SLEZSKÁ UNIVERZITA V OPAVĚ

Filozoficko-přírodovědecká fakulta

Ústav informatiky

Kooperace robotů při řešení úlohy Box-pushing

Magisterská diplomová práce

Vypracoval: Bc. Michal Nemrava

Vedoucí práce: Doc. Ing. Petr Čermák, Ph.D.

Opava 2007

Prohlášení:

Prohlašuji, že jsem tuto diplomovou práci zpracoval samostatně a uvedl veškerou literaturu a ostatní zdroje, které jsem použil.

V Opavě, ……... ……...…….……

Poděkování

Na tomto místě bych chtěl poděkovat vedoucímu své diplomové práce Doc. Ing. Petru Čermákovi, Ph.D. za cenné rady a připomínky při zpracování diplomové práce.

OBSAH

1 Úvod

Úloha Box-Pushing je již dlouhou dobu jednou ze základních úloh, kterou se zabývají výzkumné týmy v laboratořích robotiky po celém světě. Tato úloha je velmi oblíbená i proto, že při jejím řešení je nutno navrhnout a implementovat řadu dílčích podúloh a také navrhnout způsob jejich propojení a složení pro vyřešení celé úlohy. Box-Pushing jistě může najít mnoho praktických uplatnění v různých oblastech.

Cílem práce je navrhnout a implementovat řešení této úlohy za pomoci kooperace a komunikace dvou robotů Khepera. Navrhneme dekompozici řešení do více jednodušších podúloh, které v praktické části práce implementujeme v jazyku C a kompletní řešení úlohy otestujeme v reálném prostředí.

Ve druhé kapitole uvedeme hlavní cíle a zaměření kooperativní mobilní robotiky a popíšeme zadání úlohy Box-Pushing a možnosti jejího řešení.

V další kapitole popíšeme robotickou platformu, roboty Khepera, jejich vybavení, možnosti programování a přídavné moduly rozšiřující jejich funkčnost.

V dalších následujících kapitolách se budeme věnovat popisu dekompozice celé úlohy Box-Pushing do menších funkčních částí, popis algoritmů vedoucích k řešení a jejich implementaci v jazyku C. Popíšeme také implementaci všech důležitých pomocných funkcí a funkcí zajišťujících potřebné části řešení (pohyb robotů, snímání obrazu, práce se senzory, zasílání a přijímání zpráv a další).

V posledních kapitolách popíšeme zpětnou kompozici jednotlivých dílčích podúloh v řešení úlohy a popis funkce jednotlivých procesů běžících na obou robotech a výsledky a průběh experimentu.

Na přiloženém disku CD jsou k dispozici videa ve formátu MPEG-2 pořízená v laboratoři a zachycující experiment.

V práci budeme pro názornost a pro vysvětlení použitého řešení uvádět části kódu nebo implementaci celých funkcí v jazyku C s jejich popisem.

2 Kooperativní robotika a Box-Pushing

2.1 Základní charakteristika

Koordinace více robotů a možnosti jejich spolupráce jsou předmětem zájmu výzkumu a experimentů kooperativní mobilní robotiky. Podstatou všech těchto experimentů je návrh vhodných algoritmů pro řešení úloh, které jeden robot sám nedokáže vyřešit, nebo návrh takových postupů, při nichž spolupráce více robotů značně snižuje náklady a čas pro splnění úkolu [1].

Úlohy kooperativní mobilní robotiky tedy můžeme rozdělit na ty, při kterých je nutná kooperace více robotů (např. přesun těžkého nebo velkého předmětu) a úlohy, které sice jeden robot vyřešit může, ale řešení takové úlohy více roboty může přinést mnoho výhod, jako jsou. úspora času nebo prostředků [2]. Příkladem takových úloh mohou být například tyto typy úloh:

- Mapování prostředí - použití více komunikujících a kooperujících robotů při vytváření mapy prostředí může přinést značné zrychlení, když každý robot prochází jinou část prostředí a buduje tak jinou část společné mapy.
- Shromažďování (angl. Foraging) - přesun určitých předmětů na jedno místo. Použití více robotů snižuje čas potřebný ke shromáždění všech těchto předmětů.

Dalším důvodem a motivací pro použití více spolupracujících robotů může být fakt, že výroba a naprogramování více jednodušších robotů může být mnohem levnější a jednodušší než konstrukce jednoho komplexního robotu.

Dalším jistě velmi důležitým faktorem je i robustnost takového společenství robotů. Porucha jednoho z robotů nemusí nutně znamenat neúspěch všech ostatních robotů, ale ostatní roboty mohou pokračovat v řešení a zastat úkoly a nahradit funkce tohoto robotu.

Návrhy a architektura společenství kooperujících robotů jsou obecně založeny na dvou různých přístupech – na centralizovaném nebo decentralizovaném [3].

Decentralizovaný přístup spočívá v tom, že každý robot provádí svou činnost a plní své úkoly bez znalosti pozice a ostatních informací jiných robotů. To znamená, že ve skupině neexistuje žádný nadřazený prvek a všechny roboty jsou si funkčně rovny [4]. Každý robot sice může plnit svou vlastní úlohu, ale není řízen žádným jiným členem skupiny. V *centralizovaném přístupu* existuje jeden nadřazený centrální systém (robot), který má k dispozici informace o stavu a pozici ostatních robotů. Tyto informace centrální robot získá buď měřením nebo pomocí zprávy od ostatních robotů o jejich stavu. Získané informace centrální robot vyhodnocuje a určuje akce a úkoly pro ostatní jednotlivé roboty. Úkoly jim pak předává např. formou zasílání zpráv.

Nevýhodou centralizovaného přístupu je již zmíněná existence jednoho centrálního prvku. Při výpadku nebo poruše tohoto řídícího prvku dojde k selhání funkcí všech ostatních robotů a zastavení provádění celého úkolu.

Výzkum v oblasti kooperace více robotů při plnění nějaké předem určené úlohy se proto věnuje jak návrhu centralizovaných, tak i decentralizovaných řešení.

2.2 Box-Pushing

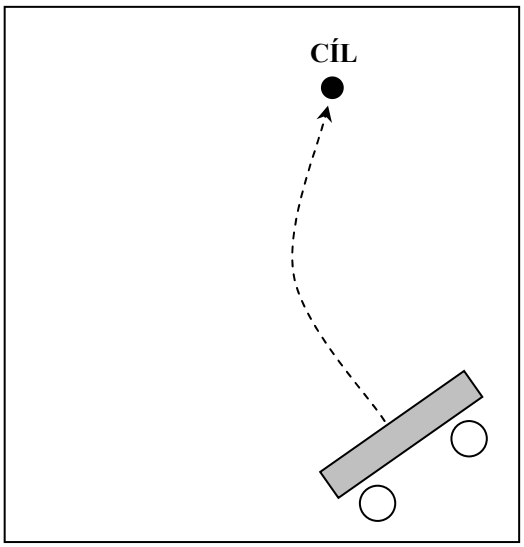
Box-pushing je velmi vhodnou úlohou pro zkoumání různých přístupů a analýzu řešení kooperace. Tento problém poskytuje velmi zajímavou teoretickou oblast a jeho následné řešení má mnoho možných praktických využití.

2.2.1 Popis úlohy

Cílem úlohy Box-Pushing je nalézt (lokalizovat) a přemístit box (nebo jiný předmět) na určené místo v prostředí. Existuje mnoho variant zadání této úlohy. Jednotlivé varianty se mohou lišit například způsobem zadání cíle, způsobem odlišení boxu v prostředí od ostatních předmětů a stěn, velikostí a tvarem předmětu atd..

Zajímavým experimentem tlačení boxu jedním robotem je [5], ve kterém je popsáno tlačení předmětu mezi dvěma stěnami pouze jedním robotem. V případě využití více robotů pro řešení této úlohy se varianty mohou lišit počtem kooperujících robotů, způsobem komunikace mezi roboty nebo například stupněm centralizace/decentralizace.

Cílové místo pro přesun boxu může být vyznačeno osvětleným místem, odlišeno určenou barvou v prostředí nebo např. v případě implementovaného GPS (Global Position System) určenými souřadnicemi.



Obrázek 1 - Box-Pushing

2.2.2 Centralizovaný přístup

Typickým příkladem centralizovaného řešení úlohy box-pushing s existencí jednoho řídícího robotu je [6]. Toto řešení je založeno na spolupráci dvou tlačících robotů a jednoho robotu, který sleduje okolí a stav prostředí a koordinuje tyto roboty k dosažení cíle. Tlačící roboty nejsou vybaveny kamerou a ani žádným jiným způsobem nemohou vnímat okolí. Pouze přijímají zprávy od řídícího robotu, který určuje další postup.

2.2.3 Decentralizovaný přístup

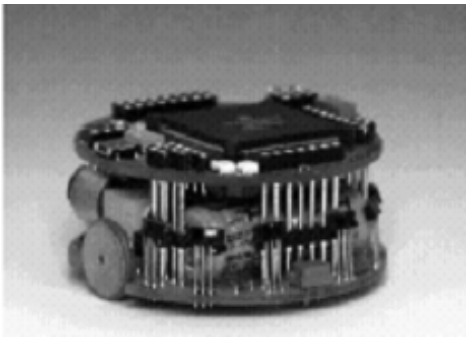
Příkladem experimentů s decentralizovanou architekturou jsou např. [7],[8] a [9]. Tato řešení jsou založena na definici úloh jednotlivých robotů a jejich společném provádění bez použití nějaké explicitní komunikace mezi roboty. V [10] je uveden návrh plně distribuované architektury pro dosažení kooperace robotů. Jednou z úloh při testování byla i úloha tlačení boxu, který svými vlastnostmi přesahoval možnosti tlačení jedním robotem a proto byla nutná kooperace dvou robotů.

3 Roboty Khepera

Roboty Khepera jsou miniaturní roboty, ale svými schopnostmi a vlastnostmi srovnatelné s většími roboty používanými ve výzkumu, ve výuce a dalších oblastech.

3.1 Vlastnosti a vybavení

Roboty Khepera (viz Obrázek 2) byly již od počátku svého návrhu projektovány tak, aby mohly být použity při výzkumu v oblasti mobilní robotiky. Právě díky jejich malým rozměrům (55mm v průměru a výšce 30mm) a díky nízké hmotnosti (asi 90g) jsou ideální pro experimenty v oblasti mobilní robotiky, při kterých potřebujeme řešit úlohy, při nichž se roboty pohybují v omezeném prostředí. S využitím těchto malých mobilních robotů můžeme i na malé ploše pracovat v prostředí, které by v případě použití větších robotů bylo naprosto nevhodné.



Obrázek 2 - Základní modul Khepera - pohled z boku

Khepera je osazen 32-bitovým procesorem Motorola 68331 s 256kB RAM a 256kB ROM paměti. Tato paměť je pro naprostou většinu aplikací dostačující. Průměrný program pracující s více procesy má velikost kolem 100kB.

Khepera je vybaven dvěma nezávislými koly poháněnými servomotorky, které umožňují robotu pohyb různými směry a různými rychlostmi. Právě použití pouze dvou kol poskytuje dobrou ovladatelnost a řízení pohybu robotu v prostředí.

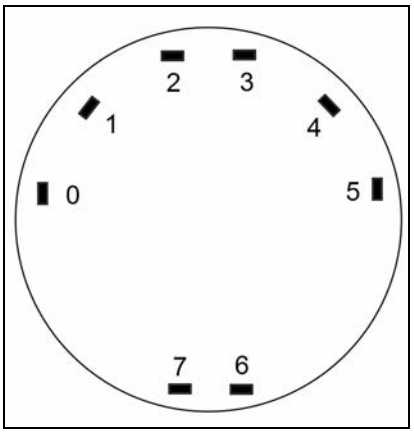
Okolí Khepera rozpoznává a mapuje pomocí osmi infračervených senzorů Siemens SFH900, které jsou rozmístěny po obvodu robotu (viz Obrázek 3). Každý senzor je

vybaven infračerveným vysílačem (infra-red light emitter) a infračerveným přijímačem (infra-red light receiver) [11],[12].

Pomocí těchto senzorů je možné provádět dva typy měření:

- Měření intenzity okolního světla - pouze za pomoci infračerveného přijímače. Frekvence snímání je každých 20ms. Naměřené hodnoty jsou z intervalu (0,1024). Hodnoty se zmenšují s rostoucí intenzitou okolního světla. Standardní hodnota v tmavém prostředí je asi 400.

- Měření vzdálenosti okolních objektů - naměřené hodnoty jsou závislé na mnoha faktorech jako jsou např. barva a povrch objektu, intenzita osvětlení objektu a samozřejmě vzdálenost. Naměřené hodnoty jsou z intervalu 0-1024, kdy 0 vyjadřuje, že senzor nezjistil ve svém dosahu žádný objekt.



Obrázek 3 - Rozmístění IR senzorů

Čtyři baterie umožňují robotu být v provozu až 30 minut bez dobití baterií. Robot může být také neustále připojen k adaptéru.

3.2 Režimy práce

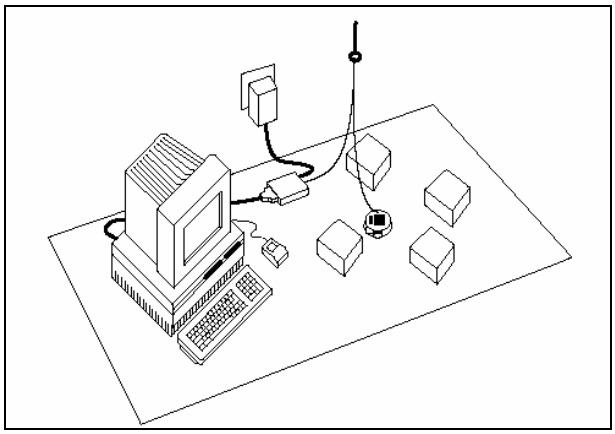
Khepera může pracovat ve dvou režimech - autonomně (samostatně) nebo připojený k počítači.

- Autonomní mód

  Řídící program je uložen přímo do paměti robotu a ten nemusí být připojen k hostitelskému počítači. Výhodou tohoto způsobu práce je rychlost, neboť program běží přímo na procesoru robotu a komunikace mezi procesorem a pamětí je rychlá. Jednou z hlavních nevýhod tohoto řešení je obtížné ladění programů, neboť při výskytu chyby nebo při úpravě jakékoliv části programu je nutné celý program na počítači znovu zkompilovat a překopírovat do paměti robotu. Toto kopírování přes sériové rozhraní může trvat řádově až desítky sekund v závislosti na velikosti programu.

- Připojení k počítači přes sériové rozhraní

  Řídící program na počítači získává veškeré informace ze senzorů robotu přes toto sériové rozhraní a také přes toto rozhraní předává všechny pokyny pro ovládání robotu (např. rychlost jednotlivých motorků apod.). Výhodou tohoto připojení je především to, že řídící program není omezen velikostí paměti robotu.



