ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

Katedra technologií a měření

DIPLOMOVÁ PRÁCE

Paralelní polohování miniaturních robotů pomocí magnetického pole

Bc. Tomáš Šašek

Plzeň 2017

Paralelní polohování miniaturních robotů
pomocí magnetického pole

Bc. Tomáš Šašek

Anotace

Předkládaná diplomová práce je zaměřena na simulaci paralelního polohování miniaturních robotů v magnetickém poli. Práce je rozdělena do dvou, dále podrobněji rozdělených částí. První část je věnována teorii dané problematiky. Je zde popsána problematika algoritmů, jak hejnových, tak ovládání pomocí jednoho globálního vstupu. Dále ovládání robotů pomocí magnetického pole, omezení, s tím spojená a popis prototypů. Následující část je věnována praktickému řešení simulace paralelního polohování. Text tedy obsahuje postup při tvorbě skriptu, včetně popisu řešeného problému, užitých algoritmů a následnému zhodnocení provedených simulací.

Klíčová slova

Algoritmus, paralelní polohování, mikroroboti

Abstract

This diploma thesis is focused on simulation of the parallel placement of miniature robots in the magnetic field. The thesis is divided into two parts, then divided in more detail. The first part is devoted to the theory of the given issue. The issue of algorithms of both flock and parallel control is described there. Furthermore there is description of robot control by means of a magnetic field, constraints, associated with it and prototype descriptions. The following part is devoted to the practical solution of simulation of positioning with one global input. The text then contains the procedure for creating the script, including the description of the problem, the algorithms used and the subsequent evaluation of the simulations.

Key words

Algorithm, parallel positioning, micro-robots

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

V Plzni dne 21.5.2017 Bc. Tomáš Šašek

Poděkování

. Tímto bych rád poděkoval vedoucímu diplomové práce Ing. Františku Machovi, Ph.D. za cenné profesionální rady, připomínky a metodické vedení práce.

OBSAH

Úvod

Předkládaná práce se zabývá problematikou paralelního polohování miniaturních robotů pomocí magnetického pole.

Obsah práce je rozdělen do šesti částí. První dvě části se zabývají teoretickou stránkou problematiky, kdežto části následující jsou zaměřeny na více praktickou část, kde je probrána problematika ovládání robotů magnetickým polem, popis prototypů pro vykonávání tohoto pohybu a simulace s vyhodnocením jejích výsledků.

Již zmíněná první část se zabývá přiblížením problematiky algoritmů jak hejnových tak algoritmů paralelního polohování robotů a současných robotických systémů využívajících kolektivní spolupráce.

1 Definice základních pojmů

Pro úvod následujících kapitol této práce je tato kapitola věnována definici několika základních pojmů. Jako jsou hejnové algoritmy, jejich vlastnosti a kritéria.

1.1 Hejnové algoritmy

Hejnové algoritmy jsou inspirovány sociálními zvířaty. Nejčastějšími takovými zvířaty jsou mravenci, ptáci, včely ale i například ryby. Zástupci těch společenství zvířat jsou jedineční tím, že jakožto celek vykazují určitou formu hejnové inteligence. Tento pojem, hejnová inteligence, jako první použil Bonabeau v roce 1999.

Právě tato zvláštnost se stala inspirací pro multi-agentní použití robotů k plnění různých úkolů. Roboti využívaní v této souvislosti mají několik charakteristických vlastností. Mezi tyto vlastnosti patří samostatnost, snímací a komunikační schopnosti robota jsou pouze lokální, nemají přístup k centralizovanému ovládání ani k tak zvaným globálním znalostem, zadané úkoly plní kooperací s ostatními roboty.

Chování celých hejn se specifikuje z pravidla třemi základními hledisky a to konkrétně robustností, škálovatelností a flexibilitou.

- ***Robustnost***

Robustnost se rozumí jako vlastnost hejna se vyrovnat se ztrátou členů hejna. Tato vlastnost je v přírodě reprezentována jako nadbytek členů a absence vedoucího člena.

- ***Škálovatelnost***

Škálovatelnost vyjadřuje schopnost hejna fungovat efektivně s různými počty členů hejna. Odstraňování členů v tomto ohledu neovlivňuje efektivitu.

- ***Flexibilita***

Hejno, které se rychle a efektivně přizpůsobí různým prostředím a úkolům je označováno jako hejno s vysokou flexibilitou.

Hejnová robotika se snaží vytvořit robotické systémy, které vykazují stejné inteligentní vlastnosti podobné těm, co můžeme sledovat u společenských zvířat a tím nabýt dobré robustnosti, škálovatelnosti a flexibility [1].

1.2 Snímání

Další velmi důležitou vlastností, kterou nesmíme opomenout je schopnost plánovat činnosti jednotlivců bez jednoznačného vůdce nebo člena, který by naváděl ostatní členy tak, aby plnili zadaný úkol.

Podle článku Collective Robotics: From Social Insects to Robots napsaného C. Ronald Kube a Hong Zhang [2] a dalších, odpověď na tuto otázku mohou poskytnout například mravenci a jejich schopnost vnímání okolí.

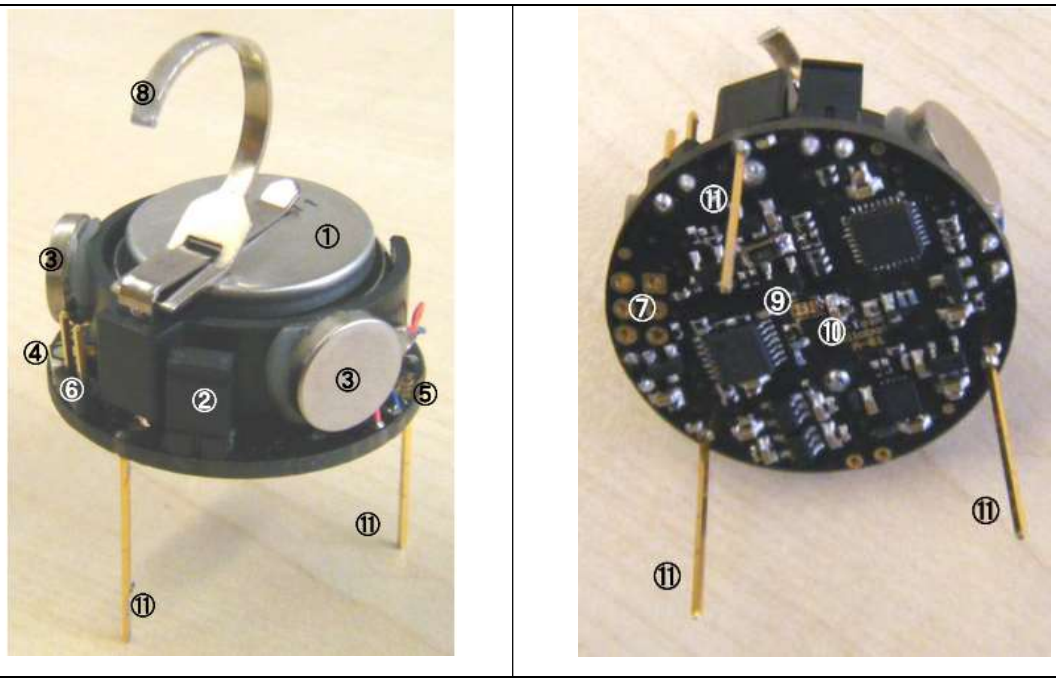
V tomto článku jsou mravenci popsáni jako malý roboti naprogramovaní k plnění specifických úkolů. Přičemž jejich nervový systém se chová jako biologický počítač. Který je aktivován pomocí štípacích lístků pokaždé, kdy jsou jejich receptory konkrétně stimulovány. Jejich receptory reagují na tlak, zvuk, světlo, teplo a chemikálie. Článek je soustředěn na čtyři hlavní druhy a to mravence, termity, včely a vosy. Z těchto druhou jsou nejvíce popsány včely a již zmínění mravenci.

U mravenců byli prováděny pokusy, které potvrdili tyto domněnky. Konkrétně se podařilo vyvolat u ostatních členů kolonie iluzi o úmrtí jednoho člena, tak, že byl potřísněn acetonem. Jak se ukázalo, právě aceton evokuje u mravenců naprogramované chování pro identifikaci a odklizení mrtvého člena kolonie.

2 Stávající robotické systémy

Odstavce v této kapitole jsou věnovány několika zástupcům robotických systémů využívajících velkého množství robotů k plnění různých úkolů.

2.1 Kilobot



Obrázek 1: Kilobot [3]
(1) 3,7 V Baterie (2) Startér (3) Vibrační motory (4) LED (5) Snímač okolního světla (6) Hlavička sériového výstupu (7) Socket pro přímé programování (8) Napájení (9) IR vysílač (10) IR přijmač (11) Robotická noha

Kilobot je navržen pro zkoušky kolektivních algoritmů na stovkách nebo tisících robotů a zároveň pro snadnou dostupnost výzkumníkům.

I přes nízké náklady, Kilobot udržuje schopnosti podobné ostatním kolektivním robotům. Mezi tyto schopnosti patří diferenciální pohon, výpočetní výkon, komunikace soused-soused, vzdálenosti soused-soused snímačů a snímání okolního světla.

Navíc jsou navrženy, aby fungovaly bez vyžadování individuální pozornosti lidského operátora. To umožňuje snadné ovládání skupiny Kilobotů, ať už je ve skupině 10 nebo 1000 robotů. Kilobot byl vyvinut na prestižní Harvardské univerzitě a nyní je vyráběn a distribuován společností K-Team.

