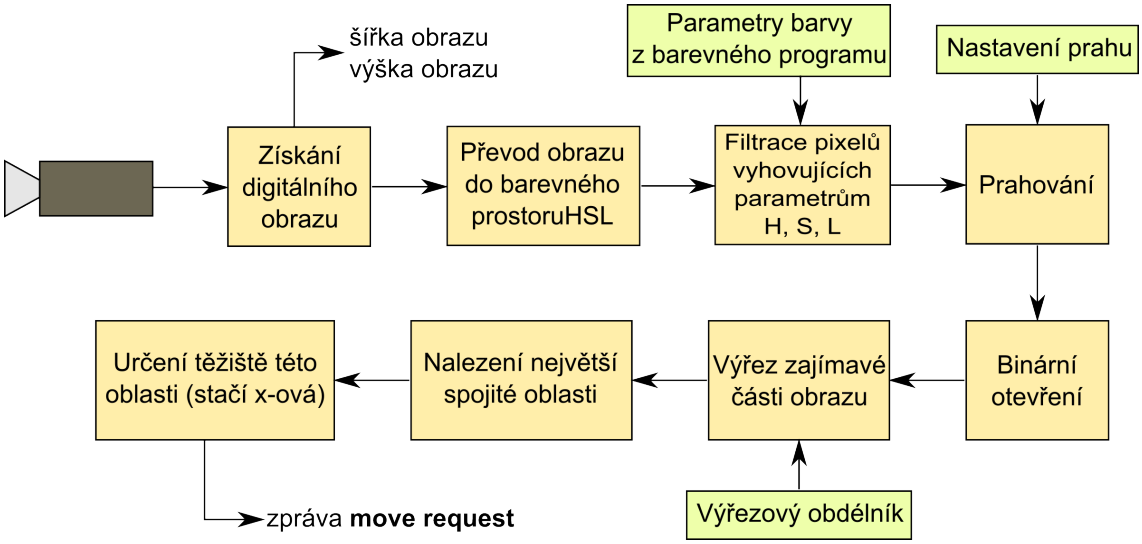
však vzácné a proto jsem si v rámci této práci dovolil zjednodušení, které nastaví barvu nevyhovujících pixelů vždy na černou.

Dalším krokem při řešení je aplikace prahování (dokonce se můžeme spojit s prahováním s pevným prahem) – práh může být nastaven nízko, protože nás zajímají i drobné odchylky od barvy pozadí, které značí přítomnost objektu požadované barvy. Následuje binární otevření, které zaplní drobné díry vzniklé barevnou nedokonalostí snímané scény a po aplikování algoritmu na nalezení největší spojité oblasti zbývá již jen k této oblasti dojet na požadovanou vzdálenost.

V případě, že není žádná dostatečně velká spojitá oblast nalezena, nevyskytuje se v dohledu kamery žádný větší objekt správné barvy. V takovémto případě se robot začne otáčet kolem své osy proti směru hodinových ručiček, dokud nenalezne dostatečně velkou oblast správné barvy.

Na obrázku 5.4.2 je zobrazen algoritmus použitý pro řešení této úlohy.

Algoritmus je implementován v aplikaci **ColorTarget**, pro zpracování obrazu je použit framework AForge.NET (více o tomto frameworku naleznete v kapitolce 5.2.2 na straně 51). Na přiloženém médiu je kromě této aplikace včetně zdrojových kódů dostupné nahrané video této prezentační úlohy, včetně zobrazení jednotlivých kroků algoritmu.



Obrázek 5.4.2: Schéma algoritmu prezentační úlohy „Barevné terčíky“

Poznámka k implementaci Barevná kamera provádí uvnitř svých obvodů automatické vyvážení bílé, což může mít negativní vliv na barevné podání jednotlivých barev. Proto je nutné při každé výraznější změně barvy osvětlení změnit i kalibrační data pro jednotlivé barvy. Kalibrace se dá velmi snadno provést přímo v programu po zobrazení okna s barvami, vybráním konkrétní barvy, kterou chceme kalibrovat a vybráním příslušné barvy z kalibračního obrazu (snímku z kamery).

Kapitola 6

Prezentační úloha s využitím plánování

6.1 Plánování

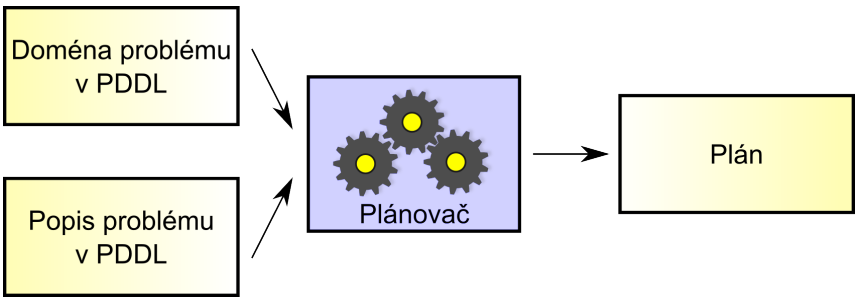
Plánování je součástí umělé inteligence a tvoří její jedno velké odvětví. Výsledkem plánování je plán (formální popis pojmů níže) – neboli přesný popis jednotlivých kroků způsobu, jakým z výchozí situace (obrazu světa), nazvané počáteční, dostaneme situaci cílovou (cílovou formuli). Pro popis plánování se využívá situační kalkul, který vychází z predikátové logiky prvního řádu (která stačí pro popis veškerého světa a nemá takové nepříjemné vlastnosti jako predikátové logiky vyšších řádů), kde je ke každému predikátu přidán situační term. Tato teorie je platná, mnohem více se však pro popis využívá systém STRIPS (Stanford Research Institute Problem Solver), což byl původně řešitel problémů, ale později se tak začal nazývat samotný jazyk pro tohoto řešitele.