Obrázek 4 - Připojení robotu přes sériové rozhraní

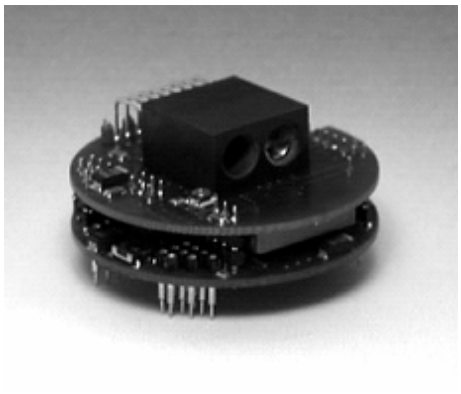
Další výhodou je možnost ladění programů oproti autonomní práci robotu. Naopak, ale může dojít ke komplikacím z důvodu pomalé komunikace mezi pamětí v počítači a procesorem robotu, s následkem „opožděné“ reakce na informace ze senzorů.

3.3 Rozšiřující moduly

Pro roboty Khepera je charakteristická jejich modularita. Každý robot může být vybaven několika rozšiřujícími moduly. Výrobce (K-Team) nabízí mnoho těchto modulů pro rozšíření funkcí a schopností robotů Khepera. Připojení a manipulace s těmito moduly je velice jednoduchá, protože všechny moduly je možné na základní modul připojovat a vrstvit jako díly stavebnice.

Moduly, které budeme využívat při implementaci jsou K213 Vision Turret a Radio Turret.

K213 Vision turret – lineární kamera



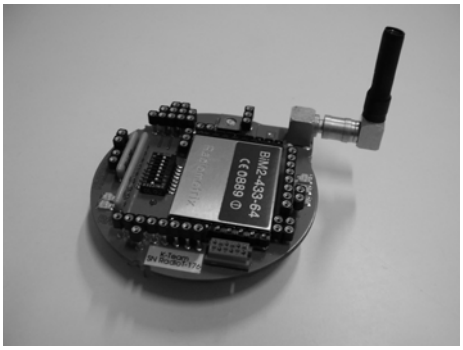
Obrázek 5 - K213 Vision

Rozšiřující modul, který přidává robotům Khepera schopnost „vidění“ ve stupních šedi [13]. Vzhledem ke tvaru a konstrukci tohoto modulu může tento být umístěn jen na vrcholu jako nejvyšší modul, tzn. nemohou na něj být nasazeny žádné další moduly. Tento modul je tvořen dvěma typy senzorů:

- Senzorem lineárního vidění
  - výstupem je vektor 64 pixelů, každý pixel je určen hodnotou z 256 stupňů šedi
  - zorný úhel je asi 36 stupňů

- Senzorem pro měření intenzity okolního světla
  - tento senzor měří intenzitu světla v okolí a podle naměřené hodnoty upravuje rychlost snímání prvního senzoru. Ve světlém prostředí je rychlost snímání vyšší než v tmavém prostředí

Radio turret



Obrázek 6 - Radio Turret

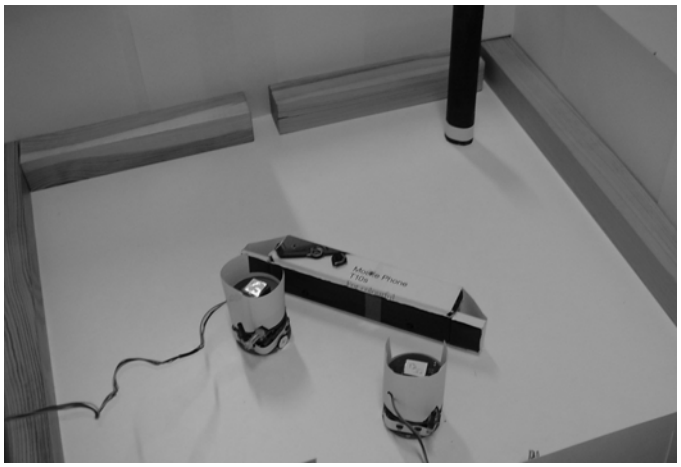
Tento modul umožňuje komunikaci s dalšími roboty vybavenými tímto zařízením a/nebo komunikaci s rádiovou základnou (Radio Base) připojenou k řídícímu počítači [14],[15]. Radiová komunikace probíhá na frekvencích 418 MHz nebo 433,920 MHz. Maximální vzdálenost pro spolehlivou komunikaci je asi 10 metrů. Kvalita přenosu a pravděpodobnost ztráty dat závisí především na vzdálenosti od kovových předmětů a dalších rádiových zařízení.

4 Box-Pushing – návrh řešení

Celou úlohu Box-Pushing je vhodné (i nutné) dekomponovat do většího počtu jednodušších úloh, které je snazší implementovat. Tyto podúlohy je nejen snazší naprogramovat, ale hlavně otestovat jejich správnost a funkčnost. Racionalitu chování robotů při řešení Box-Pushing dosáhneme správnou kombinací a vhodnou kompozicí těchto dílčích úloh. Některé tyto podúlohy budeme muset spouštět sekvenčně ve správném pořadí, jiné paralelně spolu s ostatními.

4.1 Popis prostředí pro experiment

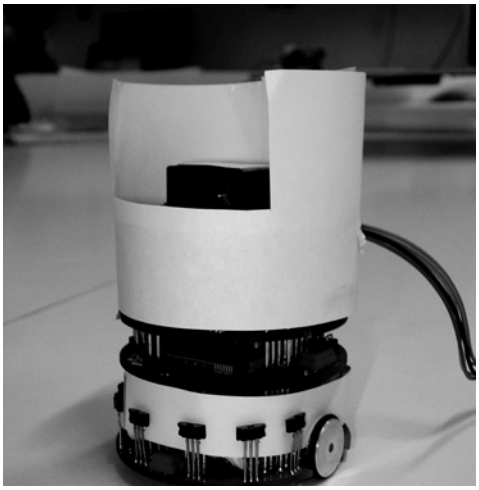
Prostředí robotů pro řešení je prostor čtvercového tvaru rozměrů asi 120x120 cm. Je ohraničeno dřevěnými hranolky výšky asi 7 cm. Nad těmito hranolky jsou dále vytvořeny stěny z bílých kartonů. Cíl pro dotlačení boxu je vyznačen v prostředí pomocí černého papírového válečku (viz Obrázek 7). Tento černý váleček je v popředí bílých stěn dobře identifikovatelný pomocí lineární kamery robotu.



Obrázek 7 - Prostředí pro experiment

Roboty Khepera, které budeme při experimentu používat, budou vybaveny Radio Turrety a lineárními kamerami K213. Nad základním modulem robotů bude nasazen bílý váleček, který použijeme proto, aby kamera jednoho robotu nezachycovala a chybně nevyhodnocovala jako cíl kameru druhého robotu, která je vyrobena

z materiálu černé barvy. V tomto papírovém válečku je vystřižen průhled pro kameru (viz Obrázek 8).



Obrázek 8 - Robot Khepera upravený pro experiment

Box, který má být v prostředí přesouván, je tvaru kvádru o rozměrech 20,5 × 4,5 × 4,0 cm.

4.2 Dekompozice úlohy

Jak jsme uvedli výše, úlohu je vhodné dokomponovat na jednodušší podúlohy. Hlavní dekompozici úlohy Box-Pushing na nejnižší úrovni můžeme provést například takto:

- Nalezení (lokalizace) boxu v prostředí
- Ustavení robotů u boxu
- Detekce cíle
- Natočení boxu směrem k cíli
- Tlačení boxu směrem k cíli

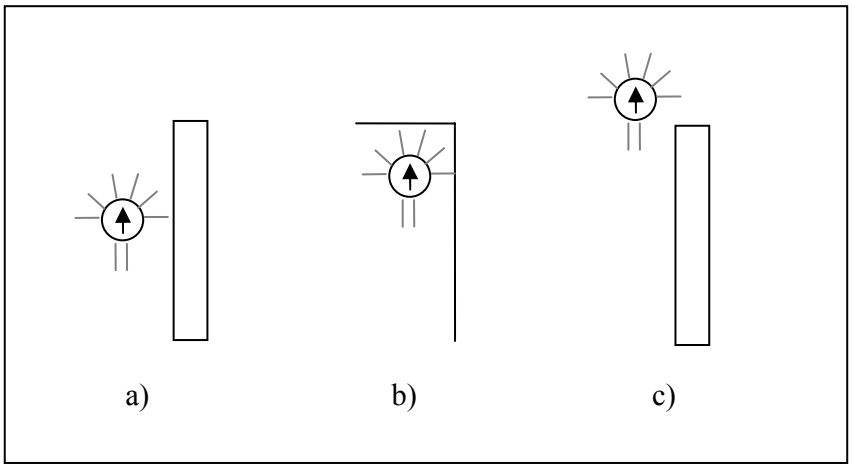
4.2.1 Nalezení boxu

Při analýze možných řešení úlohy jsme navrhli dva různé způsoby nalezení boxu v prostředí.

Varianta 1

První variantou je nalezení boxu pouze s využitím hodnot vzdáleností z IR senzorů[1] robotu. Pomocí těchto senzorů můžeme zjistit, zda se v jejich dosahu nachází nějaký předmět (překážka, stěna, box). V případě, že by se robot pohyboval v prostředí např. přímočaře a vzdálenostní senzory by zaznamenaly nějakou překážku, bychom museli rozlišit, zda se jedná o stěnu ohraničující prostředí nebo předmět, který se má tlačit. Toto rozlišení by bylo možné provést způsobem: „všechno, co někde končí, je box“. Tento postup můžeme popsat takto :

1. Pohybuj se přímočaře nějakou konstantní rychlostí a pokud je vzdálenostními senzory identifikován nějaký předmět, tak pokračuj krokem 2.
2. Pohybuj se podél předmětu (udržuj konstantní vzdálenost od předmětu) a v určitých intervalech čti hodnoty vzdáleností ze senzorů.
3. Mohou nastat tyto tři případy (viz Obrázek 9):



Obrázek 9 - Rozlišení boxu a stěny

[1] Každý robot Khepera je vybaven 8 infračervenými (IR) senzory umístěnými po obvodu robota. Více informací o IR senzorech je uvedeno v kapitole 3.

Pro vysvětlení těchto tří případů budeme používat toto označení:

- S…identifikace předmětu senzory po stranách robotu. S=1 znamená, že postranní senzory zachytily nějaký předmět, S=0 značí, že v jejich dosahu se žádný předmět nenachází.
- P…identifikace předmětu předními senzory ve směru jízdy robotu. P=1 znamená, že senzory identifikují nějaký předmět, P=0 značí, že v jejich dosahu se nenachází žádný předmět.

- Případ a (viz Obrázek 9a)
  S=1, P=0
  - v této situaci nemůžeme rozhodnout zda se jedná o stěnu nebo box a proto pokračujeme v pohybu kolem stěny

- Případ b (viz Obrázek 9b)
  S=1, P=1
  - objekt zcela jistě není box, dostali jsme se do rohu dvou stěn

- Případ c (viz Obrázek 9c)
  S=0, P=0
  - objekt, kolem kterého se robot pohyboval, je s největší pravděpodobností box

Varianta 2

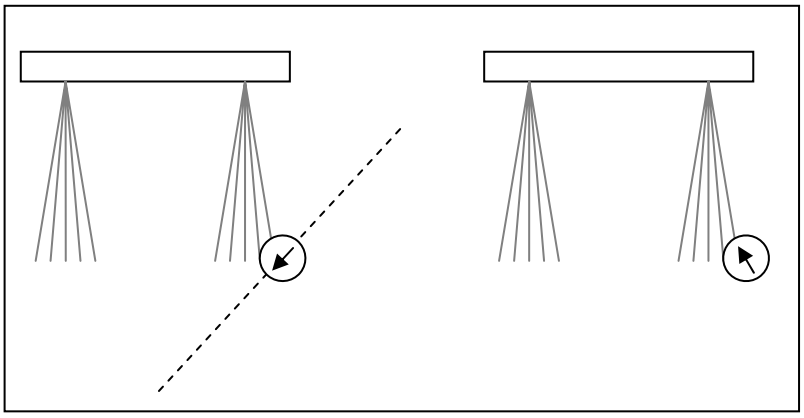
Další možností rozlišení boxu v prostředí (kterou jsme zvolili i pro implementaci v další části práce) je opatřit box dvěmi blikajícími IR diodami umístěnými ve výšce odpovídající úrovni IR senzorů robotu. Robot se v tomto případě pohybuje v prostředí a zároveň vyhodnocuje hodnoty vzdáleností od objektů a hodnoty okolního světla. Pro pohyb v prostředí je použit Braitenbergův algoritmus pracující s hodnotami vzdáleností. Pomocí tohoto algoritmu je zajištěno vyhýbání se překážkám a předcházení kolizím v prostředí. Zároveň jsou vyhodnocovány hodnoty okolního světla. V případě, že robot svými senzory zachytí IR světlo vysílané diodami v boxu, je Braitenbergův algoritmus pozastaven a robot se pohybuje směrem k boxu, přičemž jeho pohyb je řízen pouze za pomoci hodnot zachyceného IR světla.

Použité IR diody blikají s frekvencí 2 Hz (500ms rožnuto, 500ms zhasnuto). Intenzita je nastavena tak, že dosah paprsku IR světla je asi 10-15 cm od boxu. Diody jsme zvolili blikající, protože při pohybu v prostředí potřebujeme znát i hodnoty vzdáleností od překážek a stěn. Pokud by diody svítily nepřetržitě, jejich světlo by rušilo a ovlivňovalo hodnoty snímané vzdálenostními senzory. Měření vzdáleností probíhá tak, že se vyhodnocuje intenzita paprsku, který se po vyslání ze senzoru odrazí od předmětu. Pokud diody blikají, potom v okamžiku když jsou zhasnuty můžeme senzory v aktivním módu měřit vzdálenosti, a pokud IR dioda svítí, můžeme měřit intenzitu vyzařovaného světla.

Popis zapojení a konstrukce obvodu zajištujícího blikání IR diod je popsáno v Příloze A.