Kilobot je levný a snadno použitelný robotický systém pro rozvíjení "rojů" robotů, které mohou být naprogramovány tak, aby plnili užitečné funkce koordinací interakcí mezi mnoha jednotlivci. Tyto roje jsou inspirovány společenským hmyzem, jak je popsáno v dále v práci.

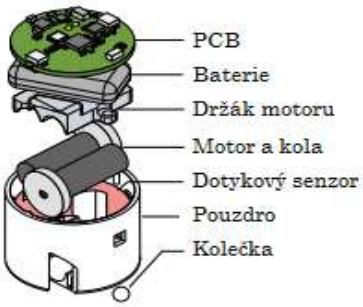
Po této inspiraci z přírody by robotické roje mohly jednou tunelovat sutinami, aby našli přeživší, sledovali životní prostředí a odstraňovali kontaminující látky, pomáhali snižujicí se populaci včel a samy se shromáždili, aby vytvořili podpůrné konstrukce ve zhroucených budovách. Kilobot je navržen tak, aby poskytl vědcům fyzickou zkušební plochu pro zlepšení chápání kolektivního chování a realizaci potenciálu poskytování řešení pro širokou škálu výzev.

Konstrukce Kilobota je popsána *obrázkem 1*, díky této konstrukci patří mezi charakteristiky Kilobotů například:

- Nízké náklady
- Malý, průměr jen 33 mm
- Jemné ovládání motoru (255 různých úrovní výkonu)
- Kilobot může komunikovat se sousedy až do 7 cm.
- Snímání vzdálenosti od sousedního souseda
- Snímání okolního světla
- RGB LED
- Nabíjecí a výměnná baterie
- Snadná manipulace s ovladačem Kilobot můžete naprogramovat a ovládat stovky kilobotů najednou.

2.2 Zooidy

Zooidy stavějí na práci z robotiky rojů, přidávají interakci a rychlost. Uživatelské rozhraní Swarm jsou rozhraní vytvořená pomocí kolekcí samojízdných fyzických objektů, které se mohou společně pohybovat a reagovat na vstup uživatele. Uživatelské rozhraní rojů lze považovat za hrubozrnnou verzi futuristických vizí Sutherlandových a Ishiových uživatelských rozhraní založených na programovatelných hmotách [4].

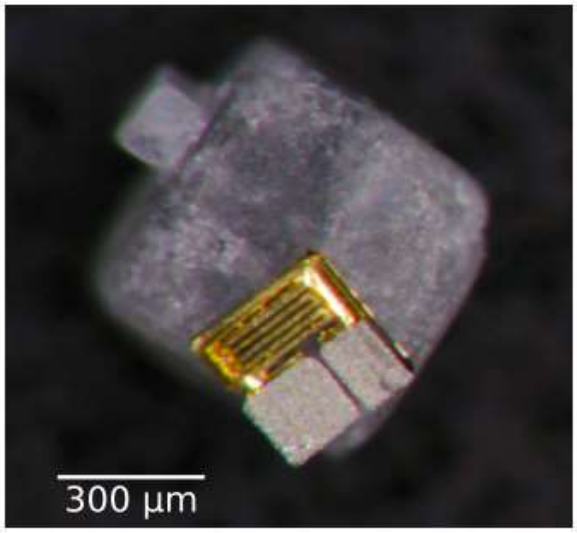


Obrázek 2: Rozvinutý pohled na Zooid [4]

Díky schopnosti zooidů volně a rychle se prostorově konfigurovat, může sbírka zooidů fungovat jako displej a může poskytnout významný uživatelský výstup. Vzhledem k tomu, že mohou chápat akce uživatelů, mohou zooidy také podporovat vstup. Například uživatelé mohou buď posunout zooidy jeden po druhém, nebo manipulovat s mnoha zooidy najednou pomocí "zametacích" gest. Na straně aplikace mohou být implementována sofistikovaná interaktivní chování, např. Zooidy mohou působit jako ovládací prvky nebo jako manipulátory pro manipulaci s jinými zooidy. Mohou dokonce přemístit další lehké objekty [4].

Současně, jelikož všechny vstupy a výstupy mohou být zprostředkovány prostřednictvím stejných fyzických prvků (*obrázek 2*), je systém schopen dosáhnout úplné fúze mezi vstupy a výstupy a poskytnout tak kompletní zážitek z fyzické manipulace. Systém je relativně lehký a vyžaduje pouze kompaktní DLP projektor (122 mm × 115 mm × 48 mm) pro optické sledování. Zooidy mohou pracovat na libovolném vodorovném povrchu (např. List papíru, špinavý kancelářský stůl, jídelní taška nebo herní deska), což umožňuje spojit uživatelské rozhraní rojů s každodenním fyzickým prostředím. Pro stimulaci budoucího výzkumu na uživatelských rozhraních rojů je platforma uživatelského rozhraní Zooidů open-source [4].

2.3 Magmites



Obrázek 3: MagMite na povrchu zrnka soli

Magmites jsou magnetické aktuátory využívající bezdrátové technologie. Tato zařízení těží magnetickou energii ze svého prostředí a mění ji na mechanický pohyb, který je plně ovladatelný. Tito mikrorobotičtí agenti o rozměrech méně než $300\mu m \times 300\mu m \times 70\mu m$ jsou schopni manévrovat ve třech stupních volnosti. Speciálně připravený substrát umožňuje nastavitelnou rychlost pohybu přesahující 12,5 mm/s, což je 42 násobek celé délky robota za sekundu. Je poháněn oscilujícím polem o několika kHz a sílou nižší než 2 mT [5].

2.3.1 Princip

Metoda pro bezdrátovou mikroaktuaci, využívá vytváření interaktivních sil mezi měkkými magnetickými tělesy vystavenými vnějšímu magnetickému poli. Magnetická pohonná jednotka mikrorobotů se skládá ze dvou měkkých magnetických těles z niklu, spojených přes nemagnetickou pružinu. Jedno tělo je spojeno se zlatým rámem, který je umístěn na substrátu. Připevněný k rámu, avšak asi 10 µm nad substrátem, je meandrová pružina, která nese druhé těleso niklu v poloze k zemi [5].

V homogenním magnetickém poli se měkké magnetické těleso otáčí tak, aby vyrovnala jejich kombinovanou dlouhou osu s vnějším polem - podobně jako kompasová jehla. Indukovaná magnetizace těles vede k protilehlým magnetickým pólům přes mezeru. Přitažlivé síly mezi oběma těly způsobují, že mezera se zužuje a se nakonec zavře, když se zavěšené tělo odvrací. Když je pole vypnuto, magnetické síly zmizí a pružina vrátí zavěšené těleso do rovnovážné polohy [5].

Řízení pružinového systému při rezonanci vede k větším průhybům při zachování minimální amplitudy použitého pole. Jak dochází k nárazu mezi těmito dvěma tělesy z niklu, pružinové těleso přenáší svou hybnost na stacionární těleso připojené k základnímu rámu. To pak obrátí svůj pohyb a znovu zahájí oscilační cyklus. Systém mechanické pružiny musí být navržen tak, aby režim nejnižší rezonance byl v požadovaném směru ovládání při zachování magnetických vlastností systému [5].

2.3.2 Ovládání

Magmit lze plně ovládat a je schopen pohybu vpřed, vzad, zastavení, otáčení v obou směrech a dokonce i otáčení na místě. Orientace robota je přirozeně řízena orientací vnějšího pole vytvořeného dvěma ortogonálními dvojicemi Helmholtzových cívek. Magmit lze řídit klávesnicí v mírných rychlostech nebo pomocí plně automatizovaného vizuálního servozesilovače při vyšších rychlostech [5].

Lineární pohyb robota je řízen následujícím způsobem: Za správných provozních podmínek je dostatečný přenos hybnosti mezi kyvadlovým a stacionárním tělesem k překonání statického tření a každý náraz způsobuje, že robot se posune o malou vzdálenost. Pro zjednodušení řízení pohybu vpřed a vzad může být robot provozován na upraveném substrátu s izolovanými elektrodami. Fázově pevnými a fázově posunutý elektrostatický potenciál drží robota u podkladu během určitých částí oscilačního cyklu, což uživateli poskytuje větší kontrolu nad třecími silami a umožňuje robotu pohybovat se dopředu i dozadu se správnými fázovými posuny [5].

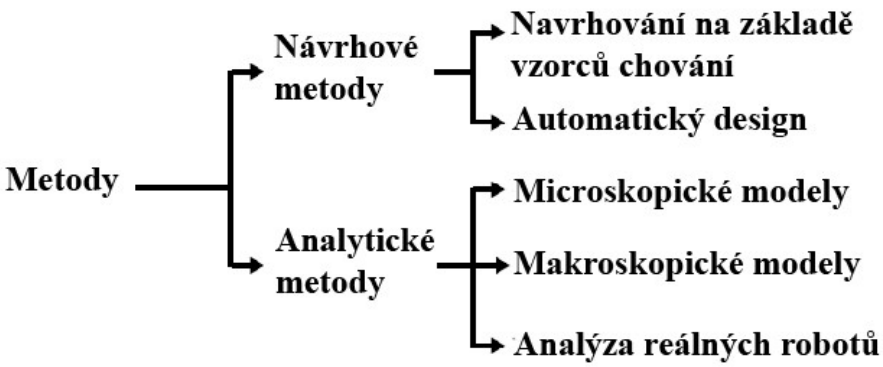
Rychlost robota je nejlépe řízena metodou podobnou modulaci šíře impulzů. Pro dané období n cyklů je definován poměr mezi normálními pohybovými cykly a cykly, během nichž je robot plně přichycen. Pro dosažení poloviční rychlosti se dá využít udržováním robota v plně upnutém stavu během prvních 50 cyklů 100 cyklové periody a jeho pohyb pouze během druhé poloviny [5].