Jazyk STRIPS se skládá z počátečního obrazu světa M_0 (což jsou množiny formulí predikátové logiky prvního řádu), cílové formule G (formule predikátové logiky prvního řádu) a množiny operátorů. Operátor je parciální zobrazení z $\{M_K\} \rightarrow \{M_K\}$ a je definován jako uspořádaná trojice $\varphi : (C_\varphi, D_\varphi, R_\varphi)$, kde $C\varphi$ vyjadřuje podmínku aplikovatelnosti operátoru (množina formulí), $D\varphi$ vyjadřuje množinu formulí, které mají být zařazeny do obrazu světa a $R\varphi$ vyjadřuje množinu formulí, které mají být z obrazu světa odebrány. Plánem je pak nazvána posloupnost operátorů $\varphi_1, \varphi_2, \ldots, \varphi_K$, kde $\varphi_1(M_0) = M_1$; $\varphi_2(M_1) = M_2$; $\ldots$, $\varphi_K(M_{K-1}) = M_K = G$ (ze zápisků přednášky Inteligentní roboty na TUL, přednášejícím byl docent Mgr. Ing. Václav Záda, CSc.).

Dnes nejčastěji používaným jazykem pro popis plánovací úlohy je jazyk PDDL (Planning Domain Definition Language) – jehož základní část tvoří právě jazyk STRIPS. Aktuální verzí specifikace jazyka PDDL je verze 3.1 z roku 2008 (Helmert, 2011). Pro účely plného porozumění ukázkám kódu této práce stačí znalost starší verze této specifikace a to konkrétně verze 1.2 z roku 1998 (Ghallab et al., 1998), protože nevyužívám pokročilých

možností nových specifikací. Zájemce o syntaxi a sémantiku tohoto jazyka bych rád odkázal na stránku s těmito specifikacemi, protože mi nepřipadá účelné přepisovat tyto dokumenty do této práce. Jazyk PDDL od sebe odděluje popis domény problému, který tvoří akce (v jazyce STRIPS zvané operátory) od vlastního popisu problému, který tvoří výchozí a cílový stav formulí.

Pokud máme doménu i problém zachycen v jazyce PDDL je zapotřebí sestrojit výsledný plán. O tento krok se stará program zvaný plánovač – obecný princip činnosti plánovače pro PDDL, je zobrazen na obrázku 6.1.1. Napsat vlastní primitivní plánovač není problém – v nejzákladnější verzi jde „jen“ o algoritmus prohledávání do šířky a zkoušení veškerých možných akcí – čistě jen velice krátkou grafickou ukázku tohoto způsobu plánování naleznete na obrázku 6.1.2 na následující straně, který zobrazuje řešení klasického problému „Jak pomocí 3 litrové a 4 litrové nádoby naměříme přesně 2 litry vody“. Avšak optimalizovat plánovač tak, aby byl rychlý a efektivní je dlouhodobá práce pro velké týmy matematiků a programátorů. Proto v této práci využívám plánovač zvaný Blackbox ve verzi 42 od Henryho Kautze (2003), který je šířen zdarma a je ke stažení na stránkách University of Rochester. Tento plánovač je pro malé úlohy (do několika desítek kroků) velmi dobrý.



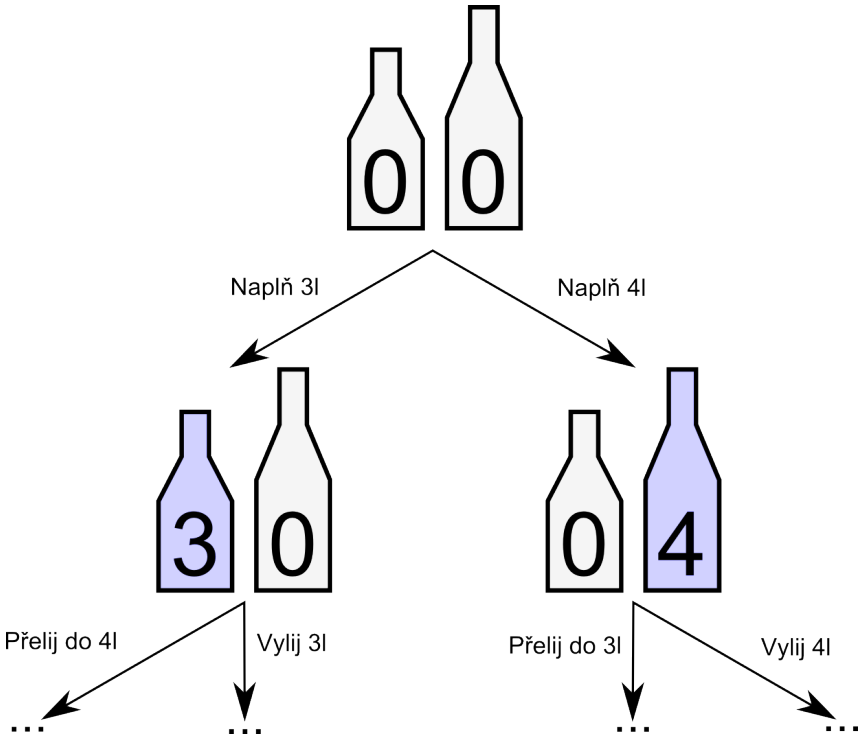
Obrázek 6.1.1: Obecné schéma PDDL plánovače

Problémem plánování, které jej činí nepoužitelným pro velké úlohy (stovky kroků) je v tom, že existuje příliš mnoho dočasných situací, které je možné převést na následující situace pomocí nějaké akce. Každá tato situace musí být vyhodnocena (i když nevede k cíli), což zpomaluje algoritmus pro nalezení plánu. Tomuto se lze bránit pomocí heuristik, kdy jsou některé situace označeny jako ty, které nevedou k cíli a proto již nejsou dále vyhodnocovány, avšak za cenu toho, že nemusí být žádný plán nalezen – situace, která vede k cíli, nemusí být díky heuristice vyhodnocena.