Nalezení boxu v prostředí můžeme v tomto případě popsat takto:

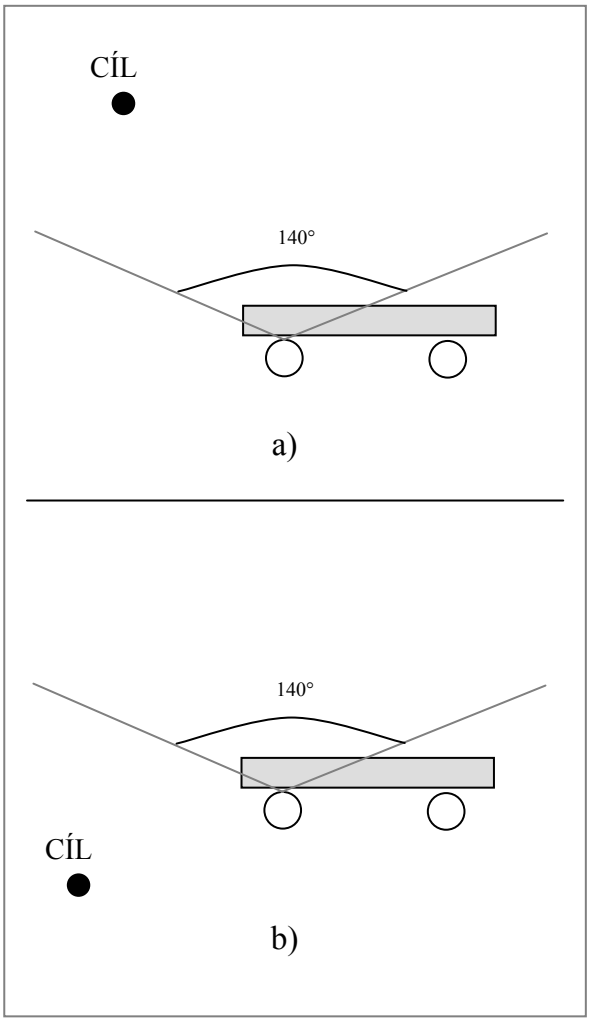
1. Pohybuj se v prostředí a vyhýbej se překážkám pomocí Braitenbergova algoritmu dokud senzory nezachytí IR světlo vysílané diodami z boxu.
2. Po zachycení IR světla zastav a ukonči Braitenbergův algoritmus.
3. Pohybuj se směrem ke zdroji světla.



Obrázek 10 - Lokalizace boxu pomocí IR diod

4.2.2 Natočení boxu směrem k cíli

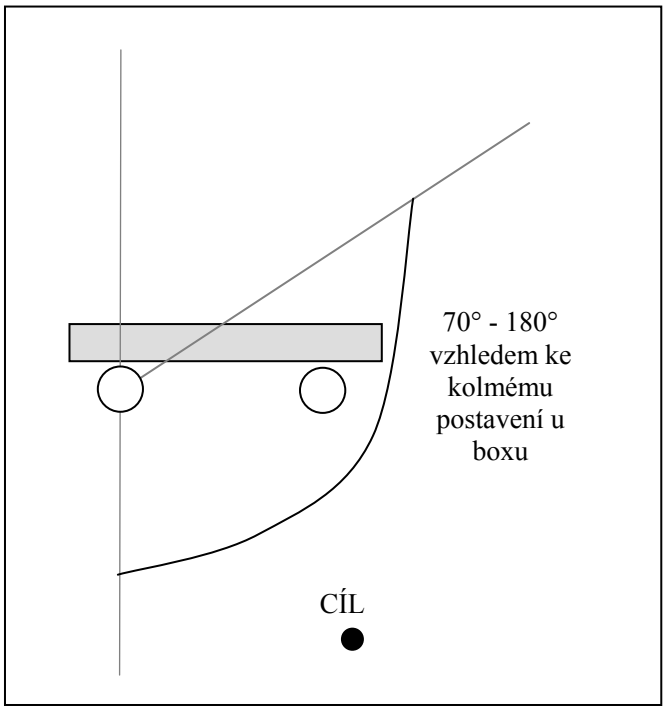
Po nalezení boxu a ustavení prvního robotu u boxu musíme zajistit správné natočení boxu směrem k cíli. Správným natočením rozumíme takové natočení boxu, při kterém je možné začít s tlačením boxu k cíli. To znamená, že roboty, které jsou ustaveny před IR diodami v boxu, mají cíl před sebou v rozmezí -70° až 70° od kolmého ustavení u boxu. Příklad správného a nevhodného natočení boxu ilustruje následující obrázek (viz Obrázek 11).



Obrázek 11 - Příklad správného (a) a špatného natočení boxu (b)

Pokud je box ustaven nesprávně, je nutné před započetím další fáze box přemístit a otočit do správné pozice. Toto natočení provedeme tlačením jednoho

z konců boxu. Který konec musíme tlačit, abychom dosáhli správného natočení, určíme takto: pokud se robot musí z kolmého ustavení u boxu k tomu, aby zachytil kamerou cíl, otočit o úhel větší než 70° a menší než 180° musíme tlačit levý konec (viz Obrázek 12).

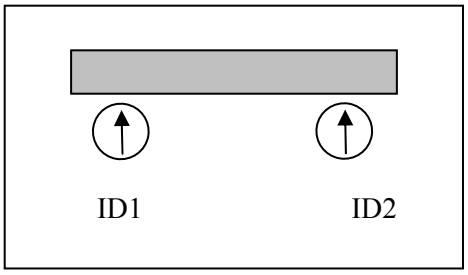


Obrázek 12. - Příklad nesprávného natočení boxu + nutnost tlačení levého konce boxu

V opačném případě musíme tlačit pravý konec. Způsob, kterým zjistíme, zda dioda kterou robot zachytil a před níž stojí je pravá či levá dioda boxu popíšeme níže v kapitole 5.5.

4.2.3 Tlačení boxu směrem k cíli

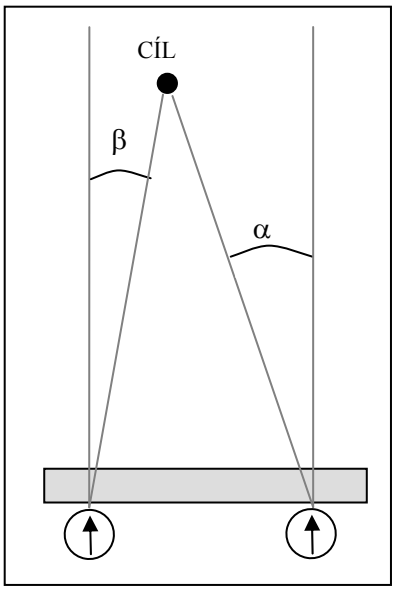
Po dokončení předchozích fází - nalezení boxu, natočení boxu a ustavení robotů u boxu - můžeme přistoupit k fázi tlačení boxu. Cíl je v prostředí vyznačen černým válečkem, který je v okolí ohraničeném bílými stěnami dobře identifikovatelný pomocí lineární kamery. Roboty jsou v tomto okamžiku ustaveny kolmo k boxu v místech označených IR diodami (viz Obrázek 13).



Obrázek 13 - Ustavení robotů u boxu

Proces tlačení boxu je tvořen posloupností kroků:

1. Robot ID1 se postupně otáčí v rozsahu 60° vlevo až 60° vpravo z kolmého postavení u boxu a snímá hodnoty z lineární kamery. V případě zachycení cíle si uloží velikost úhlu α, o který se otočil než zachytil cíl.
2. Robot ID1 zašle zprávu robotu ID2. Robot ID2 provede krok číslo 1. a zašle zpět robotu ID1 hodnotu úhlu β, o který se otočil než zachytil cíl.
3. Robot ID1 porovná velikosti úhlů α a β.
4. V případě většího úhlu α se robot ID1 posune dopředu o konstantní vzdálenost a tím provede tlačení. V případě většího úhlu β robot ID1 zašle zprávu robotu ID2, aby provedl posun a tlačení.
5. Pokračuje se opět krokem 1.



Obrázek 14 - Tlačení boxu – měření úhlů

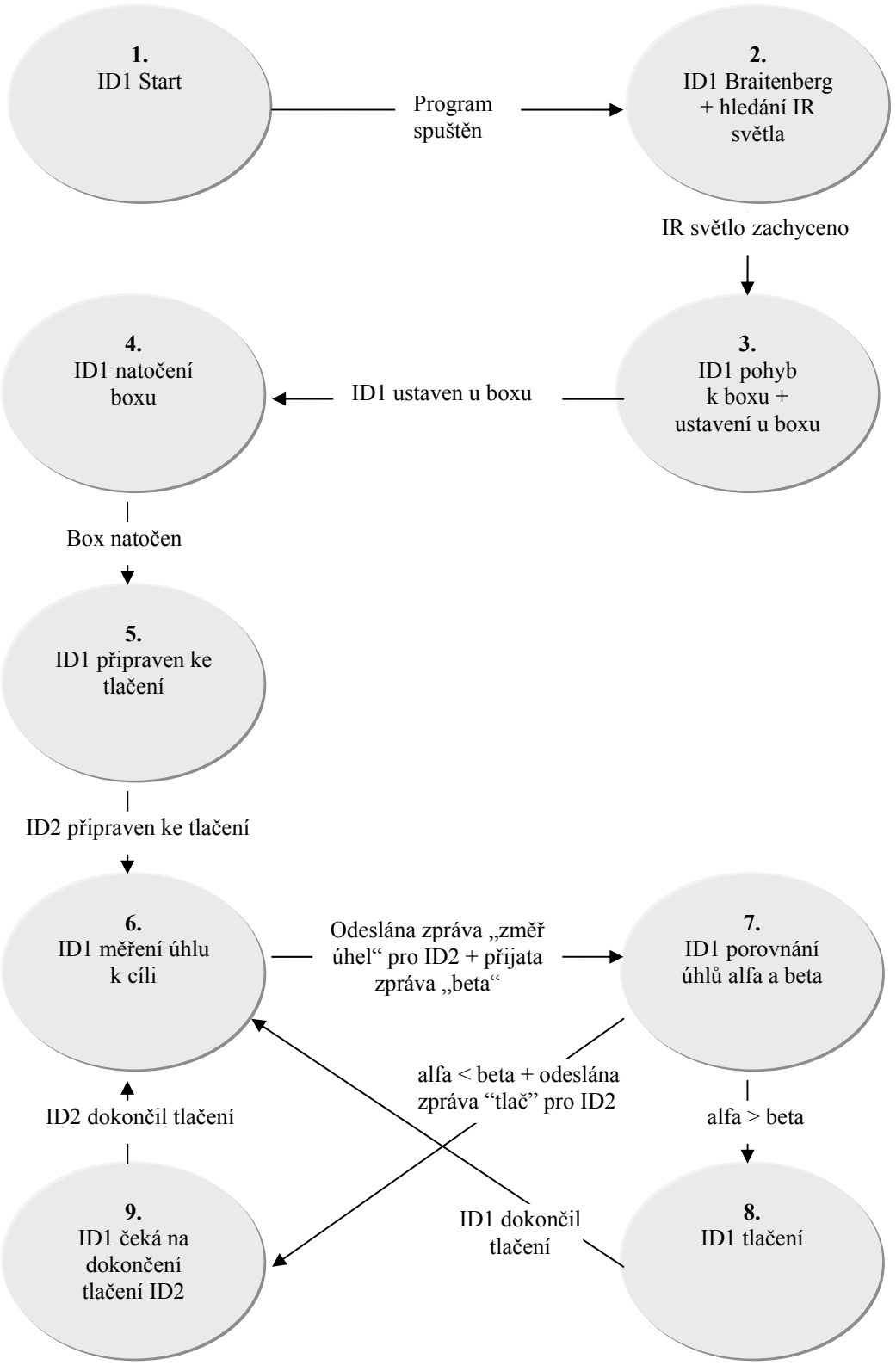
Tímto způsobem provádíme tlačení k boxu směrem k cíli za pomoci komunikace robotů.

4.3 Diagram stavů robotů při řešení

Každý robot se při řešení nachází v nějakém stavu určeném hodnotami proměnných. V tomto stavu robot zůstává a provádí akce v rámci tohoto stavu, dokud změna proměnných nebo nějaký jiný podnět, popř. dosažení podcíle tohoto kroku, neiniciuje přechod do jiného stavu. Následující diagramy zobrazují jednotlivé stavy robotů a přechody mezi nimi.

Rozdělení činností do stavů přímo neodpovídá jednotlivým implementovaným procesům řídících činnost obou robotů. Toto rozdělení do stavů má pouze ilustrovat logickou posloupnost kroků, které je nutné uskutečnit a popisuje mechanizmus a účel zasílání zpráv.

Názvy zasílaných zpráv uvádíme zapsané v uvozovkách, aby bylo zřejmé, že se jedná o zprávy. Tato označení zpráv jsou opět jen ilustrativní a proto jsme je volili tak, aby co nejvíce odrážela účel zprávy.



Obrázek 15 - Diagram stavů řešení robotu ID1

Popis jednotlivých stavů diagramu – (viz Obrázek 15):

- **1. ID1 Start**

- **2. ID1 Braitenberg + hledání IR světla**

  Robot ID1 se po nahrání a spuštění programu začíná pohybovat v prostředí a pomocí Braitenbergova algoritmu se vyhýbá překážkám. V pravidelných intervalech měří intenzitu okolního světla zachyceného IR senzory. V případě zachycení IR světla přechází do stavu 3.

- **3. ID1 pohyb k boxu + ustavení u boxu**

  Braitenbergův algoritmus je zastaven a robot se pohybuje k boxu pouze pomocí hodnot senzorů. Je důležité, aby se robot zastavil těsně před boxem a v pozici co nejvíce kolmo „čelem“ k boxu.

- **4. ID1 natočení boxu**

  Natočení boxu do správné polohy vhodné k započetí tlačení směrem k cíli.

- **5. ID1 připraven k tlačení**

  V tomto stavu je robot ID1 připraven na tlačení a posílá zprávu „start“ robotu ID2. Robot ID1 setrvává v tomto stavu dokud také robot ID2 není připraven na tlačení, tzn. dokud robot ID2 také není ustaven u boxu.