3 Algoritmy používané pro řízení

V následujících dvou kapitolách jsou popsány a hejnové algoritmy a algoritmy s globálním vstupem.

3.1 Metody navrhování

V nadcházejících odstavcích jsou popsány metody navrhování hejnových systémů. V odstavci 2.1 jsou popsány návrhové metody navrhování takovýchto systémů a v odstavci 2.2 analytické metody. Přesně jak popisuje *obrázek 4*.



Obrázek 4: Rozdělení metod[1]

3.1.1 Návrhové metody

Návrh je fáze, ve které je systém plánován a vyvíjen podle základních specifikací a nároků. Bohužel, v hejnové robotice zatím nejsou žádné formální nebo přesně popsané postupy k navrhování jednotlivých úrovní chování, které produkují vytoužené kolektivní chování. Právě proto je zatím nejdůležitějším faktorem ve vývoji těchto robotických systémů intuice lidského návrháře [1].

Při navrhování můžeme postupovat jednou ze základních metod, kterými jsou navrhování na základě vzorců chování nebo automatický návrh.

Navrhování na základě vzorců chování

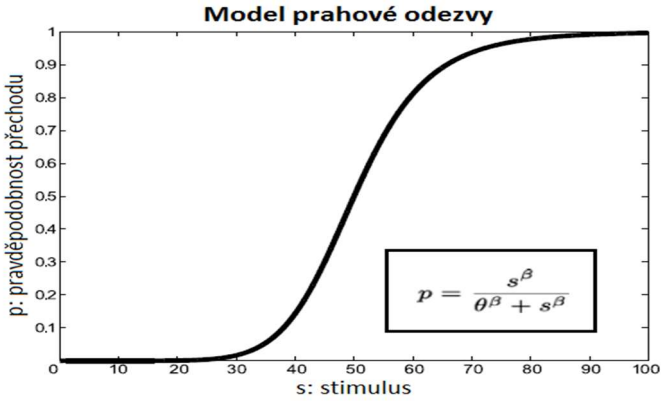
V tomto odvětví roboty (hejnová robotika) je tato metoda nejčastěji používána. Jedná se o metodu založenou v zásadně na procesu pokus omyl. Jednotlivé vzorce chování jsou postupně upravovány a dolaďovány tak, aby ideálně souhlasili nebo se co nejvíce blížili požadovanému kolektivnímu chování. Z tohoto důvodu je tato metoda označovaná jako zpětná [1].

Tento způsob navrhování můžeme ještě dále rozdělit na tři základní kategorie a to na návrh pomocí pravděpodobnostního konečného automatu, pomocí užití virtuální fyziky a ostatní metody.

- *Pravděpodobnostní konečný automat*

Co se týče kolektivních robotických systémů, není běžné, aby některý z členů hejna plánoval své budoucí kroky, ale rozhoduje se na základě senzorických vstupů a/nebo vnitřní paměti. Proto se hojně využívá metody pravděpodobnostního konečného automatu (MPKA).

V MPKA může být pravděpodobnost přechodu mezi stavy předem upravena nebo se může s časem měnit. Pravděpodobnost přechodu je fixní, když je jedna hodnota pravděpodobnosti určená a používání po celý proces kolektivního rozhodování. V případě, kdy není tato pravděpodobnost, nemá fixní hodnotu, musí být určena matematickou funkcí jednoho nebo více parametrů systému. Jednou z nejvíce používaných funkcí pro tento účel je funkce vytvořená Granovetterem roku 1978. Tato funkce byla použita pro studii kolektivního chování společenství hmyzu, což zobrazuje *obrázek 5*[1].



Obrázek 5: Funkce prahové odezvy u společenského hmyzu [1]

- *Virtuální fyzika*

Tato metoda je inspirována fyzikou a to ve smyslu, že každý robot je považován za jednu virtuální částici, která vyzařuje nějaké virtuální síly na ostatní roboty.

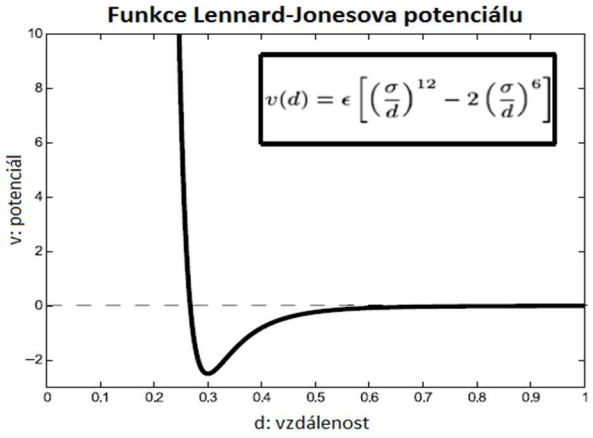
Virtuální fyzika je metoda usuzující, že roboti jsou schopni spatřit a rozpoznat sousedící roboty a překážky, také odhadnout jejich vzdálenosti a relativní polohu. Každý robot počítá vektor virtuální síly podle *rovnice 1* [1].

$$f = \sum_{i=1}^{k} f_i(d_i) e^{j\theta_i} \qquad 1)$$

Kde θ_i a d_i jsou směr a vzdálenost i-té pozorované překážky nebo robota. Funkce f_i (d_i)je odvozená od umělé potenciálové funkce, pro kterou je nejpoužívanější Lennard-Jonesova potenciálu, což je zobrazeno na *obrázku 6*.

Navrhování s použitím virtuální fyziky má několik výhod, například následující [1].

- *Jediné matematické pravidlo, které snadno překládá veškerý senzorický vstup do výstupu aktuátoru bez potřeby několika vzorců chování a pravidel.*
- *Obdržené vzorce chování se dají kombinovat za použití vektorových operací.*
- *Některé předpoklady mohou být ověřeny za použití teoretických nástrojů fyziky, teorie ovládání nebo grafických metod*



Obrázek 6: Funkce Lennard-Jones potenciálu [1]

Automatický design

Použití této metody umožňuje automatické generování vzorců chování bez značné intervence vývojáře. I tato metoda se dá dále rozdělit, konkrétně do dvou kategorií a to podpůrné učení a evoluční robotika.

- *Podpůrné učení*

Sada metod, které automaticky navrhují individuální vzorce chování pro roboty v hejnu je k nalezení v různé literatuře. Tato metoda tradičně odkazuje na třídu učících problémů. Agent se naučí určitý vzorec chování skrze proces pokus omyl v interakci se svým okolím, kdy je mu poskytnuta negativní nebo pozitivní zpětná vazba na jeho akce [1].

Zda může být metoda podpůrného učení použita v případě kolektivních systémů je diskutabilní. Tato problematika totiž není typickým problémem, který je řešen pomocí podpůrného učení. Podpůrné učení totiž pracuje na systému odměňování jednoho člena za své akce, což je důvodem pro obtížnou aplikaci na hejno, případně i hejna robotů. V takových případech by se musela odměna, tedy naučený vzorec přerozdělit mezi všechny členy hejna, který jej využijí. S tím, že není možné počítat se stejnými podmínkami, ani se stejným okolím pro všechny členy hejna v jeden okamžik.

Řešení tohoto problému se nazývá přiřazením prostorového kreditu, které byl použito Matarićem v letech 1997 a 1998, za použití 2 až 4 robotů.

3.1.2 Analytické metody

Analýza je základní fáze ve vývojovém procesu, zaměřující se zpravidla na to, zda navržený systém plní požadavky na něj kladené, nebo ne. Tato analýza se většinou provádí na určitém modelovém objektu nebo skupině objektů jak je tomu většinou u hejnových systémů.

Tyto systémy mohou být modelovány na dvou úrovních. Jak již bylo zmíněno na jednotlivém robotovi (mikroskopická) nebo v měřítko celého systému (makroskopická), případně vzorové skupiny systému, když je jednotek zbytečně mnoho pro tyto analýzy. Obě tyto úrovně jsou velice důležité pro vytváření návrhu a vývoj systému jako takového [1].

Mikroskopické modely

Na této úrovni je zkoumáno nejen, jak robot ovlivňuje ostatní roboty, ale také své okolí a vice versa. Hloubka detailů této analýzy se odvíjí od mnoha faktorů, což ovlivňuje výsledky obdržené z provedené analýzy [1].

Pro většinu případů se pro analýzy systémů skládajících se z mnoha elementů využívá počítačových simulací. Simulace patří mezi nejvíce platnou metodu analyzování těchto systémů. V simulaci je možné zkoumat i vzorce chování hejn, která byli popsány v kapitole 2.1.1.

Závěrečná část této práce, věnovaná simulaci a jejímu zhodnocení je provedena na základě právě mikroskopického modelu.

Makroskopické modely

Jak již napovídá název kapitoly, makroskopické modely se zaměřují na analýzu systémů jako jednotného celku a jednotlivé elementy systému v potaz neberou.

Práce v makroskopickém modelování lze rozdělit do tří kategorií. Z nich první kategorií je označováno vše vedoucí z pravidla k diferenciálním rovnicím. Druhou kategorii lze popsat jako analýzu ovládání a stability a třetí kategorie je rezervovaná pro vše co nespadá pod první dvě.

Hlubší detaily k této problematice jsou popsány v publikaci Swarm robotics: a review from the swarm engineering perspective [1].