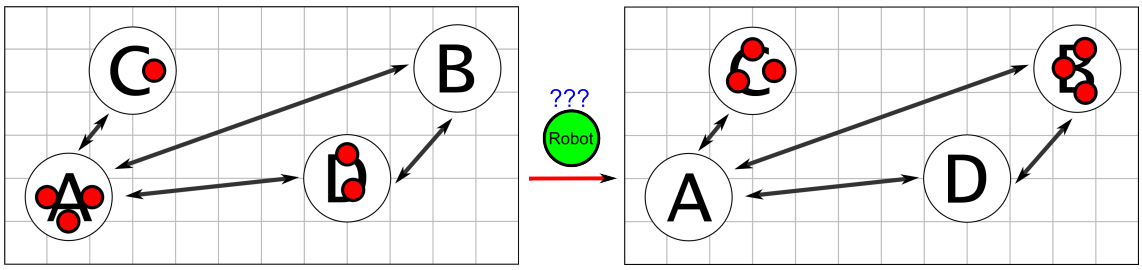
6.2 Přepravní problém

Zadání úlohy

Na ploše jsou známa místa – dopravní uzly a spojnice – cesty, mezi těmito dopravními uzly. Na jednotlivých uzlech může být umístěno několik objektů (míčků, kostek, ...). Úkolem



Obrázek 6.1.2: Grafický počátek řešení problému algoritmem prohledávání do šířky



Obrázek 6.2.1: Schéma prezentační úlohy – „Přepravní problém“

robota je přemístit tyto objekty na jiné dopravní uzly podle zadání. Pro řešení této úlohy využijte jazyk pro plánování PDDL, popište obecný problém pomocí domény a naprogramujte program pro dynamickou tvorbu zadání – popis problému. K získání plánu použijte již vytvořený plánovač. Úlohu je možno řešit pomocí simulace, nebo přímo na robotu. Na obrázku 6.2.1 je zobrazeno schéma (pohled shora) této úlohy.

Řešení

Úloha obsahuje několik postupných kroků, které je třeba vyřešit. Protože nebylo dostupné žádné rameno pro robota Robotino firmy Festo, rozhodl jsem se v souladu se zadáním úlohy zobrazit nalezený plán pouze v simulátoru. Zároveň zobrazení v simulátoru poskytuje mnohem větší variabilitu úloh – je například možné mít více přepravců (robotů) a úlohy

tak mohou být zajímavější (a výsledky plánovače překvapivější). Jednotlivými kroky se zabývám postupně v dalších částech popisu tohoto řešení.

1. Popis domény problému v jazyce PDDL.
2. Vytvoření programu pro dynamickou definici problému.
3. Simulace vyhotoveného plánu.

Popis domény problému v jazyce PDDL

Nejprve je nutné určit si typy jednotlivých objektů. Zcela určitě máme **místa** (to jsou dopravní uzly), **cesty** (to jsou spojnice mezi uzly), **náklad** (to co budeme přepravovat) a **přepravitele** (čím budeme přepravovat).

Dále je zapotřebí určit predikáty – vlastnosti jednotlivých typů, které mohou nabývat hodnoty **pravda-nepravda**. Těmito predikáty jsou **silnice (místo, místo)** – určuje zda mezi dvěma uzly vede spojnice, **věcJe (náklad, místo)** – určuje zda je daný náklad na konkrétním místě, **autoNa (přepravitel, místo)** – určuje zda je daný přepravitel na daném místě, **prázdné (přepravitel)** – určuje zda je daný přepravitel prázdný (neveze žádný náklad) a **naloženo (náklad, přepravitel)** – určuje zda je daný náklad naložen na konkrétního přepravitele. V zápise jazyka PDDL by tato úvodní definiční část vypadala jako v ukázce kódu 6.2.4 na straně 61. Plný text definice domény problému je uveden v příloze B.0.5 na straně 69.

```
  (:types misto cesta naklad prepravitel)
  (:predicates
    (silnice ?odkud - misto ?kam - misto)
    (vecJe ?co - naklad ?kde - misto)
        (autoNa ?kdo - prepravitel ?kde - misto)
        (prazdne ?kdo - prepravitel)
        (nalozeno ?co - naklad ?naKoho - prepravitel)
  )
```

Ukázka kódu 6.2.1: Definice typů a predikátů domény problému

Aby se mohl obraz světa transformovat do jiného obrazu světa je zapotřebí definovat operátory (akce). V tomto případě se bude zjevně jednat o:

- Pohyb (přesun přepravitele z jednoho místa na druhé)

```
(:action pohyb
  :parameters (?odkud ?kam - misto ?kdo - prepravitel)
  :precondition (and (silnice ?odkud ?kam) (autoNa ?kdo ?odkud))
  :effect (and (autoNa ?kdo ?kam) (not (autoNa ?kdo ?odkud)))
)
```

Ukázka kódu 6.2.2: Akce „Pohyb“ zapsaná v PDDL

Tuto akci (v jazyce STRIPS je akce nazvána operátorem) můžeme provést za současné platnosti následujících předpokladů:

1. Mezi místem **odkud** a místem **kam** vede silnice.
2. Na místě **odkud** se nachází **přepravitel**, kterého chceme mezi těmito místy přesouvat.