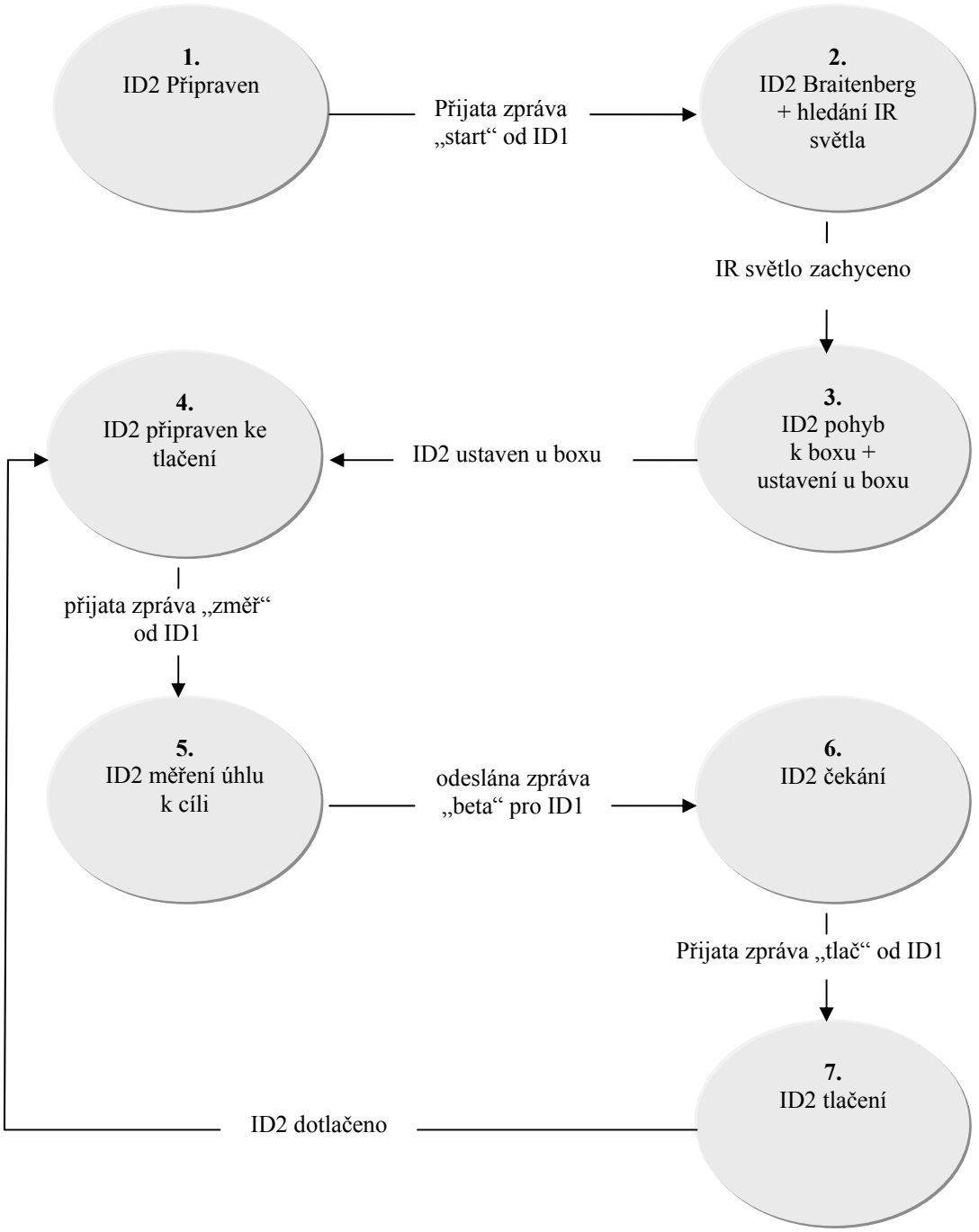
- **6. ID1 měření úhlu k cíli**

  Robot měří úhel otočení, které je potřebné k zachycení cíle pomocí kamery. Po změření úhlu zašle zprávu „změř“ robotu ID2 a čeká až také robot ID2 změří potřebný úhel β a pošle ho zpět robotu ID1.

- **7. ID1 porovnání úhlů alfa a beta**

  ID1 jako řídící robot porovná úhly α, β a podle jejich hodnot určí, který robot by měl tlačit. Pokud je jako výhodnější vyhodnoceno tlačení robotem ID2, robot ID1 mu odešle zprávu „tlač“ a čeká až robot ID2 dotlačí box o určenou

vzdálenost a ukončení tlačení potvrdí zasláním zprávy. Pokud tlačí ID1 pak po dotlačení přechází zpět do stavu 6.



Obrázek 16. - Diagram stavů řešení robotu ID2

5 Implementace a programování řešení dílčích podúloh

Programování robotů (a ne jen robotů Khepera) se zásadně liší od „klasického“ programování aplikací. Máme sice k dispozici klasický procesor umístěný v robotu, který ve všem připomíná normální procesor, většinou můžeme používat jazyky, které používáme k programování aplikací, ale styl a paradigma programování se zásadně liší.

Mezi základní problémy programování robotů Khepera patří testování a ladění programů. Tyto problémy spolu velice úzce souvisí. V případě klasického programování aplikací máme možnost naprogramovanou část kódu libovolně testovat a upravovat. Po napsání programu určíme testovací data a na nich testujeme námi vytvořené části programu. Tato data vybíráme tak, aby co nejvíce pokrývala množinu hodnot, které může náš program obdržet na vstupu a testujeme správnost výstupních hodnot. V případě, že neobdržíme správný výsledek, program upravíme a odladíme a můžeme spustit test se stejnými testovacími daty.

Při programování robotů, které se pohybují v reálném (a tím i proměnlivém) prostředí, je testování aplikací poněkud složitějším problémem. Toto prostředí se neustále mění a je složité zachytit stavy všech objektů, které prostředí tvoří a s tím související vstupy našeho programu. Například vstupní hodnoty IR senzorů robotu nebo kamery jsou ovlivňovány mnoha skutečnostmi, kterými mohou být například intenzita okolního světla, sluneční světlo, stíny atd..

To znamená, že nemáme možnost otestovat napsaný program dvakrát se stejnými vstupy. Pracujeme s daty, která se neustále v čase mění a proto musíme náš program navrhnout tak, aby byl dosti robustní a dokázal reagovat na velké množství vstupů. Proto i na výstupu musíme počítat s různými, někdy i nepřesnými a zkreslenými, hodnotami. I po velkém množství testů nemůžeme zaručit správnou funkčnost našich aplikací.

Dalším problémem je ladění programů, které s tímto souvisí. Aplikaci naprogramujeme a přeneseme do paměti robotu, který pak už v prostředí funguje autonomně. To znamená, že nemůžeme používat standardní postupy při ladění programu, jako jsou krokování nebo použití sledování proměnných (watches). Při běhu programu a činnosti robotu v prostředí prakticky nemáme možnost kontrolovat stav provádění programu a hodnoty proměnných, na kterých závisí správná funkce robotu.

Při výskytu neočekávaného chování nemůžeme identifikovat jaké skutečnosti k tomuto chování vedly. Robot se pro nás v tomto případě stává černou skříňkou. Jedinou možností, jak alespoň částečně můžeme sledovat stav programu, je připojení robotu k počítači přes sériové rozhraní. V tomto případě můžeme sledovat výpisy programu přes terminál. Samozřejmě i v tomto případě máme k dispozici pouze hodnoty, jejichž výpis si vynutíme explicitně v kritických okamžicích vykonávání programu.

Vzhledem k těmto skutečnostem se při programování snažíme navrhovat co nejjednodušší postupy řešení a omezit složité výpočty s velkým množstvím hodnot ze senzorů.

Více informací o možnostech a způsobech programování robotů Khepera můžeme najít v [16].

V této kapitole postupně popíšeme skutečnosti související se samotnou implementací řešení úlohy a programování jednotlivých dílčích funkcí. Nejprve popíšeme všechny hlavní funkce, které byly použity a poté návrh a implementaci procesů. Funkce budeme popisovat v ucelených blocích, zahrnujících funkce, které spolu souvisejí nebo realizují podobné činnosti. Popíšeme také hlavní funkce pro práci s rozšiřujícími moduly K213 Vision Turret a Radio Turret.

5.1 Programování - základní modul Khepera

Základní funkce, které budeme používat při práci se základním modulem robotu Khepera, jsou funkce pro pohyb robotu a čtení informací ze senzorů.

5.1.1 Pohyb robotů

Kola robotů jsou poháněna dvěma motorky. Ke každému motorku je připojeno jedno kolo. Na osách obou kol jsou umístěny přírůstkové enkodéry (incremental encoders), které generují impulsy během otáčení kola. Pomocí těchto enkodérů můžeme jednoduše měřit počet otočení kola a tím i ujetou vzdálenost. Těchto pulsů je generováno 12 za každý milimetr pohybu (tzn. 1 impuls = 0.08 mm) [11]. K těmto enkodérům má přístup přímo procesor robotu a může číst hodnoty počtu impulsů.

Pomocí těchto hodnot můžeme měřit ujetou vzdálenost popř. přesně nastavit o kolik se jednotlivé motorky mají otočit.

Bios robotů poskytuje funkce pro čtení popř. nastavení hodnot těchto enkodérů [17],[18]:

- *mot_get_position(motorNb)*
  - vrací aktuální absolutní hodnotu (pozici) motorku číslo motNb

- *mot_new_position_2m(position1, position0)*
  - nastaví novou (cílovou) hodnotu enkodéru obou motorků

Při programování složitějších aplikací, ve kterých chceme mít úplnou a přesnou kontrolu nad pohybem robotu, je vhodné implementovat funkce, které nám umožní např. pohyb vpřed o určitý počet milimetrů nebo otočení robotu o určitý úhel. Funkce, které jsme takto naprogramovali, jsou funkce Forward a funkce TurnLeft.

Pohyb vpřed o určenou vzdálenost

Funkce Forward využívá možnosti číst a nastavovat hodnoty enkodérů. Jak jsme uvedli výše, jedna jednotka enkodéru odpovídá asi 0.08 mm pohybu kolečka. Po výpočtu kolik jednotek enkodéru odpovídá požadované vzdálenosti jsou nastaveny parametry pro výpočet optimálních vlastností motorků jako jsou akcelerace a další. Tyto parametry jsou doporučovány přímo výrobcem a ve většině aplikací není potřeba je jakkoliv měnit. Podrobnější informace o funkci a nastavení motorků můžeme najít v [11]. Po přečtení aktuálních hodnot enkodérů je k těmto hodnotám přičtena hodnota odpovídající požadované vzdálenosti a výsledek je funkcí mot_new_position_2m nastaven jako požadovaná cílová hodnota.

```
int Forward(int number)
{
int32 pos0,pos1;
float units = (float)number / 0.08;

mot_config_speed_1m (0, 3500, 800, 100);
mot_config_speed_1m (1, 3500, 800, 100);
mot_config_position_1m(0,3000,20,4000);
mot_config_position_1m(1,3000,20,4000);

mot_config_profil_1m(0,maxSpeed,maxAccel);
mot_config_profil_1m(1,maxSpeed,maxAccel);

pos0 = mot_get_position(0);  //ziskani PID koeficientu
pos1 = mot_get_position(1);

mot_new_position_2m(pos0+units,pos1+units);
return (int)units;
}
```

Zavoláním funkce Forward zahájí robot pohyb dopředu o požadovanou vzdálenost. Ihned po jejím zavolání, ale pokračuje vykonávání programu vyhodnocením dalšího příkazu. Tato skutečnost okamžitého pokračování vykonávání nám ale znemožní používat pro řízení pohybu robotu více volání funkce Forward za sebou. Např. při použití:

```
Forward(100);
Forward(200);
```

se okamžitě po zahájení pohybu robotu vpřed o 100mm zavolá funkce Forward podruhé a celý pohyb robotu o 100mm je přerušen a aktuální pohyb robotu je nahrazen pohybem o 200mm. Řešením takové situace je zajistit, že před dalším zavoláním funkce Forward je již ukončen pohyb z prvního volání funkce. Navrhli jsme proto toto řešení: funkci Forward voláme z další funkce Forward_confirm tak, že po jejím zavolání kontrolujeme v cyklu hodnoty enkodérů a až po tom, kdy hodnoty enkodérů odpovídají situaci, že celý pohyb vpřed proběhl, se tato funkce ukončí a umožní další řízení pohybu.

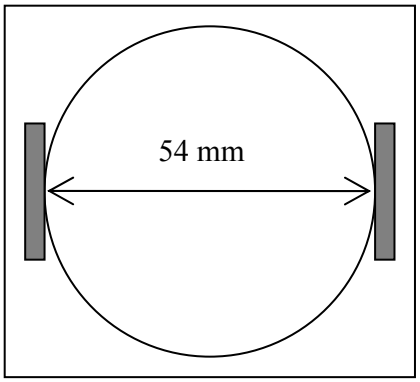
V následující části kódu funkce Forward_confirm je ukázána kontrola stavu enkodérů a pozastavení vykonávání programu, dokud není proveden celý pohyb robotu.

```
pos0 = mot_get_position(0);
pos1 = mot_get_position(1);

a = Forward(number);
if(a>=0)
 {
       do
       {
       check0 = mot_get_position(0);    ///aktualni pozice
       check1 = mot_get_position(1);
       }
       while ((check0<=pos0+a)&&(check1<=pos1+a));
}
 else
 {
       do
       {
       check0 = mot_get_position(0);    ///aktualni pozice
       check1 = mot_get_position(1);
       }
       while ((check0>=pos0+a)&&(check1>=pos1+a));
}
```

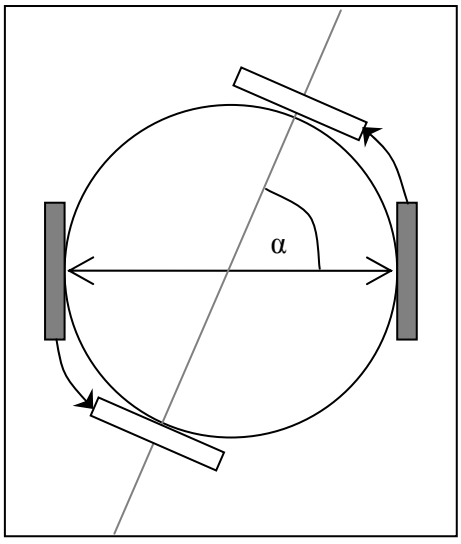
Otočení o určitý úhel

Stejně jako u implementace pohybu o určitou vzdálenost, tak i při implementaci otočení o určitý úhel budeme využívat hodnoty enkodérů. Při otočení o úhel musíme pro výpočet dráhy jednotlivých kol uvažovat i vzdálenost kol a obvod robotu.



Obrázek 17 - Průměr základního modulu Khepera

Obvod robotu je 170 mm (tzn. 2110 jednotek enkodéru). Při otočení robotu o úhel α musíme určit hodnotu otočení každého kola. Abychom docílili otočení robotu beze změny jeho polohy, musíme jedním kolem pootočit směrem dopředu a druhým kolem směrem dozadu. Při otáčení o úhel α, musíme tedy každým kolem pootočit o $(\alpha/360)\cdot 2110$ jednotek (viz Obrázek 18).



Obrázek 18 - Princip otáčení robotu na místě

Nastavení nových hodnot enkodéru je provedeno podobně jako ve funkci Forward. Podobně, jako jsme implementovali funkci Forward_confirm pro zajištění

pokračování běhu programu až po provedení celého pohybu vpřed, i u otáčení robotu použijeme další funkci TurnLeft_confirm.

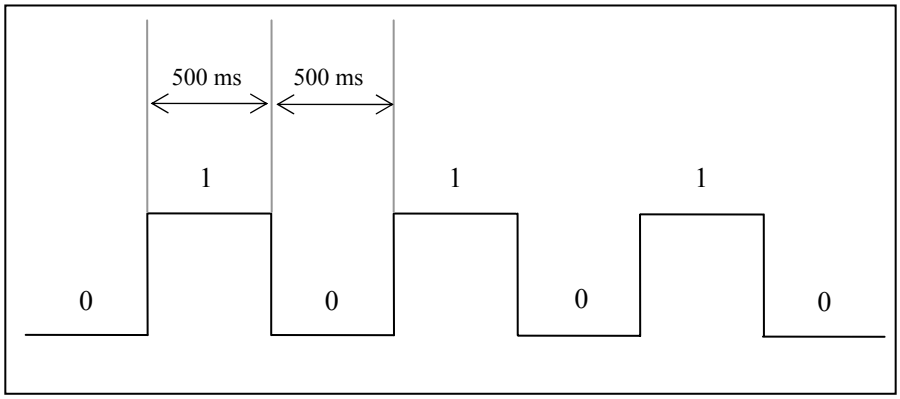
5.1.2 IR senzory

Pro pohyb v prostředí a nalezení boxu (identifikaci IR světla) budeme potřebovat funkce, které budou číst hodnoty jednotlivých senzorů. Základní funkce pro čtení těchto hodnot poskytuje přímo KhereraBIOS a jeho modul SensManager [18]. Tyto funkce jsou:

- sens_get_reflected_value(sensorNb)
  - vrací hodnotu vzdálenosti ze senzoru číslo sensorNb
- sens_get_ambient_value(sensorNb)
  - vrací hodnotu intenzity okolního světla ze senzoru číslo sensorNb

Pro zjednodušení práce budeme používat funkci Light(), která přečte hodnoty okolního světla ze všech senzorů a vyhodnotí, zda nějaký senzor zachytil světlo pod určenou hranicí. Funkce vrací 1 pokud alespoň jeden senzor zachytil IR světlo, jinak 0.