3.2 Hejnové

Hejnových algoritmů jsou desítky, jak již bylo zmíněno, většina z nich je inspirována chováním společenských zvířat. Mezi zajímavé zástupce rozhodně patří stochastické hledání, optimalizace hejnem částic a optimalizace mravenčí kolonií.

3.2.1 Stochastické hledání

Definice

V posledních letech vzrůstá zájem o distribuovaný způsob výpočtu využívající interakce mezi jednoduchými činidly (např. dále popsaná optimalizace částicových rojů, optimalizace kolonií apod.). Některé z těchto systémů inteligentních rojů byly přímo inspirovány sledováním interakcí mezi společenským hmyzem, jako jsou mravenci a včely. Na rozdíl od takových stigmatických komunikačních mechanismů je však Stochastic Diffusion Search (SDS) metaheuristickou inteligencí rojů (De Meyer et al., 2006). SDS využívá formu přímé komunikace mezi prostředky, která je analogičtější k mechanismu "tandemového volání" použitému určitým druhem mravenců zvané Leptothorax Acervorum [6].

SDS byla poprvé navržena Bishopem (1989) jako algoritmus pro optimální přizpůsobení vzoru založený na počtu obyvatel. Tyto algoritmy lze také přepracovat z hlediska optimalizace definováním cílové funkce *f (x), pro* hypotézu *x* o nejvhodnější umístění řešení, protože podobnost mezi cílovým vzorem a odpovídající oblastí na *pozice (x) ve* vyhledávacím prostoru a nalezení *x* tak, že *f (x) je* maximalizováno [6].

Obecně lze SDS nejsnadněji aplikovat na problémy s optimalizací, kdy je objektivní funkce rozložitelná do komponent, které lze vyhodnotit samostatně. Za účelem nalezení optima

dané cílové funkce SDS využívá roj *n* agentů, z nichž každá udržuje hypotézu *xi* o optimu. SDS algoritmus zahrnuje iteraci fází *TEST* a *DIFFUSION*, dokud roj agentů konverguje na optimální hypotézu [6].

Algoritmus

- *Inicializace*:
    - Výchozí hypotéza každého agenta je obvykle zvolena náhodně přes vyhledávací prostor, který může být podle pravděpodobných možných řešení omezen [6].
- *Zkouška:*
    - Pro každou látku v hypotéze zachování rojů *xi*, booleovská testovací funkce vrací hodnotu *TRUE*, pokud náhodně vybrané dílčí vyhodnocení cílové funkce v *xi* naznačuje "dobrou" hypotézu. Pokud testovací funkce vrátí hodnotu TRUE, činitel se stává aktivním členem roje, pokud ne, pak se stane pasivním členem. Testovací skóre pro danou hypotézu, *xi* je pravděpodobnost, že testovací funkce bude pravdivá a je tedy reprezentativní pro hodnotu objektivní funkce *f (xi)* [6].
- *Difuze*:
    - Hypotézy jsou komunikovány mezi agenty prostřednictvím vzájemné komunikace agentů. Ve standardním SDS komunikují pasivní agenti s náhodně zvoleným členem roje; Pokud je tento agent aktivní, komunikuje svou hypotézu s pasivním agentem, jinak pasivní agent znovu inicializuje novou hypotézu vybranou náhodně v celém vyhledávacím prostoru [6].
- *Konvergence:*
    - Jak iterace postupuje, posunují se i clustery agentů se stejnou hypotézou, dokud největší cluster agentů nedefinuje optimální řešení. Pro zjištění konvergence jsou dvě možná kritéria zastavení:

- kritérium silného zastavení, které po zjištění, že největší cluster agentů překračuje minimální práh, kontroluje, zda je velikost clusteru stochasticky stabilní v daném počtu iterací.
- Kritérium "slabého zastavení", které jednoduše kontroluje stabilitu a minimální velikost celkového počtu aktivních látek [6].

3.2.2 Optimalizace hejnem částic

Definice

Optimalizace hejnem částic (*Particle swarm optimization – PSO*) je populační stochastický přístup pro řešení nepřetržitých a diskrétních optimalizačních problémů.

V PSO se jednoduché softwarové prostředky, nazvané částice, pohybují ve vyhledávacím prostoru optimalizačního problému. Poloha částice představuje kandidátské řešení optimalizačního problému. Každá částice vyhledává lepší pozice ve vyhledávacím prostoru tím, že mění svou rychlost podle pravidel původně inspirovaných behaviorálními modely shlukování ptáků [7].

Každá částice má svou vlastní pozici a letové rychlosti, která se neustále mění v průběhu procesu optimalizace na základě následujících pravidel:

$$V_i^{P+1}\omega \cdot V_i^P + C_1 \cdot rand() \cdot (P_{bi}^{KP} - P_i^{KP}) + C_2 \cdot rand() \cdot \left(P_{gi}^{KP} - P_i^{KP}\right)$$
$$P_i^{KP} = P_i^{KP} + V_i^{KP} \qquad 2)$$

Kde V_{t+1} je aktualizovaná rychlost částic v příští iteraci, V_t je rychlost částic v aktuální iteraci, ω je inerční tlumič, který naznačuje dopad vlastních zkušenost částic na další pohyb, $C1 \cdot rand$ představuje jednotně distribuované číslo v intervalu [0, c1], které odráží, jak sousedící částice ovlivňují let částice, P_{bi}^{KP} je nejlepším místem sousedství, V_i^P je současná pozice částice, $C2 \cdot rand$ představuje jednotně rozložené číslo v intervalu [0, c2], který udává, důvěry částice v globální nejlepší pozice, P_{gi}^{KP} je globálně nejlepší pozice a V_i^{P+1} je aktualizovaná pozice částice. Pod vedením těchto dvou aktualizačních pravidel budou částice nuceny k tomu, aby se posunuly k nejlepší pozici, která byla dosud nalezena. To znamená, že v několika málo krocích, lze hledat optimální řešení. Tyto kroky jsou většinou inicializace, aktualizace rychlosti a aktualizace pozice [7], [8].

Varianty

Diskrétní PSO

Většina algoritmů pro optimalizaci rojů částic je navržena pro hledání v kontinuálních oblastech. Existuje však řada variant, které fungují v diskrétních prostorech. První variantou navrženou pro diskrétní domény byl algoritmus optimalizace binárních rojových částic (Kennedy a Eberhart 1997). V tomto algoritmu je poloha částice diskrétní, ale její rychlost je spojitá. *J-tá* komponenta vektoru rychlosti částic se používá k výpočtu pravděpodobnosti, s níž *j-tá* komponenta polohového vektoru částic má hodnotu 1. Rychlosti se aktualizují jako ve standardním algoritmu PSO, ale pozice se aktualizují pomocí následujícího pravidla:

$$x_{ij}^{t+1} = \begin{cases} 1 & když\, r\ < sig\left(v_{ij}^{t+1}\right) \\ 0 & jinde \end{cases} \qquad 3)$$

kde x_{ij} je *j-tá* složka polohového vektoru částice p_i, r je rovnoměrně distribuované náhodné číslo v intervalu [0,1] [7].

Holý PSO (Bare-bones PSO)

Bare-bones PSO (Kennedy 2003) je verze algoritmu optimalizace roji částic, ve kterém jsou pravidla pro aktualizaci rychlosti a polohy nahrazena postupem, který odebírá parametrickou funkci hustoty pravděpodobnosti. V algoritmu je pravidlo pro aktualizaci polohy částic v *j-té* komponentě $x_{ij}^{t+1} = N\ (\mu_{ij}^t, \sigma_{ij}^t)$, kde N je normální rozdělení s distribuční funkcí [7].

$$\mu_{ij}^t = \frac{b_{ij}^t + l_{ij}^t}{2}, \qquad \sigma_{ij}^t = \left|b_{ij}^t - l_{ij}^t\right| \qquad 4)$$

Plně informovaný PSO

Ve standardním optimalizačním algoritmu proti částicovým rojům je částice přitahována k nejlepším sousedům. Varianta, ve které částice využívá informace poskytnuté všemi jejími sousedy k aktualizaci své rychlosti, se nazývá plně informovaný roj částic (FIPS) (Mendes et al., 2004). Ve FIPS je pravidlo aktualizace rychlosti.

$$\vec{v}_i^{t+1} = w\vec{v}_i^{t+1} + \frac{\varphi}{|N_i|}\sum_{p_j \in N_i} W\left(\vec{b}_j^t\right)\vec{U}_j^t\left(\vec{b}_j^t - \vec{x}_i^t\right) \qquad 5)$$

Kde W je funkce, která váží příspěvek osobní polohy nejlepší částice k pohybu cílové částice na základě její relativní kvality [7].

3.2.3 Optimalizace mravenčí kolonií

V originále Ant Colony Optimization (ACO) je jeden z velmi užívaných zástupců metod, které jsou založeny na inteligenci hejna. Je slibnou metaheuristikou, se velmi početnými výzkumy věnovanými empirice a teoretickým analýzám. Inspirace pro jeho vývoj pochází z chování mravenců v kolonii. Pro nalezení nejkratší cesty od zdroje jídla do hnízda kolonie využívá mechanizmu pozitivních zpětných vazeb. Používají k tomu nepřímou komunikaci na základě fyzických změn prostředí, která je založena na zanechávání a detekci feromonových stop.

Uvnitř optimalizace kolonií mravenců je obecný kombinatorický problém optimalizace zakódován do omezeného problému nejkratší cesty. Ve stylu metody Monte Carlo se vytváří řada cest na základě pravděpodobnostního modelu, jehož parametry se nazývají umělý feromon - nebo jednodušší feromon. V metafoře o optimalizaci kolonií mravenců jsou tyto cesty konstruovány umělými mravenci, kteří chodí po grafu, ve kterém zakódován problém. Náklady na generované cesty se používají k modifikaci feromonu, což vede na vytváření pouze cest k slibným oblastem vyhledávacího prostoru [9].