Tyto dva předpoklady odpovídají formulím v C_φv jazyce STRIPS. Výsledným efektem akce je vlastní přesun – z obrazu světa je odebrána formule vyjadřující vlastnost **přepravitel** je na místě **odkud** (tato formule by v jazyce STRIPS byla v R_φ) a do obrazu světa je přidána formule vyjadřující vlastnost **přepravitel** je na místě **kam** (tato formule by v jazyce STRIPS byla v D_φ).

- Nalož (v aktuálním místě nalož náklad na přepravitele)

```
(:action naloz
  :parameters (?kde - misto ?co - naklad ?naKoho - prepravitel)
  :precondition (and (prazdne ?naKoho) (vecJe ?co ?kde)
                     (autoNa ?naKoho ?kde))
  :effect (and (not (prazdne ?naKoho))
               (not (vecJe ?co ?kde)) (nalozeno ?co ?naKoho))
)
```

Ukázka kódu 6.2.3: Akce „Nalož“ zapsaná v PDDL

Tuto akci můžeme provést pokud platí všechny následující předpoklady:

1. **Přepravitel** je prázdný – nemá naložen žádný náklad.
2. **Přepravitel** se nachází na **místě** na kterém chceme nakládat.
3. **Náklad** se nachází na stejném místě jako **přepravitel**.

Efektem této akce je naložení **nákladu** na **přepravitele**. Do obrazu světa je přidána formule **naloženo(náklad, přepravitel)**, naopak jsou z obrazu světa odebrány formule **věcJe(náklad, místo)** – která vyjadřovala umístění nákladu a **prázdné(přepravitel)** – protože na přepravitele je naložen náklad. Umístění těchto formulí do R_φa D_φ je obdobné jako v předešlé akci.

- Vylož (v aktuálním místě vylož náklad z přepravitele)

```
(:action vyloz
  :parameters (?kde - misto ?co - naklad ?zKoho - prepravitel)
  :precondition (and (not (prazdne ?zKoho)) (autoNa ?zKoho ?kde)
                     (nalozeno ?co ?zKoho))
  :effect (and (prazdne ?zKoho) (vecJe ?co ?kde)
               (not (nalozeno ?co ?zKoho)))
)
```

Ukázka kódu 6.2.4: Akce „Vylož“ zapsaná v PDDL

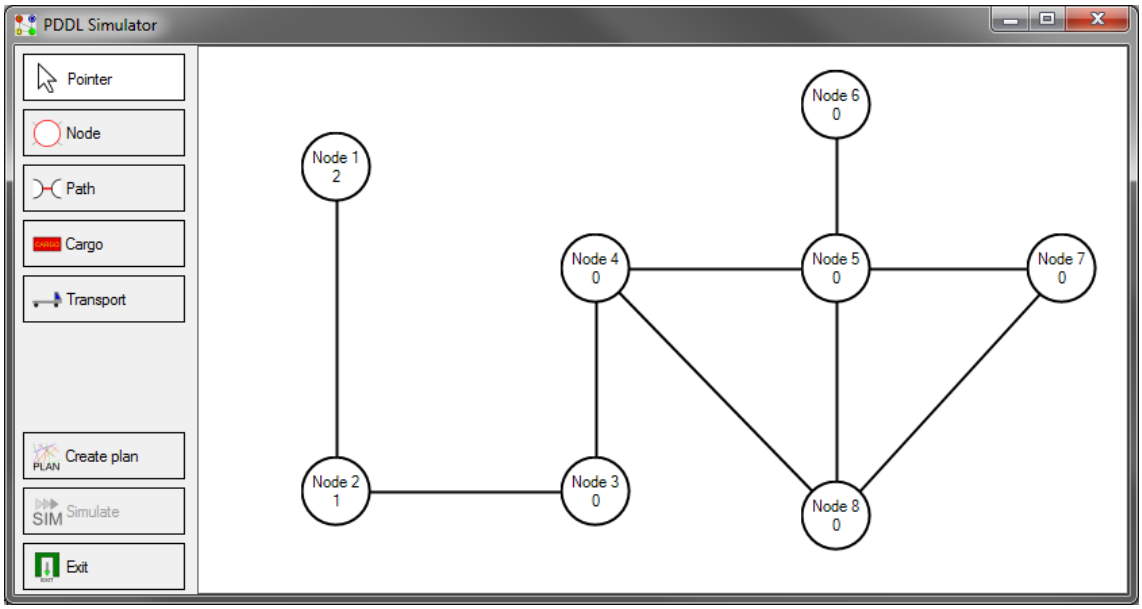
Tuto akci můžeme provést pokud platí všechny následující předpoklady:

1. **Přepravitel** je naložený – není možné vyložit prázdného přepravitele.
2. **Přepravitel** se nachází na **místě** na kterém chceme vykládat.
3. **Náklad** se nachází na tomto **přepraviteli**.