Protože budeme potřebovat měřit vzdálenost robotu nejen od stěn prostředí, ale i od boxu, který je opatřen blikajícími diodami, musíme zajistit, abychom četli hodnoty vzdálenostních senzorů pouze tehdy, pokud diody nejsou rozsvíceny. Proto budeme v kódu používat kontrolu, zda je před měřením a čtením hodnot vzdáleností dioda zhasnuta.



Obrázek 19 - Průběh signálu IR diod boxu

Stav IR diody	1 - rozsvícena	0 - zhasnuta
Režim měření	Pasivní	Aktivní
Druh měření	Intenzita světla	Vzdálenost

tabulka 1 - Přehled možností měření IR senzory

Jak jsme uvedli výše, dioda je vždy 500ms rozsvícena a 500ms zhasnuta (viz Obrázek 19) a hodnoty ze senzorů jsou čteny každých 20ms. To znamená, že pokud před čtením i po čtení hodnot je dioda zhasnuta, jistě dostaneme správné a neovlivněné hodnoty vzdálenosti.

Tuto kontrolu můžeme v kódu provádět např. takto:

```
do
{
      light_before=Light();
      for (i = 0; i < 8; i++)
            {
            sensBufDist[i] = sensor->oProximitySensor[i];
            str_cnvt_bin32_dascii (strout, sensBufDist, 8);
            }
      light_after=Light();
}while((light_before!=0)&&(light_after!=0));
```

5.2 Programování – rozšiřující moduly

Při programování úlohy budeme v mnoha částech řešení potřebovat také funkce pro obsluhu a získávání informací pomocí rozšiřujících modulů. Většina těchto funkcí je podporována přímo KheperaBIOSem [18]. Na tomto místě uvedeme nejdůležitější z těchto funkcí.

5.2.1 Radio Turret

Tento rozšiřující modul je připojen přímo na systémovou sběrnici robotu Khepera. Všechny funkce potřebné pro kódování, přenos a příjem zpráv, detekci a opravu chyb při přenosu jsou prováděny procesorem implementovaným přímo v tomto modulu [14]. Pro komunikaci mezi roboty je nutné, aby každý Radio Turret měl nastaveno svoje jedinečné ID. Toto nastavení je nutné provést nastavením switchů přímo na modulu.

Khepera BIOS podporuje několik funkcí pro obsluhu a práci s Radio Turretem. Mezi nejvýznamnější a nejpoužívanější funkce patří:

- *int32 radio_reset(void);*

- funkce provede inicializaci a zresetování Radio Turretu a všech proměnných.
- funkci je nutné v programu zavolat ještě před použitím jakékoliv jiné funkce pro obsluhu rádia, nejlépe ještě před spuštěním procesů

- *int32 radio_getIDNumber(void);*

- funkce vrací aktuální ID Radio Turretu. Správné nastavení ID je vhodné vždy zkontrolovat. Zároveň je možné zavoláním této funkce ověřit správné připojení a funkčnost Radio Turretu

Přenášené zprávy mezi jednotlivými roboty jsou reprezentovány pomocí vektoru v tomto tvaru:

- *uint8 zprava[] = {ID adresáta zprávy, velikost zprávy,zpráva…};*

- např. `zprava[]={2,5,2,3,1,4,1}` - zpráva bude odeslána robotu s ID 2, velikost zprávy je 5 bytů a zpráva je (2,3,1,4,1)

- *int32 radio_sndBuffer(uint8 *buffer);*

- funkce provede odeslání zprávy a vrací hodnotu indikující úspěšnost odeslání

Příklad:

```
uint8 zprava[] =  {2, 5, 0, 1, 2, 3, 4};
status = radio_sndBuffer (zprava);
  if (status < 0)
        printf ("Zprávu se nepodařilo odeslat");
else
        printf ("Zpráva byla úspěšně odeslána ");
```

- *int32 radio_recBuffer(uint8 *buffer);*

- funkce pro příjem zpráv od ostatních robotů. Přijme pouze zprávy, jejichž ID adresáta je shodné s vlastním ID Radio Turretu.
- formát přijaté zprávy je: buffer[]={ID odesílatele, velikost zprávy, zpráva,….}

- *int32 radio_getStatus(void);*

- vrací stav komunikačního kanálu Radio Turretu.
    - pokud (radio_getStatus()&0x2)==1 potom v bufferu je příchozí zpráva
    - pokud (radio_getStatus()&0x1)==1 potom v bufferu je ještě neodeslaná zpráva
- tuto funkci je vhodné používat především pro zjišťování příchodu zprávy např. v nějakém cyklu
- z vlastní zkušenosti víme, že je nutné tuto funkci zavolat dvakrát za sebou. Při prvním pokusu o výběr bufferu vrací nejspíš náhodnou hodnotu!

5.2.2 K213 Vision Turret

K získávání obrazové informace pomocí rozšiřujícího modulu K213 budeme potřebovat několik funkcí. Základní funkce podporované KheperaBIOSem [18] jsou:

- *int32 k213_getVersion ()*
  - vrací version a revision softwaru K213 Vision Turretu

- *int 32 k213_get8BitImage (image);*

- fukce uloží do vektoru image 64 8-bitových hodnot stupňů šedi načtených z kamery
- *image = (uint8 *) malloc (64 * sizeof (uint8))*
- vrací hodnotu indikující správnost provedení

- *int 32 k213_get4BitImage (image)*
  - stejně jako funkce k213_get8BitImage, ale vrací 4-bitové hodnoty

5.3 Komunikace robotů

Jak jsme uvedli výše, komunikaci a zasílání zpráv mezi roboty zajišťuje rozšiřující modul Radio Turret. Oba roboty jsou vybaveny tímto modulem a pomocí přepínačů na tomto modulu je každému robotu přiřazeno jeho identifikační číslo pro komunikaci (ID). Řídícímu robotu je nastaveno ID1 a druhému robotu ID2.

Navrhli jsme mechanismus zasílání zpráv a jejich potvrzování. Potvrzování zpráv je důležité z důvodu nespolehlivosti zasílání a přijímání zpráv jednotlivými roboty. Jednání robotů je přímo ovlivňováno zasílanými zprávami a v případě ztracení nebo nepřijetí zprávy by došlo k zastavení provádění celého řešení.

Zasílání a potvrzování zpráv je zajišťováno jedním procesem zpráv běžícím v paměti obou robotů. Tento proces periodicky spouští funkci pro výběr zprávy z messagebufferu a funkci pro odeslání zprávy z messagebufferu. Pokud nějaká funkce nebo proces požaduje zaslání zprávy druhému robotu, musí zavolat speciální funkci, která naplní messagebuffer požadovanými daty a nastaví příznak, že se má začít odesílat. Pokud funkce pro odesílání, spuštěná z procesu zpráv, zjistí nastavení tohoto příznaku, provede odeslání zprávy podle zadaných dat v messagebufferu.

V případě příchozí zprávy je tato zpráva zpracována a podle čísla zprávy jsou nastaveny globální proměnné ovlivňující činnosti robotu (spuštění dalších procesů, větvení programu atd.). Činnost odesílajícího robotu je většinou pozastavena až do doby, než je jím přijata zpráva potvrzující přijetí odeslané zprávy druhým robotem. Tento proces tedy realizuje veškerou komunikaci mezi roboty - zajišťuje vyhodnocování zpráv a reakce na ně pomocí nastavování globálních proměnných.

5.3.1 Odeslání zprávy

Celý proces zpracování zprávy, od požadavku nějakého procesu nebo funkce na její odeslání, až po její skutečné odeslání, můžeme popsat posloupností těchto kroků:

1. požadavek na odeslání zprávy – volání funkce Send_Main
   - tato funkce nastaví hodnoty v messagebufferu, jako jsou velikost zprávy, příjemce zprávy, číslo zprávy, data (obsah) zprávy a nastaví globální proměnnou send_request na *true*.
2. detekce požadavku na odeslání zprávy v procesu zpráv
   - funkce function_send spuštěná z procesu zpráv periodicky kontroluje stav proměnné send_request. V případě nastavení této proměnné na true začíná odesílání bufferu.

Změnu hodnot messagebufferu před odesláním a nastavení proměnné send_request zajišťuje funkce Send_Main:

```
void Send_Main(int ID, int length, int zprava1, int zprava2, int
zprava3)
{
send_ID = ID;
send_length = length;
send_buffer[0]=ID;
send_buffer[1]=length;
send_buffer[2]=zprava1;
send_buffer[3]=zprava2;
send_buffer[4]=zprava3;
send_request = 1;
}
```

Funkce zajišťující odeslání messagebufferu z procesu zpráv je funkce function_send:

```
void function_send ()
{
  int32 status;
 if(send_request==1)
 {
  status = radio_sndBuffer (send_buffer);
  tim_suspend_task(10);
  status = radio_sndBuffer (send_buffer);
  if (status < 0)
    {
    printf ("Function_send  Network Communication Problem
    %ld\r\n", status);
    }
    else
       {
       printf ("Function_send Message sent \r\n");
       send_request=0;
       }
 }
}
```

V případě potvrzované zprávy zůstává proměnná send_request stále nastavena na *true* a zpráva se odesílá stále znovu až do přijetí potvrzovací zprávy od adresáta.

5.3.2 Přijetí zprávy

Proces zpráv opakovaně spouští i funkci pro výběr zprávy z messagebufferu. Pokud po kontrole obsahu tohoto bufferu je zjištěna přítomnost zprávy, je tato zpráva okamžitě vyhodnocena a zpracována. Zpracování zprávy a reakce na její přijetí jsou závislé na čísle zprávy a jejím obsahu. Po vybrání zprávy z bufferu jsou podle čísla zprávy provedeny odpovídající akce, jako např. okamžitý požadavek na odeslání potvrzující zprávy zpět k odesílateli nebo nastavení globálních proměnných programu.

Tímto způsobem jsou například odblokovány ostatní procesy a umožněno pokračování v provádění části programu.

Proces přijetí zprávy:

1. proces zpráv spouští funkci function_receive
   - při detekci zprávy v bufferu je tato zpráva analyzována a jsou nastaveny proměnné obsahující ID odesílatele, délku zprávy a obsah zprávy
2. vyhodnocení zprávy
   - podle čísla zprávy jsou nastaveny globální proměnné ovlivňující činnosti robotu (spuštění dalších procesů, větvení programu atd.)

Při přijetí zprávy, která si žádá potvrzení, je odeslána další zpráva, která u odesílatele zprávy po jejím vyhodnocení nastaví hodnotu send_request na *false* a tím zamezí dalšímu odesílání zprávy. Tímto je zpráva potvrzena a odesílatel může pokračovat v provádění programu.

5.4 Funkce pro práci s obrazem

Pro zjištění, zda se v zorném poli kamery K213 nebo v okolí robotu nachází nějaký objekt zachytitelný kamerou a případně kterým směrem objekt leží, budeme používat následující naimplementované funkce.

- int Image_k213()
  - funkce vrací počet prvků ve vektoru získaném funkcí k213_get8BitImage, které jsou pod nastavenou hranicí pro černou barvu, tzn. „velikost“ černé plochy zachycené kamerou

```
int Image_k213()
{
 int32 status;
 uint8 *image;
 uint32 i;
 int number;
 number =0;
  image = (uint8 *) malloc (64 * sizeof (uint8));
  if (image == 0)
    {
      printf ("Memory problem K213\r\n");
      return;
    }
   status = k213_get8BitImage (image);
   if (status < 0)
     {
       printf ("Image_k213 Network Communication Problem\r\n");
     }
   else
     {
 for(i=0;i<64;i++)
       {
       if(image[i]<160)
             number++;
       }
     }
return number;
}
```

Další funkcí související s hledáním objektů pomocí kamery je funkce Find_target_360(). Funkce provádí postupné otáčení robotu kolem svého středu a vyhodnocování sejmutého obrazu. Funkce vrací hodnotu úhlu otočení do směru, ve kterém kamera zachytí největší počet černých hodnot obrazu, tzn. největší hodnotu z funkce Image_k213. Tuto funkci použijeme například v situaci, kdy potřebujeme zjistit, zda je box správně natočen před tlačením k cíli.

```
int Find_target_360()
{
int numberpixelu1 =0;
int angle = 0;
 do
 {
 numberpixelu1 = Image_k213();
 tim_suspend_task(100);
 if(numberpixelu1 > 3)  //nalezeno
       {
       TurnLeft_confirm(angle);
       return angle;
       }
 TurnLeft_confirm(-10);
 angle  = angle + 10;
 if(angle >360) angle=0;
 }while (1);
}
```

Podobnou funkcí jako fce *Find_targetl_360()* je i funkce *Image_angle_rotation()*. Tato funkce slouží k měření úhlu k cíli v části tlačení boxu. Oproti funkci Find_target_360 se liší v rozsahu úhlu ve kterém hledá předmět. Tento rozsah je pouze -60° až 60° z výchozího postavení. Návratová hodnota je celé číslo z intervalu 0 až 11 (0 znamená -60°, 11 odpovídá úhlu 60° vpravo). V případě, že žádný objekt nebyl v tomto rozsahu nalezen, funkce vrací hodnotu 99. V případě, že je kamerou zachycen černý předmět, není ještě hned vrácena hodnota úhlu, ale je provedeno ještě měření v úhlu o 10° větším a porovnáno, jestli v tomto úhlu není zachycena větší černá plocha a tím i vrácená hodnota úhlu přesnější. Přesnost určení úhlu je velmi důležitá, proto se tímto postupem snažíme najít nejpřesnější hodnotu.