Metaheuristika

V ACO vytvářejí umělý mravenci řešení kombinatorického problému optimalizace tím, že prolézají plně připojený konstrukční graf definovaný následujícím způsobem. Nejprve je každá instanční proměnná $X_i = v_i^j$ pojmenována jako složka řešení a označena c_{ij}. Soubor všech možných složek řešení je označen C. Pak je konstrukční graf $G_C(V, E)$ definován spojením složek C buď se soustavou vrcholů V, nebo se souborem okrajů E [10].

Hodnota feromonové stopy je spojena s každou složkou c_{ij} (hodnoty feromonu jsou obecně funkcí iteračního algoritmu $t : \tau_{ij} = \tau_{ij}(t)$). Hodnoty feromonu umožňují modelovat distribuci pravděpodobnosti různých složek řešení, dále se používají a aktualizují algoritmem ACO během vyhledávání [10].

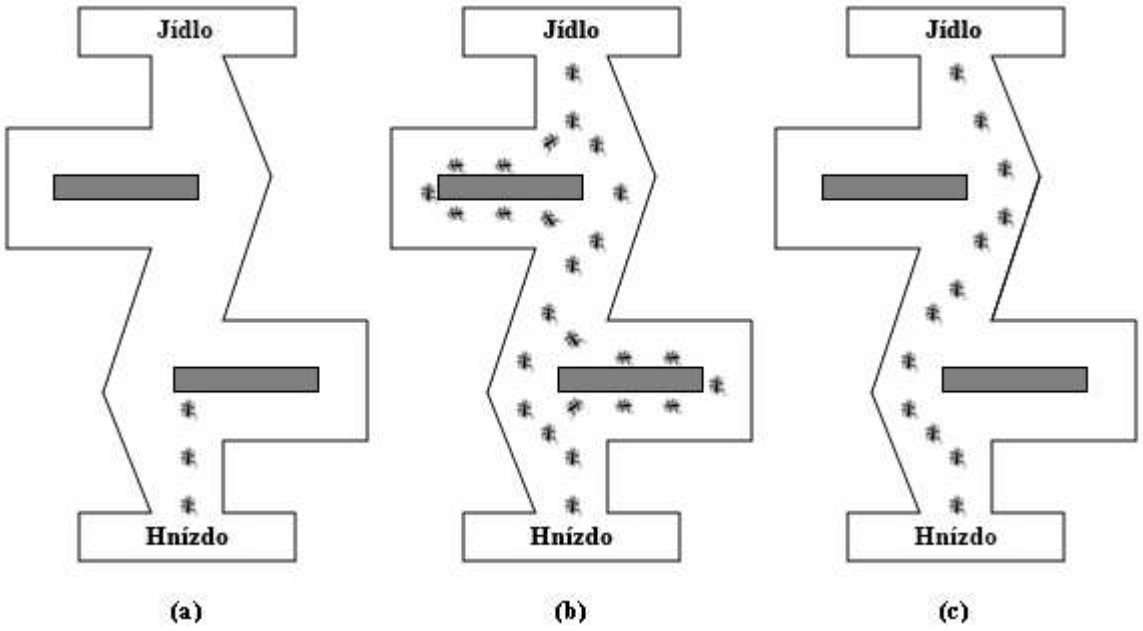
Mravenci se pohybují od vrcholu k vrcholu podél okrajů konstrukčního grafu za využití informací poskytovaných hodnotami feromonu a tím postupně vytvářejí řešení. Mravenci navíc na komponentech ukládají určité množství feromonu, to znamená buď na vrcholech, nebo na okrajích, které procházejí. Množství $\Delta\tau$ uloženého feromonu může záviset na kvalitě nalezeného řešení. Následující mravenci využívají informace o feromonech jako vodítko k slibnějším oblastem vyhledávacího prostoru [10].

Metaheuristika optimalizace mravenčí kolonií má následující strukturu:

1. Nastavení parametrů, inicializace feromonů
2. Naplánované Aktivity
   2.1. Sestroj Mravenčí Řešení
   2.2. Aktualizuj Feromon
3. Konec Naplánovaných Aktivit

Skládá se z inicializačního kroku a tří algoritmických komponent, jejichž aktivace je regulována konstrukcí Naplánované Aktivity. Tento konstrukt se opakuje, dokud není splněno kritérium ukončení. Typickými kritérii je maximální počet iterací nebo maximální doba výpočtů [10].

Konstrukce Naplánované Aktivity neurčuje, jak jsou tři algoritmické součásti naplánovány a synchronizovány. Ve většině aplikací ACO na NP-těžké problémy však tři algoritmické komponenty prochází smyčkou, která spočívá v konstrukci řešení všemi mravenci, (nepovinné) zlepšení těchto řešení pomocí místního vyhledávacího algoritmu a aktualizace feromonů. Tyto tři komponenty jsou nyní podrobněji vysvětleny v článku od Marco Dorigo [10].



Obrázek 7: Postup kolonie mravenců při hledání optimální cesty[11]

3.3 Algoritmy s globálním vstupem

Ovládání globálním vstupem znamená, že vydaný příkaz obdrží a provede každý robot. Tento způsob s sebou nese určité překážky, ale také značnou využitelnost. Takto ovládaní roboti jsou použitelní například při doručování léků na konkrétní místa nebo také ke snímání.

Zmíněné překážky se projevují hlavně při vyšších počtech robotů. V takovém případe se musí vyřešit dvě hlavní překážky. První překážkou se týká ovládání, druhá odhadu polohy velkých počtů robotů [12].

Ovládání velké populace

Zodpovězení otázek, čeho jsou a čeho nejsou roboti, kteří obdrží globální signál, schopni.

Odhadování polohy

Řeší se například pomocí počítačového vidění, kdy se snímá směr pohybu a úhel pohledu robota. Nicméně toto řešení se stává velice obtížným na úrovních mikro a nano-robotů. Jiný přístup pracuje pouze se startovní a koncovou pozicí populace, nebo s okraji prostoru, ve kterém se roboti pohybují.

3.3.1 Systémový model

Algoritmům pro ovládání robotů pomocí globálního vstupu se věnuje například Aaron T. Becker PhD., který na toto téma vydal i několik publikací. Příkladem systémového modelu pro velké populace mikro robotů je model publikovaný v článku Massive uniform manipulation: Controlling large populations of simple robots with a common input signal [12]. V následujících odstavcích je tento model popsán.

Architektura

Jsou porovnávány tři n-členné architektury s následujícími pohybovými modely na *obrázku 8*.

$$\begin{bmatrix}\dot{x}_i\\ \dot{y}_i\\ \dot{\theta}_i\end{bmatrix} = \underbrace{\delta_{ai}\begin{bmatrix}v\cos(\theta_i)\\ v\sin(\theta_i)\\ \omega\end{bmatrix}}_{\text{Adresovatelný}}, \underbrace{\begin{bmatrix}v\cos(\theta_i)\\ v\sin(\theta_i)\\ \epsilon_i\omega\end{bmatrix}}_{\text{Lokální}}, \underbrace{\begin{bmatrix}v\cos(\theta_i)\\ v\sin(\theta_i)\\ k\sin(\psi-\theta_i)\end{bmatrix}}_{\text{Globální}}$$

Obrázek 8: Pohybové modely [12]

Stav i-tého robota je $[x_i y_i \theta_i] \in R^3$a stav systému $\in R^{3n}$.

Adresovatelná kontrola

Tato architektura vyžaduje, aby byl každý robot adresovatelný. Existují v ní tři skalární vstupy, rychlost vpřed $v \in R$, úhlová otáčecí míra $\omega \in R$ a adresa $a \in \{1, \dots, n, all\}$. I-tý robot se pohybuje pouze tehdy, je-li jeho adresa *i* nebo *all.* Tento systém je plně kontrolovatelný, jelikož každý robot může být manipulován samostatně k cílové pozici a směrovému úhlu [12].

Lokální kontrola

V tomto případě má systém dvě skalární ovládací veličiny a to rychlost vpřed $v \in R$ a úhlovou míru otáčení $\omega \in R$. Všichni jsou ovladatelní, za předpokladu že každý robot má unikátní parametr $\in_i$, který mění míru otáčení [12].

Globální kontrola

Systém má dva skalární vstupy, rychlost vpřed $v \in R$ a chtěný směr $\psi \in [0,2\pi)$. Parametr *k* ovládá otáčecí příkaz. V prostoru bez překážek tento systém není kontrolovatelný. Roboti se pohybují a otáčejí v stejném tempu a tak je jejich koncová pozice výsledkem stejné přikázáním homogenní transformace na startovní pozici každého robota. Nicméně přidáním již jedné překážky je možné docílit rozbití symetrie a tím ovládat koncovou pozici každého robota [12].

Block World abstrakce

Pracovní plocha je čtvercová $m_1 \times m_2$ síť, v které každý čtverec je označen buď jako *volný, pevný* nebo *robot*. Všichni roboti jsou kontrolováni sdíleným příkazovým vstupem ze sady $\{\uparrow, \rightarrow, \leftarrow, \downarrow, \emptyset\}$, a mohou být pohybování vertikálně a horizontálně v síti pokud nenarazí na *pevný* čtverec, z kterých jsou také sestaveny hranice pracovní plochy.