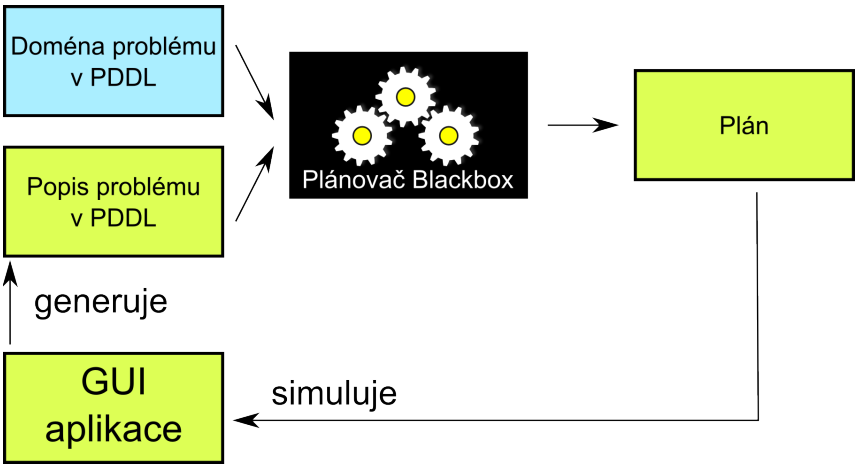
Efekt této akce je přesně opačný, než efekt u akce „Nalož“.

Vytvoření programu pro dynamickou definici problému a simulaci vyhotoveného plánu

Vytvořený program umožňuje zadávat uzly, cesty a specifikovat odkud a kam má být přepraven náklad. V programu je také možné určit výchozí uzel a počet přepravujících přepravitelů (symbolicky znázorněných obrázkem nákladního automobilu). Uživatelské rozhraní (zobrazeno na obrázku 6.2.2 na následující straně) programu je přívětivé a jednoduché a nepočítám s tím, že by s ovládáním měli problémy i méně znalí uživatelé. Program je napsán v jazyce C#, pro plánování však využívám externího plánovače – v tomto případě používám již výše zmíněný plánovač Blackbox. Schéma spolupráce jednotlivých součástí řešení této úlohy je zobrazeno na obrázku 6.2.3 na následující straně. Plánovač je volán jako externí program a je tak možné jej nahradit za jiný a pozorovat tak například odlišné chování plánovačů v různých situacích.



Obrázek 6.2.2: Hlavní okno aplikace pro dynamickou definici problému a simulaci vyhotoveného plánu



Obrázek 6.2.3: Schéma spolupráce jednotlivých částí řešení úlohy „Přepravní problém“

Program umožňuje i simulaci vytvořeného plánu pomocí animace a poskytuje tak grafickou nadstavbu pro vizualizaci plánu z plánovače.

Kompletní popis problému tak jak jej vygeneruje program v situaci zadané na obrázku – existují 2 přepravitelé (jeden startuje v uzlu 3 a druhý v uzlu 6) a je zapotřebí převézt náklad z uzlu 1 do uzlu 7 a 8 a z uzlu 2 do uzlu 5 – je přiložen v příloze B.0.6 na straně 70. V příloze B.0.7 na straně 71 je přiložen kompletní plán pro splnění cíle.

Na přiloženém médiu je kromě této aplikace včetně zdrojových kódů dostupné nahrané video z práce v tomto programu.

Kapitola 7

Závěr

Dle zadání byla v úvodní kapitole provedena rešerše informačních zdrojů, které se dotýkaly tématu, nebo zaměření této práce.

Ve druhé kapitole jsem se věnoval celkovému popisu mobilního robota Robotino od firmy Festo a určitý prostor jsem věnoval i jeho některým senzorům a efektorům. Dálkoměry použité v tomto robotu nejsou opatřeny dostatečně přesnou dokumentací od výrobce a proto jsem provedl vlastní měření závislosti napětí na vzdálenosti od senzoru. Z naměřených údajů jsem sestrojil tabulku, graf a aproximační polynom, který byl důležitou součástí pro pozdější praktickou implementaci.

Ve třetí kapitole jsem se podrobně zabýval ideovým a funkčním návrhem komunikačního protokolu. V kapitole byla podrobně vysvětlena a zdůvodněna komunikace prostřednictvím zpráv a také byla podrobně navržena struktura těchto zpráv – včetně jednotlivých parametrů zpráv různých typů. Ve druhé části této kapitoly byly stručně podány detaily implementace navrženého komunikačního protokolu – plná implementace včetně technické dokumentace je součástí přiloženého média. Na závěr této kapitoly bylo krátce představeno uživatelské rozhraní programu pro ladění a inspekci zpráv navrhovaného řešení.

Ve čtvrté kapitole bylo ukázáno uživatelské rozhraní univerzálního programu pro ovládání libovolného mobilního robota, který by implementoval navržené řešení komunikačního protokolu. V rámci této práce je možno tímto programem ovládat mobilního robota Robotino firmy Festo z důvodu implementace zprostředkovatelské aplikace pouze pro tohoto robota.

V úvodu páté kapitoly jsem vysvětlil a matematicky popsal některé pojmy a algoritmy počítačového vidění a podal jsem přehled a porovnání dostupných počítačových knihoven zaměřených na řešení úloh počítačové vidění. V hlavní části této kapitoly jsem se věnoval zadání a a podrobnému popisu řešení dvou prezentačních úloh – jízda robota po čáře a jízda mezi barevnými terčíky. Praktickou implementací těchto úloh jsem ověřil nejen správnost návrhu algoritmického řešení každé z těchto úloh, ale zároveň jsem otestoval i návrh a

provedenou implementaci komunikačního protokolu v reálných podmínkách provozu, kde navrhované řešení obstálo.

Šestá kapitola představila část umělé inteligence – plánování. Na začátku kapitoly byl podán teoretický a matematický úvod do teorie plánování. Větší část této kapitoly byla zaměřena na praktické řešení transportní úlohy spojením klasického programovacího jazyka a jazyka pro plánování úloh PDDL. V klasickém programovacím jazyce byla navržena a napsána aplikace pro dynamickou tvorbu zadání transportní úlohy, v jazyce PDLL byl vytvořen popis domény problému a za pomoci externího plánovače byla úloha vyřešena a výsledek byl poté simulován zpět v aplikaci.