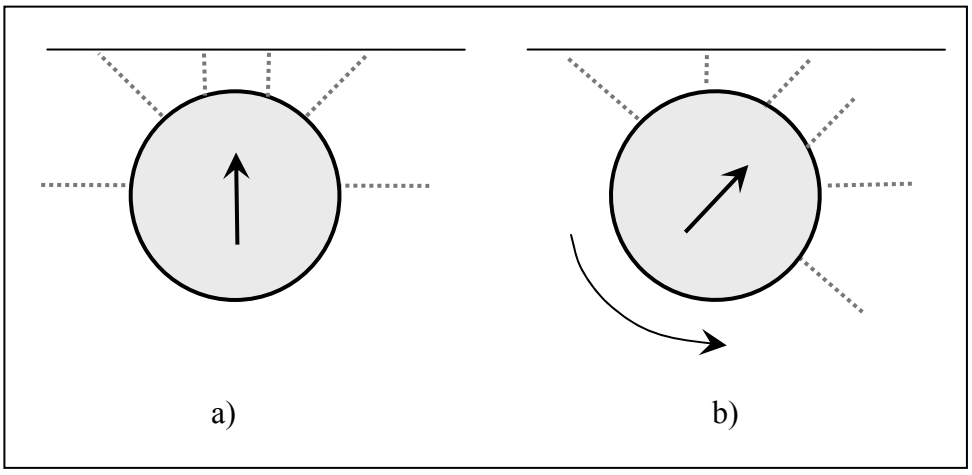
```
int Image_angle_rotation()
{
int i=0;
int numberpixelu1 = 0;
int numberpixelu2 =0;
int pocitadlo = -6;
TurnLeft_confirm(60);
for(i=0;i<12;i++)
 {
 numberpixelu1 = Image_k213();
 if(numberpixelu1>=3)
       {
       TurnLeft_confirm(-10);
       numberpixelu2 = Image_k213();
       if(numberpixelu2>=numberpixelu1)    //o jedno doprava je to
lepsi
             {
             TurnLeft_confirm((pocitadlo*10)+10);
             return (pocitadlo+6);
             }
             else
             {
             TurnLeft_confirm(10);
             TurnLeft_confirm((pocitadlo*10));
             return (pocitadlo+6);
             }
       }
 else
       {
       TurnLeft_confirm(-10);
       pocitadlo = pocitadlo + 1;
       }
 }
TurnLeft_confirm(60);
return 99; //objekt nenalezeno!!!
}
```

5.5 Další funkce

Následující funkce realizují řešení jednotlivých dílčích podúloh, jejichž kompozicí budeme dosahovat řešení celé úlohy.

Jednou z podúloh, kterou budeme muset vyřešit a poté ji budeme často používat, je ustavení robotu kolmo k překážce nebo boxu. Kolmé nastavení bude vždy výchozím ustavením pro další akce, při kterých budeme měřit vzdálenosti nebo úhly otočení, proto je důležité, aby kolmé nastavení bylo co nejpřesnější.

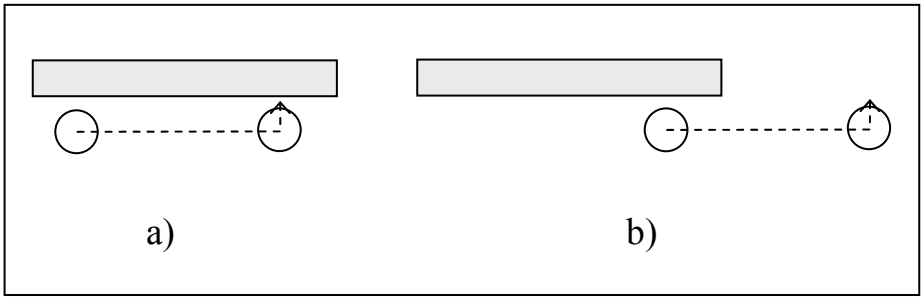
Jednou z možností, jak realizovat toto ustavení, je snímání vzdáleností ze senzorů a porovnávání součtu hodnot senzorů na levé a pravé straně robotu a podle těchto hodnot řízení kol a otáčení robotu (větší součet v levé části senzorů - otočení vlevo, a naopak). V tomto řešení musíme určit hodnotu rozdílu součtu hodnot na levé a pravé straně, při které se otáčení zastaví a robot bude ustaven kolmo. Nastavení této konstanty je velmi problematické, protože hodnoty ze senzorů závisí na materiálu a dalších vlastnostech předmětu a proto většinou nedosáhneme dobrého kolmého ustavení.



Obrázek 20 - a) Ustavení robotu kolmo k boxu
b) Otáčení robotu v závislosti na hodnotách vzdáleností

Toto řešení můžeme upravit tak, že z výchozí polohy budeme podle hodnot v levé a pravé části senzorů otáčet buď vlevo, nebo vpravo, ale při první změně směru otáčení okamžitě zastavíme.

Při řešení úlohy budeme potřebovat i určení, zda robot stojící před IR diodou u boxu, je ustaven u levé nebo pravé diody (tzn. blíže k levému nebo pravému okraji boxu). Tato informace je potom v dalším průběhu řešení velmi důležitá, kvůli rozhodování, který robot je ve výhodnější pozici pro tlačení a bude tlačit. Řešení jsme navrhli následovně: robot stojící u IR diody se otočí o 90° vpravo, posune se o 12cm vpřed a otočí se o 90° vlevo. Po provedení tohoto přesunu nám k určení, zda robot stál ve výchozí pozici na levé nebo pravé straně boxu, stačí hodnoty vzdálenostních senzorů robota. Pokud vzdálenostní senzory zachytí box, tak výchozí pozice byla vlevo, jinak robot stál u boxu vpravo.



Obrázek 21 - Výchozí ustavení robotu vlevo (a) a vpravo (b)

Po měření se robot vrátí zpět do své původní pozice.

Jednou z částí řešení, jak jsme uvedli v kapitole 4.2.2, je i nastavení boxu do správné pozice před tlačením k cíli. Funkce realizující nastavení boxu do správné polohy pro tlačení je funkce *Turn_box*.

Celý postup můžeme popsat takto:

- Pomocí funkce Find_target_360 (viz kapitola 5.4) určíme úhel otočení k cíli.
- Mohou nastat tři různé případy vzhledem k velikosti úhlu otočení z kolmého ustavení robotu u boxu směrem k cíli:
    - úhel otočení je větší než 290° nebo menší než 70° - box je postaven správně a není nutno pozici boxu měnit.

- úhel otočení je větší než 70° a menší než 180° - box je pro dosažení správné pozice nutno tlačit na jeho levé straně
- úhel otočení je větší než 180° a menší než 290° - box je pro dosažení správné pozice nutno tlačit na jeho pravé straně

- Po zjištění na které straně je nutno box tlačit, je buď hned provedeno tlačení (pokud robot stojí na správné straně boxu), nebo je nejprve nutno robot přesunout na druhý kraj boxu a provést tlačení.

Uvádíme část kódu funkce *Turn_box* zajišťující tlačení boxu na levé straně:

```
if((angle >70 )&&(angle<180))
 {
       printf ("Turn_box - Must push left\r\n");
       if(pravy==0)
                   {
                   Forward_confirm(push_distance);
                   }
             else
                   {
                   TurnLeft_confirm(-90);
                   tim_suspend_task(50);
                   Forward_confirm(-120);
                   tim_suspend_task(50);
                   TurnLeft_confirm(90);
                   tim_suspend_task(50);
                   Forward_confirm(push_distance);
                   pravy = 0;
                   }
 }
```

6 Implementace řešení BoxPushing a experimenty

V předchozích kapitolách jsme popsali dekompozici řešení úlohy do menších funkčních částí a jejich implementaci. Nyní můžeme popsat návrh řešení úlohy za použití těchto funkcí a jejich kompozici pro dosažení správného řešení.

Každou část zajišťuje jeden nebo více procesů běžících na procesoru robotu. Procesy jsou reprezentovány jako nekonečné cykly *for*. Všechny procesy jsou inicializovány a spuštěny hned na začátku při startu programu. Tyto procesy jsou sice ihned spuštěny, ale některé z nich jsou na začátku blokovány a jejich odblokování je vyvoláváno jako reakce ostatních procesů v průběhu řešení úlohy. Blokování a odblokování procesů je řešeno nastavováním globálních proměnných programu a každý proces kontroluje stav těchto proměnných. Jakmile se změní stav proměnné, příslušné k danému procesu, proces začne plnohodnotně provádět svou činnost. Stav „blokovací" proměnné může být měněn např. jako reakce na přijatou zprávu nebo po detekování nějaké neočekávané situace.

Princip blokování procesů:

```
static void process_1()
{
for (;;)  //nekonečný cyklus
        {
         if(process_1_enabled==1)   //blokovací proměnná
               {
               kód procesu
               }
        }
}
```

Při řešení úlohy používáme tyto procesy s následujícími názvy[1]:

- process_RadioParser
- process_Sensors
- process_Braitenberg
- process_GetBox
- process_PushBox

6.1 process_RadioParser

Tento proces je spuštěn a je aktivní okamžitě po přenosu programu do robotu. Zajišťuje přijímání a odesílání zpráv mezi roboty. Funkce pro odesílání a přijímání zpráv (kontrolu messagebufferu) spouštíme v tomto procesu s časovým odstupem 1000ms. Tento čas používáme proto, abychom měli jistotu, že celá zpráva byla odeslána a že již bylo dokončeno uložení zprávy popř. vyjmutí zprávy z messagebufferu. Z praxe víme, že tato prodleva je nutná i proto, že jinak dochází k velkému zatížení neustálým čtením bufferu a uvíznutí programu. Popis funkcí zajištujících komunikaci mezi roboty jsme uvedli výše.

```
static void process_RadioParser1 ()  /////RADIO PARSER
{
    tim_suspend_task(1000);
for(;;)
 {
 printf("Proces 0 \r\n");
 Function_send();
 tim_suspend_task(1000);
 Function_receive();
 tim_suspend_task(1000);
 }//for
```

[1] Názvy procesů v tomto textu se liší od názvů procesů ve zdrojových kódech jen číslicí za názvem procesu. Číslice 1 nebo 2 používáme ve zdrojových kódech jen pro udržení logiky číslování, tzn. např. proces process_RadioParser je ve zdrojových kódech jednotlivých robotů pojmenován jako process_RadioParser1 ve zdrojovém kódu pro robot ID1 a jako process_RadioParser2 ve zdrojovém kódu pro robot ID2. Pokud bude v textu nutné oba procesy rozlišit, vždy uvedeme kompletní název procesu.

6.2 process_Sensors

Proces zajišťující asociaci mezi pointerem na strukturu typu IRSENSOR a jeho pojmenováním 16 znakovým řetězcem [18]. Struktura IRSENSOR se skládá ze dvou vektorů (každý 8 × 16 bitů), které obsahují hodnoty naměřené pomocí IR senzorů (tj. hodnoty okolního světla a hodnoty vzdáleností). Vytvoření asociace znamená, že se vytvoří globální (High level reference) reference pojmenovaná tímto 16 znakovým řetězcem. Potom všechny funkce a další procesy, které potřebují číst informace ze senzorů, získají pointer na strukturu pomocí této reference.
Pro názornost uvádíme kompletní kód tohoto procesu i se stručným komentářem:

```
static void process_Sensors1 ()
{
  int32 status;
  IRSENSOR *sensor;   //pointer na strukturu IRSENSOR
  sensor = sens_get_pointer ();
  status = tim_define_association ("IR sens", (uint32 *) sensor);
//vytvoření asociace
  if (status < 0)
    {
      exit (0);
    }
  exit (0);
}
```

6.3 process_Braitenberg

Proces realizující pohyb robotu v prostředí a vyhýbání se překážkám pomocí Braitenbergova algoritmu. Braitenbergův algoritmus vypočítává změnu rychlosti jednotlivých kol na základě hodnot vzdáleností z IR senzorů a jejich vynásobení vektorem koeficientů.

Obrázek 22 - Pohyb robotu v prostředí a hledání IR světla

Implementaci kódu pro pohyb robotu v prostředí podle Braitenbergova algoritmu jsme převzali z ukázkových příkladů zdrojových kódu prostředí KTProject.

```
static void process_Braitenberg1 ()  //BRAITENBERG
{
  int32 status;
  IRSENSOR *sensor;
  uint32 sensBuf[KNBSENSORS];
  int i, s, m;
  int32 potential[KNBMOTORS], speed[KNBMOTORS];
  int32 matrix[KNBMOTORS][KNBSENSORS] =  {{-5, -15, -18, 6, 4, 4,
3, 5}, {4, 4, 6, -18, -15, -5, 5, 3}};
  mot_config_speed_1m (0, 3800, 800, 100);
  mot_config_speed_1m (1, 3800, 800, 100);
  do
    {
      status = tim_find_association ("IR sens");
    }
  while (status < 0);
  sensor = (IRSENSOR *) status;
  for (;;)
    {
    if(BraitenbergEnabled == 1)
       {
      for (i = 0; i < KNBSENSORS; i++)
             sensBuf[i] = sensor->oProximitySensor[i];
      for (m = 0; m < KNBMOTORS; m++)
       {
         potential[m] = 0;
         for (s = 0; s < KNBSENSORS; s++)
           {
              potential[m] += (matrix[m][s] * (int32) sensBuf[s]);
           }
         speed[m] = (potential[m] / KSCALING) + KSPEED;
       }
      mot_new_speed_2m (speed[0], speed[1]);
      }
    }///nekonecny for
}
```

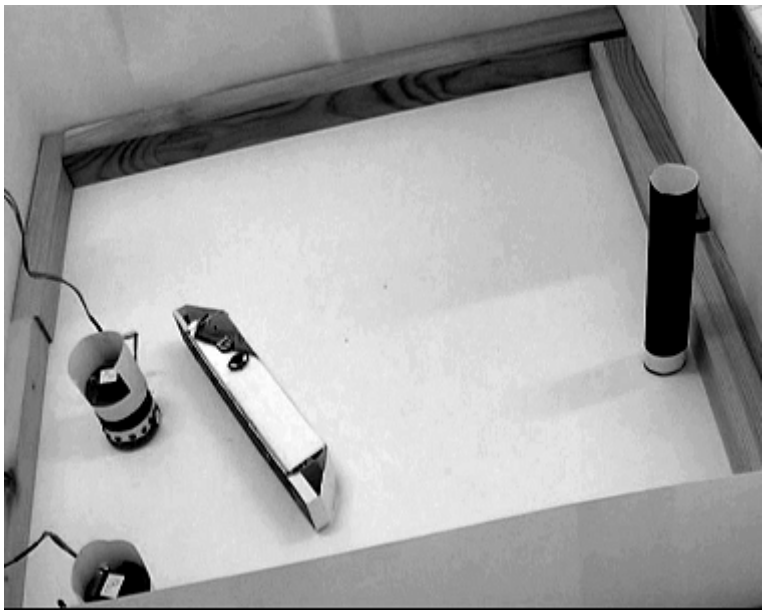
6.4 process_GetBox

V tomto procesu je implementováno hned několik kroků řešení. Zatímco proces process_Braitenberg řídí pohyb robotu v prostředí, proces process_GetBox zajišťuje hledání boxu a pohyb robotů k němu. Tento proces tedy realizuje snímání hodnot okolního světla, pohyb robotu směrem k boxu a v případě robotu ID1 i natočení boxu do pozice vhodné k tlačení směrem k cíli.

Kompletní kód tohoto procesu i s komentářem je uveden v příloze, proto uvádíme pouze sled jednotlivých důležitých kroků, které tvoří řešení a krátké fragmenty kódu ilustrující řešení jednotlivých částí..