Čištění čtvercového prostoru

Vyčištění oblasti *A* s využitím jediné čtvercové překážky, sestavené z *pevných* čtverců. Procedura tohoto čištění sestává z pohybu robotů v oblasti *A* a jejich posunu sem a tam přes překážku, dokud se oblast *A* nevyčistí.

Je dáno:

- Oblast k vyčištění s levým spodním rohem v $[A_x, A_y]$, šířkou A_w a výškou A_h
- Překážka s levým spodním rohem $[O_x, O_y]$, šířkou O_w a výškou O_h

```
CLEARREGION(A, O)
 1: move ← O_x − A_x          ▷ přesuň překážku vlevo dolů
 2: move ↓ O_y − A_y
 3: c = 0
 4: while c · O_w < A_w do
 5:     move ↓ A_h            ▷ vyčisti sloupec dolů
 6:     move ← O_w
 7:     c = c + 1
 8:     if c · O_w < A_w then
 9:         move ↑ A_h        ▷ vyčisti sloupec nahoru
10:         move ← O_w
11:         c = c + 1
12:     end if
13: end while
14: move → A_x − O_x − c · O_w  ▷ přesuň vyčištěnou oblast do A
15: move ↑ A_y − O_y
```

Obrázek 9: Algoritmus pro vyčištění oblasti [12]

Algoritmus popsaný na *obrázku 9* vyžaduje proporcionální oblast vzhledem k *A* a *O:*

$$(|A_x - O_x| + A_w) \times \left(\left|\left(A_y + A_h\right) - \left(O_y + O_h\right)\right| + A_h\right) \qquad 6)$$

Ovládání pozice

Tato část popisuje algoritmus pro ovládání pozice *n* robotů použitím jedné překážky. Je zde využita Block World abstrakce, ve které jsou roboti a překážka jednotkovými čtverci. Každé zavolání tohoto algoritmu posune jednoho robota ze startovní pozice na pozici cílovou [12].

Deklarace

Startovní pozice k-tého robota je ve světových souřadnicích $k^W(0)$, jeho cílová pozice k^W_{goal} a jeho pozice v čase t je $k^W(t)$. Jsou definovány osově srovnané krychle *S* a *F* s pevnými rozměry $k^W(0) \in S^W(0)$ a $k^W_{goal} \in F^W(0)\ \forall k \in [1, n]$. Levé spodní rohy S a F jsou $\left[S^W_x(t), S^W_y(t)\right]$ a $\left[F^W_x(t), F^W_y(t)\right]$. Jejich šířka jsou S_w, F_w a výšky S_h, F_h. Jelikož jsou všichni roboti identičtí jejich indexace je zleva doprava, shora dolů v oblasti S a shora dolů, zleva doprava v oblasti F. Je nutné poznamenat, že pozice k-tého robota může být určena v rámci $k^W(t) = F^W(t) + k^F(t)$. Nehybná překážka je situována v $\left[O^W_x, O^W_y\right]$. Předpokládá se, že pozice překážky $O_{x,y}$, startovní pozice $S_{x,y}$ a cílová pozice $F_{x,y}$ nejsou na stejné pozici a jsou situována jako na *obrázku X* [12].

Procedura

PositionControl(S, F, O, k)

1: move $\uparrow$ until $S^W_y(t) > O^W_y$
2: move $\leftarrow$ until $k^W_x(t) = O^W_x$
3: move $\downarrow$ until $k^W_y(t) > S^W_y(t) + S_h$
4: move $\uparrow$ until $S^W_y(t) > O^W_y$
5: move $\leftarrow$ until $S^W_x(t) < O^W_y - S_w$
6: move $\downarrow$ until $k^W_y(t) = O^W_y$
7: move $\rightarrow$ until $k^W_x(t) = F_{\text{goal},x} + k^F_{\text{goal},x}$
8: move $\uparrow$ 1
9: move $\rightarrow$ 1
10: move $\downarrow$ until $k^W_y(t) = F^W_{\text{goal},y} + k^F_{\text{goal},y}$
11: move $\uparrow$ until $F^W_y(t) > O^W_y$
12: move $\leftarrow$ until $F^W_x(t) < O^W_x - F_w$

Obrázek 10: Algoritmus pro ovládání pohybu

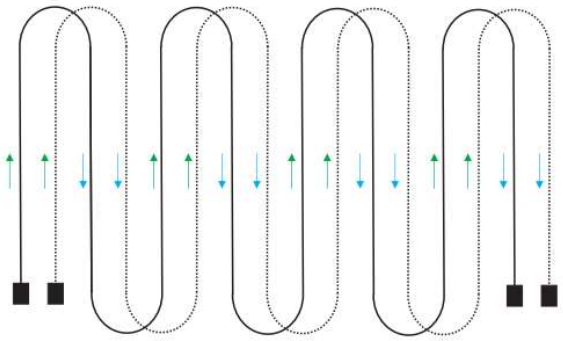
Tento algoritmus, popsaný *obrázkem 10*, potřebuje $S_w + F_w + 1$ volného prostoru vlevo, $S_w + F_w$ volného prostoru vpravo a $S_h + F_h + 1$ nad a $S_h + F_h$ pod překážkou.

4 Popis řízeného systému

V následujících kapitolách je popsán systém, včetně fyzikálního popisu systému, pravidel omezujících pohyb systému a popis prototypů tohoto systému.

4.1 Fyzikální popis systému

Systém, jehož simulace je předmětem této práce, je sestaven z robotů pohybujících se na desce. Kdy robot je tvořen permanentním magnetem, který nenese žádný aktivní prvek. Veškeré ovládání je zajišťováno vlivem změny magnetického pole cívek v desce. Tyto cívky jsou uspořádány ve dvou o 90° pootočených vrstvách, přičemž v jedné vrstvě jsou dvě prostorově posunuté cívky dle *obrázku 11*.



Obrázek 11: grafické zobrazení rozloženého provedení cívek

Jelikož jsou roboti tvořeny permanentními magnety, mají kolem sebe inherentně magnetické pole odpovídající jejich rozměrům. Postupným spínáním jednotlivých ovládacích obvodů závitů cívek se kolem nich vytváří nestacionární magnetické pole, které ovlivňuje magnetické pole robotů. Sledem spínání těchto cívek se tímto způsobem vytváří magnetická síla nutící robota k pohybu.

Tento pohyb se dá popsat využitím příkladu vložení závitu cívky do magnetického pole. Vlivem tohoto pole se závit cívky natočí podle polarity magnetického pole, do kterého byl vložen. Tento příklad je vlastně inverzním příkladem toho, jak funguje pohyb v prototypech této práce. V tomto případě se pohybuje magnet, jehož pole působí na cívku, kdežto cívka je stacionární.

Pohyb robota lze popsat i matematickým modelem, který vychází pohybové rovnice 7. Kde m je hmotnost, s dráha, t čas a F síla. Jelikož derivací dráhy za čas je rychlost, můžeme tuto rovnici přepsat do tvaru rovnice 8, kde v je rychlost. Výsledná síla je poté vektorovým součtem sil třecích a sil vyvolaných polem cívky ($F_{mx} + F_{my}$), jak popisuje rovnice 9 [13].

$$m\frac{d^2s}{dt^2} = F \tag{7}$$

$$m\frac{dv}{dt} = F \leftarrow v = \frac{ds}{dt} \tag{8}$$

$$F = F_{mx} + F_{my} + F_t \tag{9}$$

$$F_t = k \cdot m \cdot g \tag{10}$$

Průběhy těchto sil popisuje obrázek 11, kde Ω_1 je klaně napájený vodič, Ω_2 je vodič bez napájení, Ω_3 je záporně napájený vodič, F_m Složka síly působící na magnet od cívky, F_t je třecí síla respektující nadzvedávání robota magnetickým polem cívek, F_{mmax} je maximální hodnota síly působící na magnet od cívky, F_{tmax} je maximální třecí síla F_{tmin} je minimální třecí síla [13].

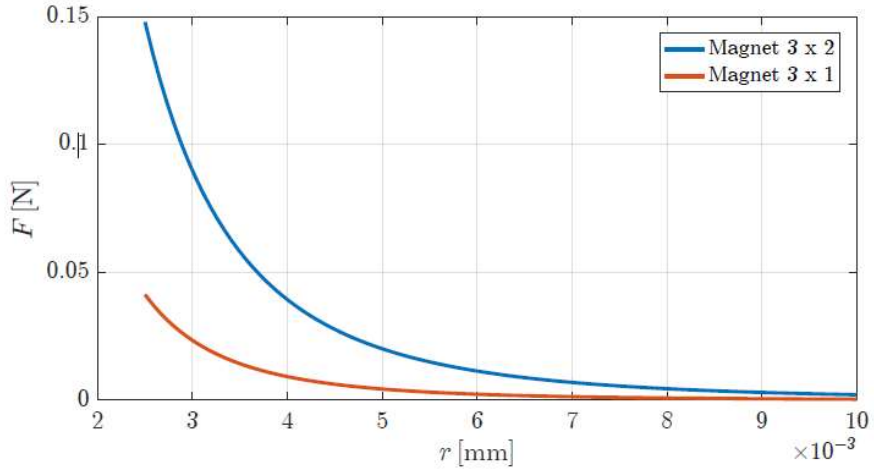
4.2 Pravidla řízeného systému

Simulace popisovaného systému je omezena určitými pravidly, která jsou založena na fyzikálních vlastnostech robotů i desky. Další pravidla jsou zavedena pro jednoduchost simulace, za účelem dosažení vyšší názornosti.