V praxi je možné využít výsledků této práce především v oblasti robotiky, kdy je dán konstruktérům robotů již hotový návrh komunikačního protokolu, který mohou využít ve svůj prospěch. Tito konstruktéři se tak nemusí zabývat implementací vlastního ovládacího programu a zároveň mohou dát všem uživatelům svých robotů svobodnou volbu pro výběr vlastního ovládacího prostředí. Tato práce může být využita i ve výuce předmětů souvisejících s robotikou, počítačovým viděním, nebo při dnech otevřených dveří a to díky zpracovaným prezentačním úlohám, které jsou na přiloženém médiu uloženy i ve formě videa.

Na práci lze dále navázat implementací zprostředkujících aplikací pro další roboty nebo rozšířením plánovací úlohy o fyzickou implementaci na některém z robotů.

Literatura

AFORGE.NET. 2012. *AForge.NET :: Framework* [online]. [Cit. 12. března 2012]. Dostupné z: <http://www.aforgenet.com/framework/>.

APACHE SOFTWARE FOUNDATION. 2011. *Apache log4net* [online]. [Cit. 21. února 2012]. Dostupné z: <http://logging.apache.org/log4net/>.

BALLUFF CZ. 2011. *Balluff CZ s.r.o. (Indukční snímače - Principy, popisy funkcí, definice)* [online]. [Cit. 31. ledna 2012]. Dostupné z: <http://www.balluff.cz/bes_principy-funkce-definice.asp>.

BARTÁK, R. 2011. *Umělá inteligence I - Úvod, témata a vymezení UI, pohled do historie, úspěšné aplikace* [online]. [Cit. 7. února 2012]. Dostupné z: <http://kti.mff.cuni.cz/~bartak/ui/lectures/lecture01.pdf>.

BRADLEY, D. – ROTH, G. 2007. *Adaptive thresholding using the integral image* [online]. *Journal of graphics, GPU, and game tools*. 12, 2, s. 13–21. doi: 10.1080/2151237X.2007.10129236. Dostupné z: <http://www.tandfonline.com/doi/abs/10.1080/2151237X.2007.10129236>.

EMGU CV. 2012. *Main Page - Emgu CV: OpenCV in .NET (C#, VB, C++ and more)* [online]. [Cit. 12. března 2012]. Dostupné z: <http://www.emgu.com/wiki/index.php/Main_Page>.

EVOLUTION ROBOTICS. 2008a. *Northstar II Projector* [CD-ROM]. Evolution Robotics, Revision E edition.

EVOLUTION ROBOTICS. 2008b. *Northstar II Detector* [CD-ROM]. Evolution Robotics, Revision H edition.

FEIPENG, L. 2011. *YCbCr Color Space–An Intro and its Applications* [online]. [Cit. 27. února 2012]. Dostupné z: <http://www.roman10.net/ycbcr-color-spacean-intro-and-its-applications/>.

FESTO DIDACTIC. 2009. *FESTO Robotino® Manual* [CD-ROM]. Festo Didactic GmbH & Co. KG, v2 edition.

GHALLAB, M. et al. 1998. *PDDL - the planning domain definition language* [online]. [Cit. 28. března 2012]. Dostupné z: <http://www.informatik.uni-ulm.de/ki/Edu/Vorlesungen/GdKI/WS0203/pddl.pdf>.

HELMERT, M. 2011. *PddlResources - IPC-2008, Deterministic Part* [online]. [Cit. 23. dubna 2012]. Dostupné z: <http://ipc.informatik.uni-freiburg.de/PddlResources>.

HORÁK, K. et al. 2008. *Počítačové vidění* [online]. [Cit. 5. března 2012]. Dostupné z: <http://www.uamt.feec.vutbr.cz/vision/TEACHING/MPOV/Pocitacove_videni_S.pdf>.

CHALOUPKA, J. 2010. *Segmentace obrazu, popis oblastí* [online]. [Cit. 28. února 2012]. [Přístup po přihlášení]. Dostupné z: <https://www.ite.tul.cz/vyuka/mod/resource/view.php?id=278>.

CHENG, H. D. et al. 2001. *Color image segmentation: advances and prospects* [online]. *Pattern Recognition*. 34, 12, s. 2259 – 2281. ISSN 0031-3203. doi: 10.1016/S0031-3203(00)00149-7. Dostupné z: <http://www.sciencedirect.com/science/article/pii/S0031320300001497>.

KAUTZ, H. 2003. *Blackbox* [online]. [Poslední revize 15. ledna 2003], [cit. 23. dubna 2012]. Dostupné z: <http://www.cs.rochester.edu/u/kautz/satplan/blackbox/>.

KONTRON. 2004. *MOPSlcdVE* [CD-ROM]. Kontron.

KOSEK, J. 2011. *XSLT v příkladech* [online]. [Cit. 21. února 2012]. Dostupné z: <http://www.kosek.cz/xml/xslt/>.

ODVÁRKA, P. 2000. *TCP handshake krok za krokem* [online]. [Cit. 13. února 2012]. Dostupné z: <http://www.svetsiti.cz/clanek.asp?cid=2315>.

OPENCV. 2012. *Welcome - OpenCV Wiki* [online]. [Poslední revize 24. února 2012], [cit. 12. března 2012]. Dostupné z: <http://opencv.willowgarage.com/wiki/Welcome>.