Pokud je proces spuštěn a odblokován, tak neustále v cyklu while kontroluje stav IR senzorů. Pokud senzory zachytí IR světlo (viz Obrázek 23), vyzařované IR diodami boxu, okamžitě zastaví pohyb robotu a blokuje provádění procesu process_braitenberg. Příklad blokování běhu procesu, dokud není senzory identifikováno IR světlo:

```
do            //hledani svetla
 {}
while(Light()==0);  //svetlo nalezeno
```



Obrázek 23 - Zachycení IR světla bočními senzory robotu

V tomto okamžiku začne provádět pohyb robotu směrem ke zdroji světla (viz Obrázek 24). Pohyb robotu směrem k boxu je řízen tak, že je neustále vypočítáván součet intenzity světla v levé a pravé polovině senzorů robotu a podle těchto hodnot je nastavována rychlost jednotlivých motorků.

Obrázek 24 - Navigace robotu směrem k IR diodě v boxu

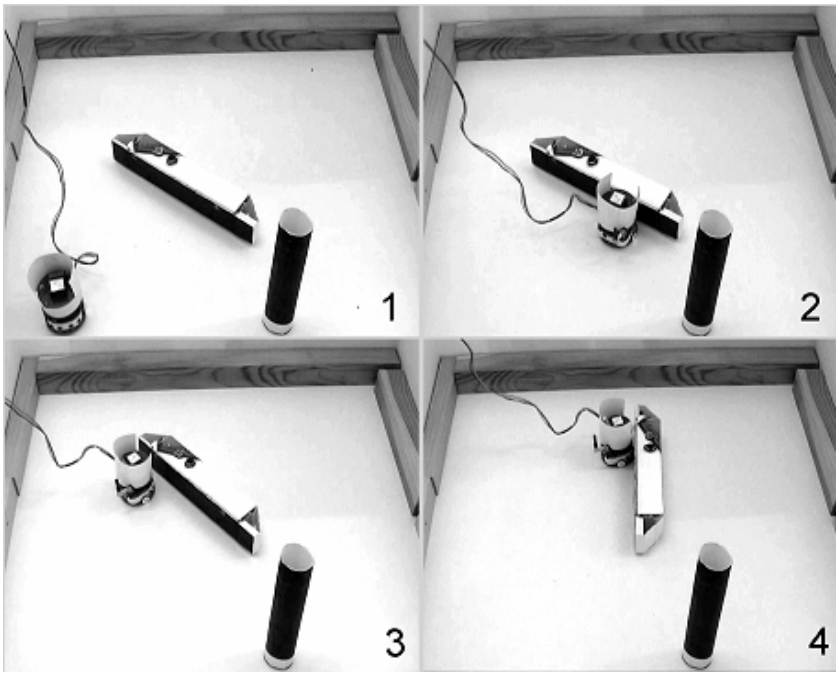
Řízení robotu směrem k cíli zajišťuje tato část kódu:

```
for(i=0;i<6;i++)
 {
 if(sensBuf[i]<IR_LIGHT_LIMIT)
       {
       if(i < 3)
                    leftIR +=sensBuf[i];
       if((i<6)&&(i>=3))
             rightIR +=sensBuf[i];
       }
 }
 if(leftIR<rightIR)
       {
       mot_new_speed_2m(-1,3);
       }
 if(leftIR>rightIR)
       {
       mot_new_speed_2m(3,-1);
       }
```

Zároveň se snímáním hodnot okolního světla je prováděno měření vzdálenosti k boxu. Pokud je senzory zachycen box (tzn. v případě použití černé barvy čelní strany boxu maximální hodnota 1023), je pohyb směrem k boxu zastaven (viz Obrázek 25).

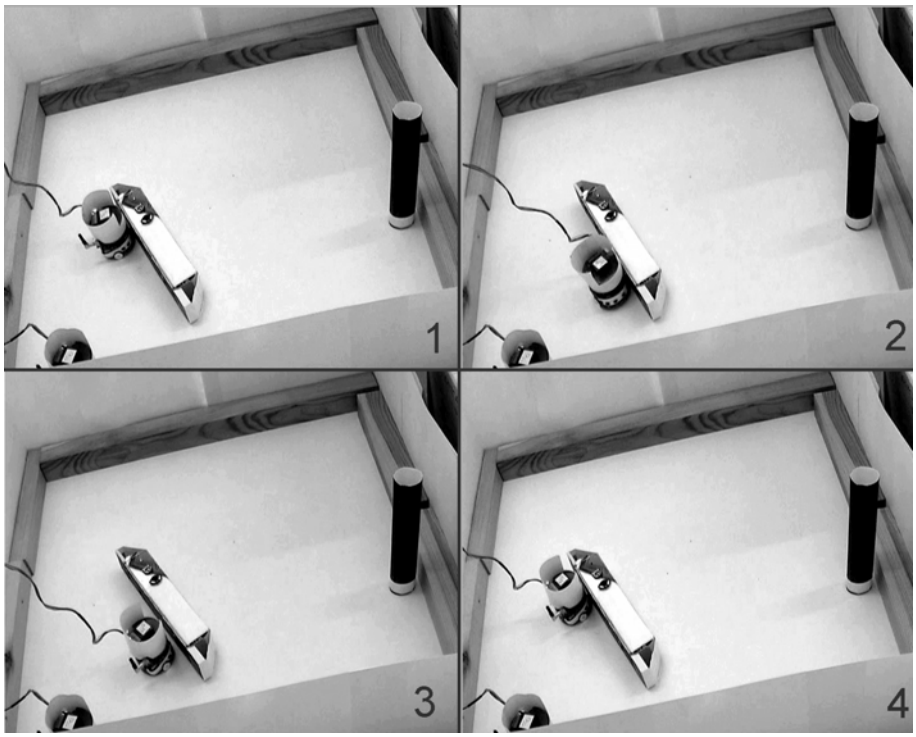
Obrázek 25 - Ustavení robotu u boxu a nastavení směrem kolmo k boxu

Poslední funkční částí tohoto procesu je natočení boxu do správné pozice pro tlačení. Princip natočení boxu do správné pozice vhodné pro tlačení je popsán v kapitole 4.2.2.



Obrázek 26 - Postup natočení boxu do správné polohy

Po ustavení robotu u boxu a jeho natočení do správné pozice je provedena kontrola, zda je robot ustaven u levé nebo pravé IR diody v boxu podle algoritmu uvedeném v kapitole 5.5.

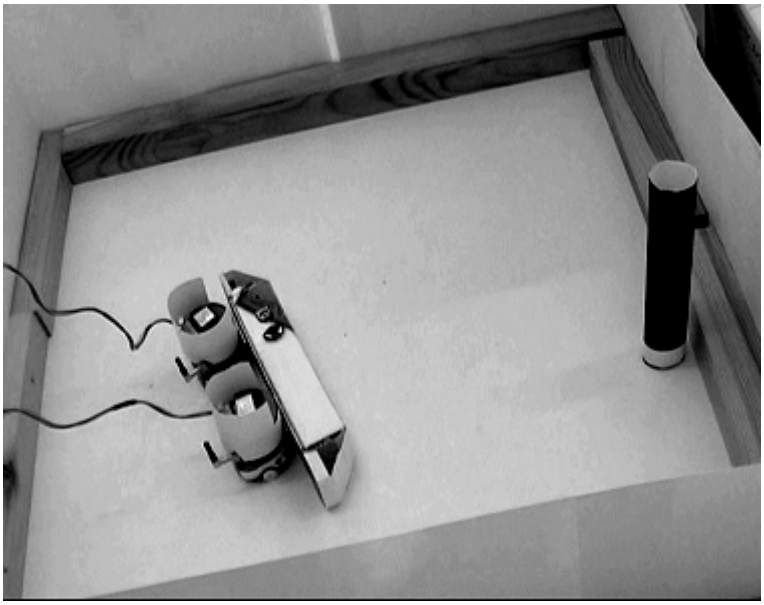


Obrázek 27 - Zjištění ustavení u levé nebo pravé IR diody

Po dokončení této kontroly je odeslána zpráva robotu ID2, aby zahájil svou funkci. Robot ID2 se pohybuje v prostředí stejně jako robot ID1 a ustaví se u druhé IR diody.

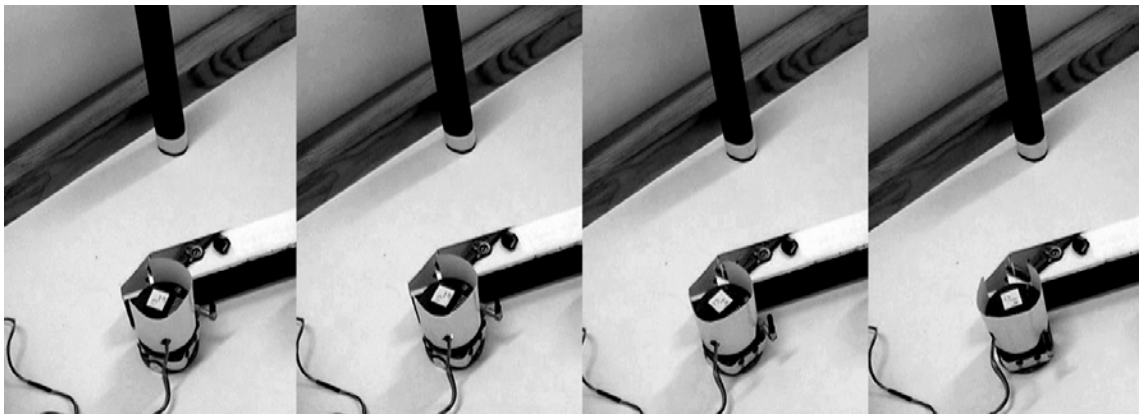
6.5 process_PushBox

Předpokladem pro aktivování tohoto procesu je ustavení obou robotů u boxu a jeho správné natočení. V tomto procesu je realizováno samotné tlačení boxu směrem k cíli oběma roboty a rozdělování jednotlivých příkazů robotům. Jedná se o hlavní a stěžejní proces celé úlohy.



Obrázek 28 - Výchozí pozice - ustavení robotů u boxu

Proces začíná tím, že robot ID1 změří svůj úhel k cíli (viz Obrázek 29 - Detail měření úhlu k cíli) pomocí funkce Image_angle_rotation a odešle zprávu robotu ID2, aby změřil svůj úhel otočení k cíli. Po přijetí zprávy od ID2, která obsahuje hodnotu úhlu otočení robotu ID2, provede robot ID1 vyhodnocení hodnot úhlů a rozhodne, který robot bude provádět tlačení.



Obrázek 29 - Detail měření úhlu k cíli

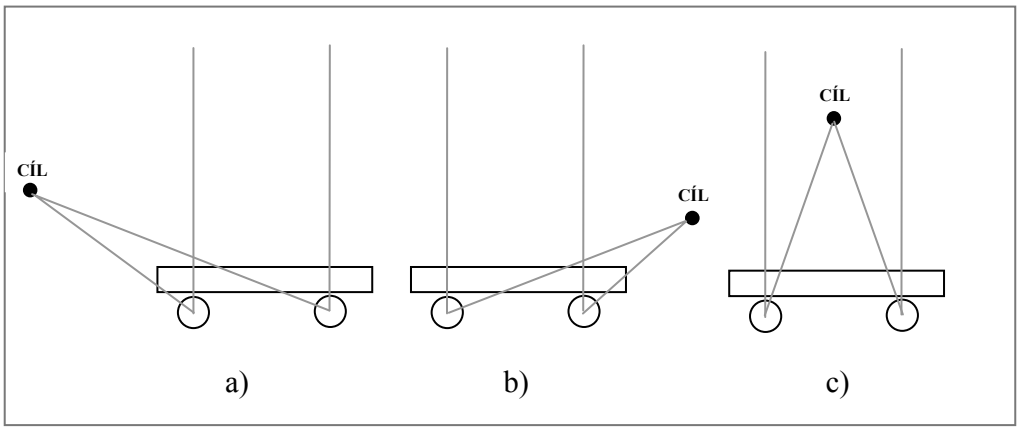
Hodnoty úhlů otočení jsou hodnoty z funkce Image_angle_rotation, tedy hodnoty v intervalu 0 až 11 (hodnota 0 znamená 60 ° vlevo, hodnota 11 odpovídá 60° vpravo).

Podle naměřených úhlů obou robotů můžeme rozlišit tyto možné situace (viz Obrázek 30):

- angleID1<=6 a angleID2<=6
  - oba roboty zachytily cíl vlevo od svého kolmého postavení a tlačení provede robot, který naměřil menší úhel (viz Obrázek 30a)

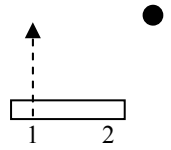
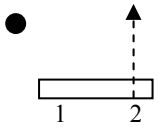
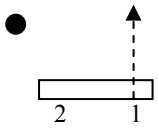
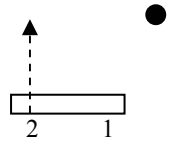
- angleID1>6 a angleID2>6
  - oba roboty zachytily cíl vpravo od svého kolmého postavení a tlačení provede robot, který naměřil větší úhel (viz Obrázek 30b)

- (angleID1>6 a angleID2<=6) nebo (angleID1<=6 a angleID2>6)
  - oba roboty zachytily cíl v prostoru před sebou a proto provedou tlačení oba roboty současně (viz Obrázek 30c a Obrázek 31)

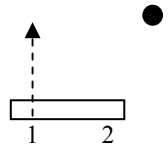
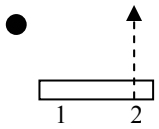
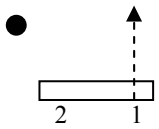
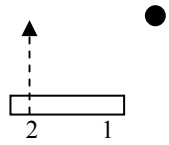


Obrázek 30 - Měření úhlů k cíli

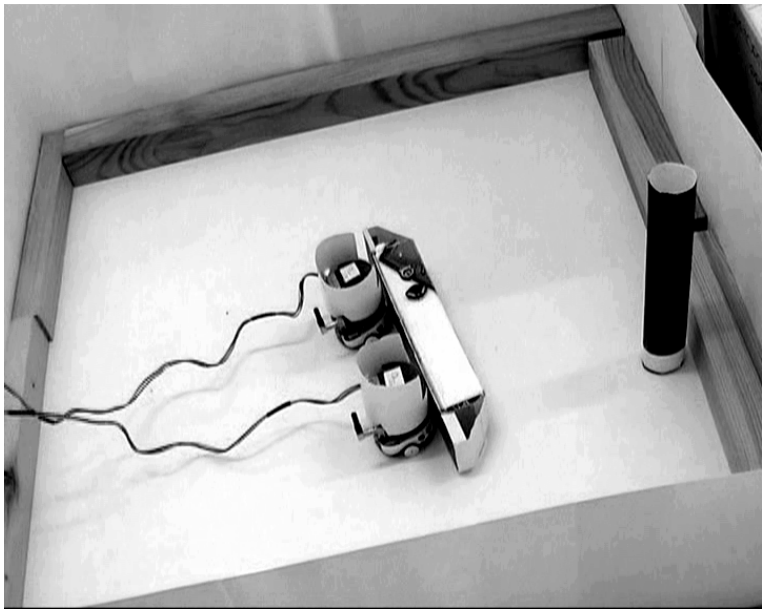
Může dojít k situacím, kdy jeden z robotů cíl nezachytí. K této situaci může dojít buď z důvodu zaclonění cíle druhým robotem nebo špatným vyhodnocením funkcí Image_angle_rotation. V tomto případě o tlačícím robotu rozhodneme podle velikosti úhlu robotu, který cíl zachytil a informace o tom, zda robot stojí vlevo či vpravo.
Popis těchto situací, když jeden z robotů při měření nezaměří cíl je popsán v následujících tabulkách (viz tabulka 2 a tabulka 3):