Mezi pravidla odvíjející se od fyzikálních předpokladů patří pravidlo o nemožném překrývání robotů. Tím je myšleno, že není možné, aby řekněme při pohybu systému doleva v případě, že první robot narazí do přepážky, se druhý robot nemůže vtlačit na jeho místo. Toto pravidlo vede tedy k vytváření front na hranách desky nebo překážky.

Pravidlo zavedené pro zjednodušení simulace je nemožnost překročení hranic desky robotem. Pro simulaci několika desek spojených do určité matice je ovšem možné použít desku o rozměrech, které by měla tato soustava jako jediná deska.

Systémovým pravidlem, které v simulaci je zahrnuto pouze v ohledu prvního pravidla, je vzájemné ovlivňování robotů mezi sebou. Tento jev je nicméně velice zajímavý a teoreticky využitelný i pro určité úkony systému. Ovlivňováním robota jiným robotem se v tomto případě rozumí vzájemné odtahování/přitahování se, jelikož roboti jsou tvořeni permanentními magnety, které kolem sebe mají inherentně magnetické pole. Využitelnost tohoto jevu je například posun těžších objektů. V případě, že robot táhnoucí objekt sám nemá dostatečnou sílu, je možné využít magnetického pole dalšího robota jako pomoc. Tato teorie byla i ověřena na prototypu MagStriver a síly působící mezi roboty jsou zobrazeny na *obrázku 12*.



Obrázek 12: Graf síly působící mezi dvěma magnety v závislosti na jejich vzájemné vzdálenosti[13]

4.3 Popis prototypu řízeného systému

Předmětem této kapitoly je zevrubný popis prototypů. Tento model byl vytvořen na Katedře teoretické elektroniky a následující odstavce jsou parafrází diplomové práce Bc. Jiřího Kuthana [13], kde jsou zmíněné prototypy popsány detailněji.

4.3.1 Mikroroboti

Mikroroboti jsou v tomto případě vytvořeny z permanentních magnetů, a to konkrétně ze slitiny neodym-železo-bor. V následujících odstavcích je rozlišení těchto magnetů do dvou skupin a sice monomagnet a vícemagnet. Jak již napovídá označení skupin, jde o rozdělení podle počtu použitých magnetů z výše uvedené slitiny v jednom mikrorobotovi.

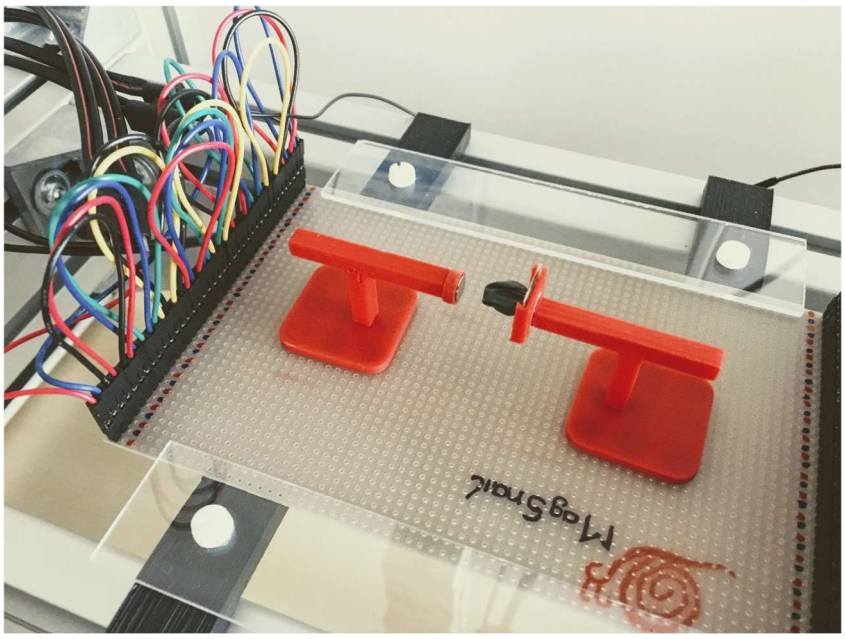
Jak zařízení MagSnail tak MagStriver, jsou blíže popsány v práci Bc. Jiřího Kuthana [13], včetně postupu jejich konstrukce a ovládání spínacích cívek. Následují odstavce, poskytují pouze zevrubný popis a zhodnocení těchto dvou experimentálních provedení.

4.3.2 MAGSNAIL

MagSnail, tedy první prototyp experimentálního zařízení dostal své jméno nejen díky rychlosti pohybu robotů na tomto zařízení, ale i díky způsobu pohybu, který byl šnekem inspirován. Tento prototyp byl sestaven pro ověření základní myšlenky, tedy využití změn elektromagnetického pole k pohybu permanentních magnetů. Polohování na tomto prototypu je možné pouze s jedním stupněm volnosti. Navíc se díky jeho testování podařilo odhalit několik nedostatků.

Mezi tyto nedostatky se řadí například překmitávání robota mezi jednotlivými kroky nebo nestabilita robota, při menších hodnotách proudu. Dalším nedostatkem se projevilo pootáčení robota, které jej vyřadilo z provozu.

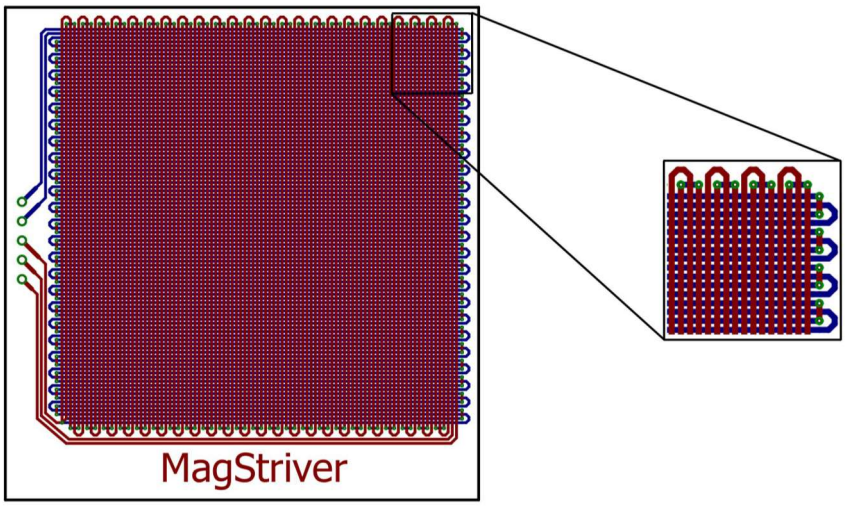
MagSnail byl vyroben z pospojované univerzální desky, tak jak lze odvodit z *obrázku 13.*



Obrázek 13: Platforma MagSnail s roboty vybavené magnetickým úchopovým systémem[13]

4.3.3 MAGSTRIVER,

Název druhého prototypu je odvozen z anglického slova strive, které lze přeložit jako dřít. Návrh MagStriveru vychází z nedostatků MagSnailu. Hlavním cílem při jeho vývoji tedy bylo zmenšení desky, zjemnění kroku a snížení chybovosti za použití co nejmenších proudů a napětí, což vedlo k návrhu zobrazeného na *obrázku 14*.



Obrázek 14: Návrh desky plošných spojů MagStriver[13]

4.4 Simulace

V následujících kapitolách je popsán postup tvorby simulace výše popsaného systému. Kapitoly jsou rozděleny do tří celků, kdy první celek slouží k formulaci problému, druhý na algoritmizaci a poslední se zabývá hotovou simulací a jejím zhodnocením.