ŘÍČNÝ, V. 2010. *Tajemství systému Quattron: jak se Sharpu podařilo do televizorů dostat žlutou* [online]. [Cit. 27. února 2012]. Dostupné z: <http://www.digizone.cz/clanky/tajemstvi-systemu-quattron-zlute-od-sharpu/>.

SHARP. 2005. *Analog output type distance measuring sensor - model no. GP2D120XJ00F* [CD-ROM]. SHARP Corporation.

SZELISKI, R. 2010. *Computer Vision: Algorithms and Applications* [online]. [Cit. 27. dubna 2012]. Dostupné z: <http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf>.

[TEZAUR], R. 2003. *Barevné prostory* [online]. *Paladix foto on-line...* ISSN 1213-5704. Dostupné z: <http://www.paladix.cz/clanky/barevne-prostory.html>.

ÚSTAV PRO JAZYK ČESKÝ AKADEMIE VĚD ČR, V. V. I.. 2012. *Internetová jazyková příručka: robot* [online]. [Poslední revize 1. prosince 2008], [cit. 5. května 2012]. Dostupné z: <http://prirucka.ujc.cas.cz/?id=robot>.

VISION-SUPPLIES.COM. [200-?]. *Festo SIEA-M12B-UI-S* [online]. [Cit. 31. ledna 2012]. Dostupné z: <http://www.vision-supplies.com/Product.aspx?pid=87656>.

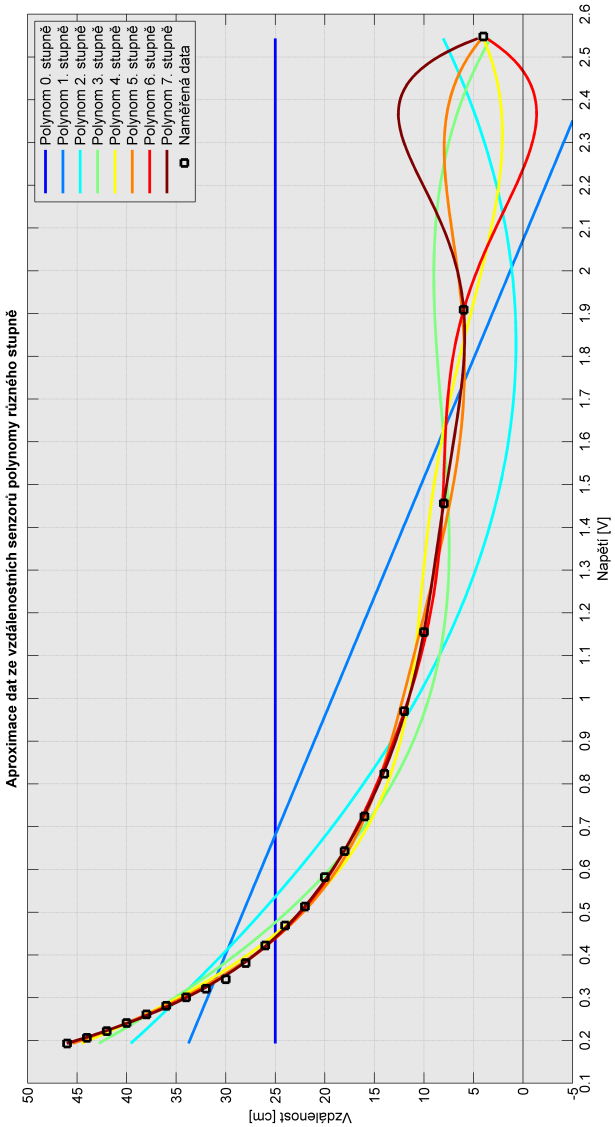
VOLNÁ, E. 2005. *VYBRANÉ PARTIE UMĚLÉ INTELIGENCE* [online]. [Cit. 22. února 2012]. Dostupné z: <http://www1.osu.cz/~volna/Vybrane_partie_UI_1_dil_skripta.pdf>.

W3SCHOOLS. [200-?]. *XSLT Tutorial* [online]. [Cit. 21. února 2012]. Dostupné z: <http://www.w3schools.com/xsl/>.

WIKIPEDIA CONTRIBUTORS. 2012. *Endianness — Wikipedia, the free encyclopedia* [online]. [Poslední revize 9. února 2012], [cit. 10. února 2012]. Dostupné z: <http://en.wikipedia.org/w/index.php?title=Endianness&oldid=476037017>.

Příloha A

Obrázky



Obrázek A.0.1: Aproximace dat ze vzdálenostních senzorů polynomy různého stupně

Příloha B

Ukázky kódu v jazyce PDDL

```
(define (domain preprava)
  (:requirements :strips :typing)
  (:types misto cesta naklad prepravitel)
  (:predicates
    (silnice ?odkud - misto ?kam - misto)
    (vecJe ?co - naklad ?kde - misto)
        (autoNa ?kdo - prepravitel ?kde - misto)
        (prazdne ?kdo - prepravitel)
        (nalozeno ?co - naklad ?naKoho - prepravitel)
  )
  (:action pohyb
    :parameters (?odkud ?kam - misto ?kdo - prepravitel)
    :precondition (and (silnice ?odkud ?kam) (autoNa ?kdo ?odkud))
    :effect (and (autoNa ?kdo ?kam)
                 (not (autoNa ?kdo ?odkud)))
  )
  (:action naloz
    :parameters (?kde - misto ?co - naklad ?naKoho - prepravitel)
    :precondition (and (prazdne ?naKoho) (vecJe ?co ?kde)
                       (autoNa ?naKoho ?kde))
    :effect (and (not (prazdne ?naKoho))
                 (not (vecJe ?co ?kde))
                 (nalozeno ?co ?naKoho))
  )
  (:action vyloz
    :parameters (?kde - misto ?co - naklad ?zKoho - prepravitel)
    :precondition (and (not (prazdne ?zKoho))
                       (autoNa ?zKoho ?kde) (nalozeno ?co ?zKoho))
    :effect (and (prazdne ?zKoho)
                 (vecJe ?co ?kde)
                 (not (nalozeno ?co ?zKoho)))
  )
)
```