Úhel ID1	Úhel ID2	ID1 vlevo (L)/ vpravo(P)	Tlačení provede	Situace
X	≥ 0°	L	ID1	
X	< 0°	L	ID2	
X	≤ 0°	P	ID1	
X	> 0°	P	ID2	

tabulka 2 - Popis situací v případě, že robot ID1 nenaměří úhel k cíli

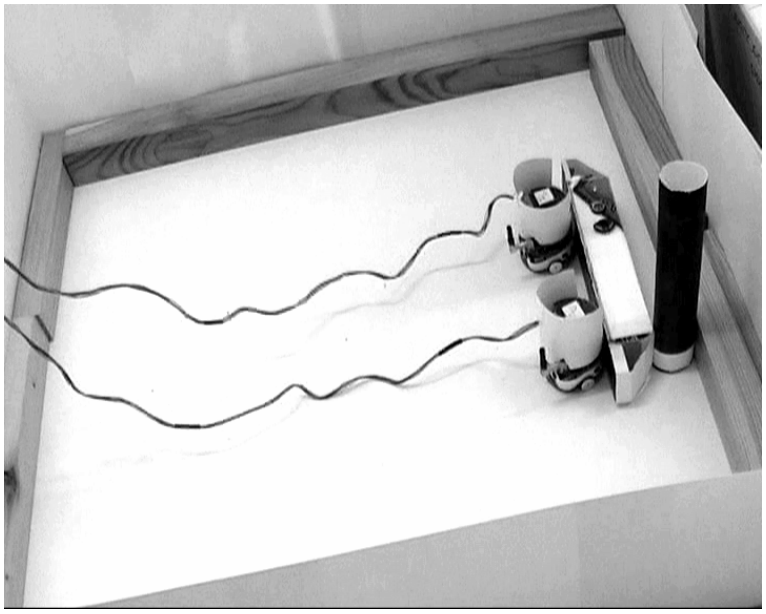
Úhel ID1	Úhel ID2	ID1 vlevo (L)/ vpravo(P)	Tlačení provede	Situace
≥ 0°	X	L	ID1	
< 0°	X	L	ID2	
≤ 0°	X	P	ID1	
> 0°	X	P	ID2	

tabulka 3 - Popis situací v případě, že robot ID2 nenaměří úhel k cíli



Obrázek 31 - Příklad situace při níž tlačí box oba roboty zároveň

Celý postup měření úhlu k cíli oběma roboty, vyhodnocení hodnot úhlů a tlačení směrem k cíli se opakuje až po dotlačení boxu k cíli.



Obrázek 32 - Dotlačení boxu k cíli

6.6 Psaní kódu, kompilace a přenos do paměti robotu

Všechny postupy a algoritmy řešení úlohy Box-Pushing a různých podúloh, uvedené v předchozích kapitolách, jsme implementovali na robotické platformě Khepera. V experimentu byly použity dva roboty Khepera s rozšiřujícími moduly K213 Vision a Radio Turret. Každý robot byl připojen k jednomu počítači. Zdrojový kód jsme editovali v prostředí KTProject a kompilovali cross-compilerem, který je součástí distribuce tohoto prostředí. Pro přenos zkompilovaného zdrojového kódu jsme využívali terminál KTerm, který je rovněž součástí distribuce KTProject.

Pro start experimentu je nutné nahrání obou zkompilovaných zdrojových kódů do paměti robotů. Pro správnou funkci je nutné nahrání kódu pro robot ID2 před nahrátím kódu pro robot ID1. Řešení úlohy začne automaticky po dokončení přenosu kódu pro ID1, proto je nutné, aby v této době byl již dokončen přenos kódu pro ID2.

Průběh řešení úlohy jsme zachytili kamerou a záznam je obsažen na přiloženém disku CD.

7 Závěr

V práci jsme popsali návrh řešení úlohy Box-Pushing za použití dvou komunikujících a kooperujících robotů. Postup a algoritmy řešení jsme navrhovali s přihlédnutím k možnostem a vybavení robotů Khepera. Domníváme se ale, že dekompozice řešení úlohy a většina algoritmů mohou být použity i pro řešení úlohy jinými roboty s podobným vybavením jako roboty Khepera. Některé implementované funkce mohou být použity i při programování řešení jiných úloh a experimentů s roboty Khepera.

Některé experimenty, jejichž výsledky jsme měli k dispozici, předpokládají splnění výchozích podmínek pro experiment, jako je např. správné natočení boxu směrem k cíli. V našem řešení jsme navrhli postup, který toto ustavení nutně nevyžaduje a box je před započetím tlačení nejprve natočen do správné polohy.

Několik experimentů používá k vyznačení cíle v prostředí zdroj světla a box vyrobený z průhledného materiálu, kterým světlo prochází, z důvodu možnosti identifikace boxu i cíle pomocí IR senzorů robotu. V našem experimentu využíváme box z neprůhledného materiálu opatřený IR diodami pro možnost rozlišovat stěny prostředí a box pomocí IR senzorů v pasivním a aktivním módu a lokalizaci cíle pomocí kamery. Vzhledem k tomu, že tedy máme k dispozici tři různé možnosti měření a identifikace, domníváme se, že naše řešení je aplikovatelné na množství různých výchozích situací v prostředí.

S použitím lineární kamery, kterou jsme použili pro identifikaci cílové pozice, by mělo být možné nejen identifikovat směr k cíli, ale při použití vhodného odlišení více podcílů (např. pomocí označení jednotlivých podcílů čárovými kódy), implementovat postupné tlačení boxu po trase vyznačené těmito podcíli.

Podobný experiment našemu experimentu a návrhu můžeme nalézt v [7], ale s jiným technickým vybavením robotů než mají roboty Khepera. Při našem návrhu jsme vycházeli z možností dostupného vybavení robotů Khepera a návrh všech algoritmů jsme podřizovali těmto skutečnostem. Návrh řešení by také bylo možné použít pro implementaci na robotické platformě Koala a využít širší možnosti a vybavení robotů

Koala (barevná kamera se standardem PAL, laserový skener pro horizontální snímání okolí v úhlu 180°).

Správnost návrhu algoritmů a implementace řešení Box-Pushing jsme ověřili experimentem v reálném prostředí. Experimenty, které jsme provedli, můžeme hodnotit jako úspěšné. Počet experimentů, v nichž jsme dosáhli správného vyřešení úlohy Box-Pushing, značně převládal nad počtem experimentů, při nichž jsme v důsledku chyb a nepřesností správného řešení nedosáhli.

Během implementace řešení jsme se potýkali i s některými problémy. Tyto problémy můžeme rozdělit do dvou skupin. První skupinou problémů byly problémy o kterých jsme se již několikrát zmínili, a to sice problémy proměnlivosti reálného prostředí, které jsou pro robotiku typické. Činnost robotů je většinou závislá na informacích, které robot získává ze svého prostředí pomocí svých senzorů a přesnost těchto informací se odvíjí od vlastností prostředí. S proměnlivostí prostředí souvisí nutnost návrhu řešení tak, aby robot byl schopen reagovat na nepředvídatelné situace, které mohou v prostředí nastat. Další skupinou problémů byly ty, které jsou způsobovány chybami technického vybavení robotů. Tyto chyby nemusí být nutně způsobeny jen nefunkčností jednotlivých komponent, ale třeba i opotřebením těchto jednotlivých částí. Příkladem takového problému může být opotřebení a nepřesná funkce motorků a kol robotů Khepera. Při řešení naší úlohy jsme se setkali například s tím, že po nastavení hodnot pro pohyb robotu rovně vpřed se robot vpřed pohyboval ne přímočaře, ale „vlnitým“ pohybem a cílová pozice, které měl robot dosáhnout se lišila od skutečné pozice do které se robot přesunul.

Domníváme se, že i přes všechny tyto problémy, se nám podařilo navrhnout správné a funkční řešení a že principy a algoritmy uvedené v této práci mohou inspirovat další zájemce o programování robotů Khepera a některé implementované funkce a části kódu mohou být využity i v dalších aplikacích. Problémy a řešení uvedené v této práci mohou také sloužit těm, kteří se programováním robotů nezabývají, k vytvoření představy o programování robotů, různých přístupech a řešení úloh mobilní robotiky.

Literatura a použité zdroje

[1] ARKIN, Ronald C., BEKEY George A., *Robot Colonies*, Kluwer Academic Publisher, Norwell, MA, USA, 1997, ISBN 0-7923-9904-8.

[2] CAO Y. Uny, FUKUNAGA Alex S., KAHNG Andrew B., MENG Frank, *Cooperative Mobile Robotics: Antecedents and Directions*, in `IEEE/TSJ International Conference on Intelligent Robots and Systems', Yokohama, Japan

[3] MURPHY, Robin R., *Introduction to AI robotics*, The MIT Press, 2000, ISBN-10: 0-262-13383-0, ISBN-13: 978-0-262-13383-8.

[4] KUBÍK, Aleš, *Inteligentní agenty: tvorba aplikačního software na bázi multiagentových systémů*, Computer Press, Brno, 2004,ISBN:80-251-0323-4.

[5] SPRINKHUIZEN-KUYPER Ida G., *Artificial Evolution of Box-Pushing Behaviour*, Universiteit Maastricht, IKAT, The Netherlands.

[6] GERKEY Brian P., MATARIC Maja J., *Pusher-watcher: An approach to fault-tolerant tightly-coupled robot coordination*, in submission to IROS, 2001.

[7] KUBE, C. Ronald , ZHANG, Hong. *The use of perceptual cues in multi-robot box-pushing,* Department of Computing Science, University of Alberta, Edmonton, 1996.

[8] YAMADA S., SAITO J., *Adaptive Action Selection without Explicit Communication for Multi-robot Box-pushing*. To appear in 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems.

[9] MATARIC M.J., NILSSON M., SIMSARIAN K.T., *Cooperative multi-robot box-pushing*. In IEEE/RSJ IROS, pages 556--561, 1995.

[10] PARKER Lynne E., *Alliance: An architecture for fault tolerant cooperative control of heterogeneous mobile robots*. In Proc. IROS 1994, Munich, Germany, Sept. 1994.

[11] K-TEAM, *Khepera User Manual,* Laussane, 1999.

[12] HOEKSTRA, Robert L., Robotics and automated systems, Robotíce, South-Western Publishing Co., Cincinnati, Ohio, 1986, ISBN: 0-538-33650-1.

[13] K-TEAM, Khepera K213 Vision Turret User Manual, K-Team, Laussane, 1999.

[14] K-TEAM, Khepera Radio Turret User Manual, K-Team, Laussane, 1999.

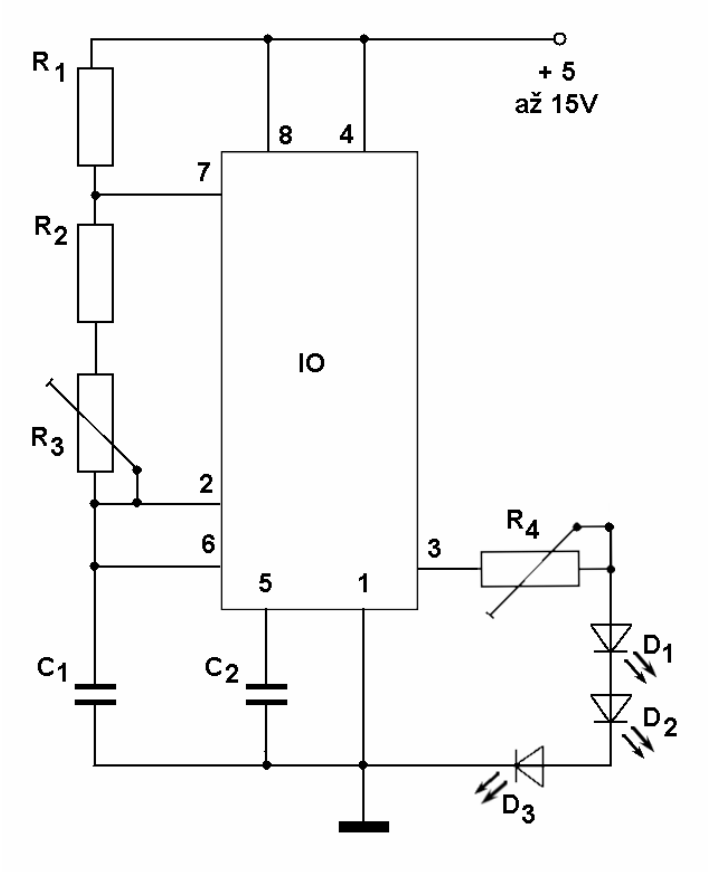
[15] K-TEAM, Khepera Radio Base User Manual, K-Team, Laussane, 1999.

[16] NEMRAVA, Michal. *Aplikace pro roboty Khepera*., 2003, Slezská univerzita v Opavě. Bakalářská práce.

[17] K-TEAM, Khepera 2 Programming Manual, Laussane, 2002.

[18] K-TEAM, Khepera BIOS Manual, Version 5.0.1, Laussane, 1999.

Příloha A - Popis obvodu a zapojení IR diod



Zdrojem signálu jsme zvolili zapojení s integrovaným obvodem NE 555 v astabilním módu. Kmitočet zařízení je určen velikostí kapacity časovacího kondenzátoru C_1 a rezistorů R_1 a R_2 podle vztahu

$$f = \frac{1.49}{[R_1 + 2 \cdot (R_2 + R_3) \cdot C_1]}$$

Na výstupu zařízení dostáváme signál pravoúhlého průběhu.

K zajištění možnosti změny nastaveného kmitočtu jsou použity dva rezistory zapojené v sérii - R2 pevný a R3 nastavitelný, kterým lze volit požadovaný kmitočet.

Na výstupu zařízení jsou sériově zapojeny proměnný rezistor R_4, dvě IR diody D_1 a D_2 a LED dioda D_3. Proměnným rezistorem je možné nastavit intenzitu signálu a tím i dosah IR světla vysílaného IR diodami. LED dioda je použita pro vizuální kontrolu vysílacího signálu a kontrolu funkce celého zařízení. Zařízení je napájeno destičkovou baterií 9V.

Hodnoty použitých součástek:

R_1, R_2	12kΩ
R_3	220kΩ
R_4	470Ω
C_1	4,7μF
C_2	10nF
IO	NE555
D_3	LED 2x5mm červená difůzní
D_1,D_2	IR LED 5mm modrá 3000mcd/20°

Příloha B - Disk CD

Přiložený disk CD obsahuje:

- zdrojové kódy programů pro roboty Khepera
- binární soubory těchto programů
- videa z experimentu ve formátu MPEG-2 (adresář VIDEO)