Ukázka kódu B.0.5: Kompletní výpis definice domény problému

```
(define (problem preprava-problem)
  (:domain preprava)
  (:objects
    Node1 Node2 Node3 Node4 Node5 Node6 Node7 Node8  - misto
    Transport1 Transport2  - prepravitel
    Cargo1 Cargo2 Cargo3  - naklad
  )
  (:init
    (prazdne Transport1)
    (autoNa Transport1 Node6)
    (prazdne Transport2)
    (autoNa Transport2 Node3)
    (vecJe Cargo1 Node1)
    (vecJe Cargo2 Node1)
    (vecJe Cargo3 Node2)
    (silnice Node1 Node2) (silnice Node2 Node1)
    (silnice Node2 Node3) (silnice Node3 Node2)
    (silnice Node3 Node4) (silnice Node4 Node3)
    (silnice Node4 Node5) (silnice Node5 Node4)
    (silnice Node5 Node6) (silnice Node6 Node5)
    (silnice Node5 Node7) (silnice Node7 Node5)
    (silnice Node5 Node8) (silnice Node8 Node5)
    (silnice Node8 Node7) (silnice Node7 Node8)
    (silnice Node4 Node8) (silnice Node8 Node4)
  )
  (:goal (and
    (vecJe Cargo1 Node8)
    (vecJe Cargo2 Node7)
    (vecJe Cargo3 Node5)
  ))
)
```

Ukázka kódu B.0.6: Popis problému v jazyce PDDL

```
1 (pohyb node6 node5 transport1)
1 (pohyb node3 node2 transport2)
2 (pohyb node5 node4 transport1)
2 (naloz node2 cargo3 transport2)
3 (pohyb node4 node3 transport1)
3 (pohyb node2 node3 transport2)
4 (pohyb node3 node4 transport2)
4 (pohyb node3 node2 transport1)
5 (pohyb node4 node5 transport2)
5 (pohyb node2 node1 transport1)
6 (vyloz node5 cargo3 transport2)
6 (naloz node1 cargo2 transport1)
7 (pohyb node5 node4 transport2)
7 (pohyb node1 node2 transport1)
8 (pohyb node4 node3 transport2)
8 (vyloz node2 cargo2 transport1)
9 (pohyb node3 node2 transport2)
9 (pohyb node2 node1 transport1)
10 (naloz node2 cargo2 transport2)
10 (naloz node1 cargo1 transport1)
11 (pohyb node2 node3 transport2)
11 (pohyb node1 node2 transport1)
12 (pohyb node3 node4 transport2)
12 (pohyb node2 node3 transport1)
13 (pohyb node4 node5 transport2)
13 (pohyb node3 node4 transport1)
14 (pohyb node5 node7 transport2)
14 (pohyb node4 node8 transport1)
15 (vyloz node8 cargo1 transport1)
15 (vyloz node7 cargo2 transport2)
```

Ukázka kódu B.0.7: Plán pro řešení problému v dané doméně

Příloha C

Obsah přiloženého média

Přiložené médium obsahuje následující stromovou strukturu

```
/
├── Programy
│   ├── PDDL ... Složka obsahuje popis domény v jazyce PDDL.
│   ├── Zdrojové soubory ... Složka obsahuje solution pro MS Visual Studio
│   │                        2010 s projekty jednotlivých programů vytvořených
│   │                        v rámci této diplomové práce.
│   └── Zkompilované programy ... Složka obsahuje jen přeložené programy určené pro
│                                 přímé spouštění.
├── Protokol ... Složka obsahuje veškeré podklady ke komunikačnímu
│               protokolu.
│   ├── ComLib ... Implementace komunikačního protokolu v jazyce C#.
│   │   ├── Dokumentace ... HTML dokumentace ke knihovně.
│   │   └── Knihovna ... Knihovna implementující komunikační protokol.
│   └── XMLSchema ... XML schema textové verze komunikačního protokolu.
├── Text ... Složka obsahuje text diplomové práce v několika
│           formátech.
│   ├── Bib ... Bibliografie ve formátu .bib, která byla použitá
│   │           v této práce.
│   ├── Dálkoměr - měření ... Obsahuje XML soubor se surovými naměřenými daty
│   │                         z dálkoměru.
│   └── Obrázky ... Obsahuje veškeré obrázky použité v této práci,
│                   včetně obrázků ESF.
└── Video ... Složka s videoukázkami.
    ├── Úloha 1 ... Obsahuje videozáznam první prezentační úlohy a
    │               dalších dílčích materiálů.
    ├── Úloha 2 ... Obsahuje videozáznam druhé prezentační úlohy a
    │               dalších dílčích materiálů.
    └── Úloha 3 ... Obsahuje videozáznam práce s PDDL simulátorem.
```

Poděkování: Tento materiál vznikl v rámci projektu ESF (CZ.1.07/2.2.00/07.0247)
Reflexe požadavků průmyslu na výuku v oblasti automatického řízení a měření.
Formát zpracování originálu: titulní list barevně, další listy včetně příloh barevně.