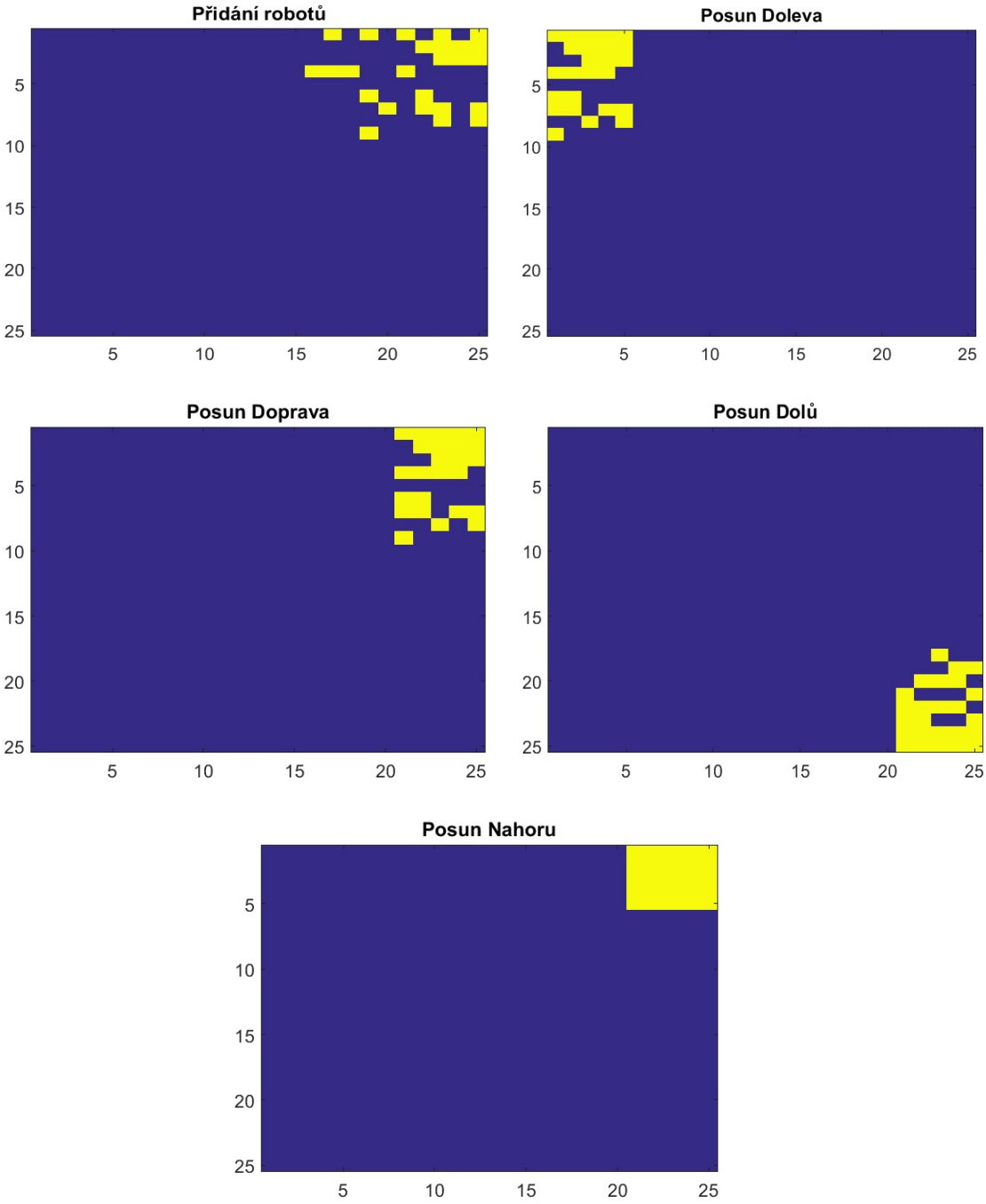
4.4.1 Popis problému

Pro provedení simulace je nutné znát několik vstupních parametrů. Konkrétně pro tuto práci se jedná o omezující pravidla a vytyčený cíl. Cílem této simulace se rozumí možnost zkoušení různých algoritmů pro ovládaný systém a jejich vizualizace.

4.4.2 Obecný popis

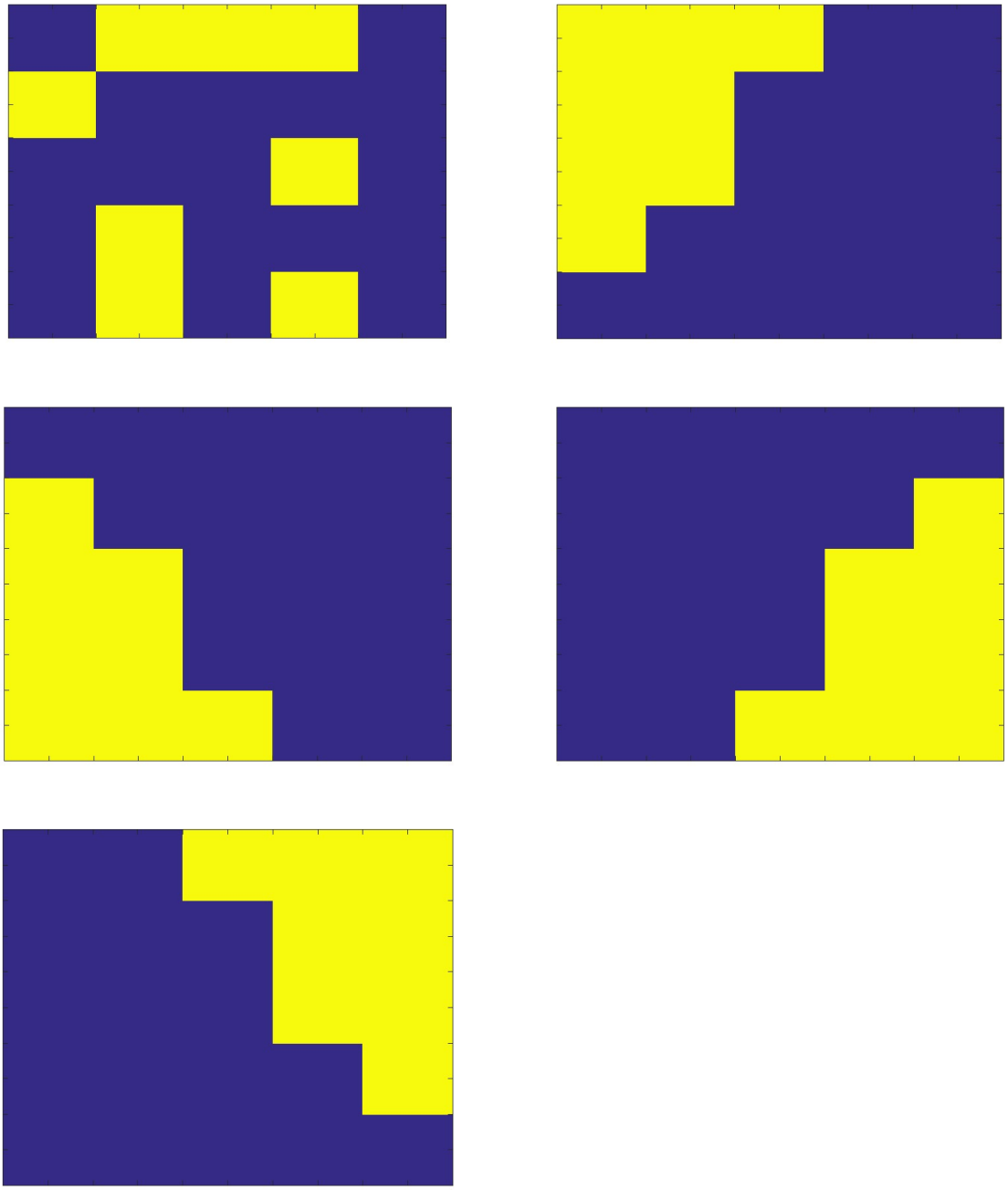
Simulace je napsána v programovacím prostředí Matlab. Její tvorba je složena z několika malých snadno dosažitelných kroků, vedoucích k finální formě, která je přiložena k této práci.

Prvním takovým krokem je v tomto případě jednotlivý pohyb všech robotů v jednom určeném směru. Tento krok je samozřejmě proveden pro všechny čtyři, tím systémem proveditelné, směry pohybu. Na *obrázku 15* je znázorněn zmíněný pohyb do všech čtyř stran. Přičemž v prvním obrázku je náhodně vygenerované pole robotů a na dalších obrázcích je znázorněn pohyb těmito vygenerovanými roboty do všech respektovaných směrů, tedy doleva, doprava, nahoru a samozřejmě i dolů. Velikost pole není v tomto případě nijak limitována.



Obrázek 15: (1) Pole naplněné náhodným počtem robotů
(2) Posun robotů doleva (3) Posun robotů doprava
(4) Posun robotů dolů (5) Posun robotů nahoru

V následujícím kroku byli tyto pohyby sestaveny za sebe a uzavřeny do pohybu všech jednotek nejprve do levého horního rohu a poté pohyb proti hodinovým ručičkám. Proces posunu proti hodinovým ručičkám je znázorněn v příloze A a na *obrázku 16*.



Obrázek 16: (1) Pole naplněné náhodným počtem robotů
(2) Posun robotů doleva (3) Posun robotů dolů
(4) Posun robotů doprava (5) Posun robotů nahoru

Tímto krokem lze ověřit funkčnost jednotlivých skript pro pohyb a jejich následného vykreslování. Funkční skript lze pak použít pro aplikaci různých algoritmů hejnového charakteru.

4.4.3 Použité algoritmy

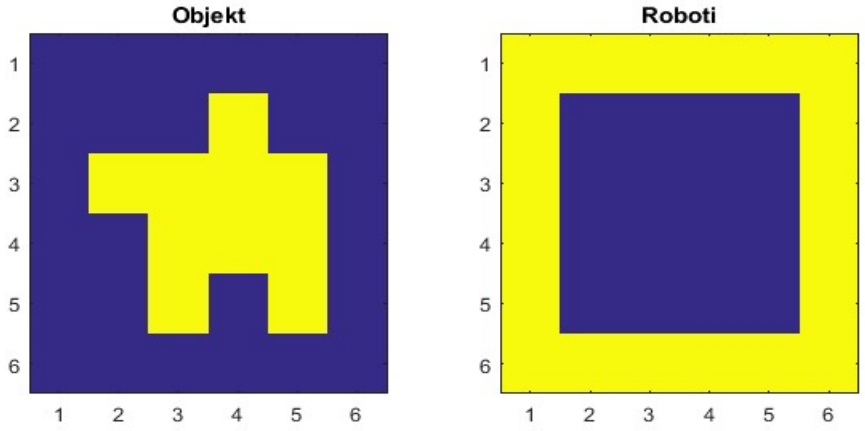
Jako ukázkový příklad je v této práci zvolen algoritmus, jehož úkolem je, za použití pohybu robotů ve výše zmíněných směrech, zjistit tvar objektu a algoritmus pro ovládání robotů pomocí překážek jak nástroje.

Tyto algoritmy jsou opět vytvořeny v prostředí MATLAB. Provedená simulace využívá poznatků získaných z výše popsaných algoritmů pro pohyb doleva, doprava, nahoru a dolů.

4.4.4 Rozpoznávání tvaru objektu

Algoritmus pro zjištění tvaru neznámého objektu, jehož skript je v příloze D, je také vytvořen v prostředí MATLAB. Pro účel tohoto algoritmu je vytvořen objekt, který je pro použité roboty neviditelný. Jako ukázkový objekt byl zvolen objekt s tvarem vyobrazeným na *obrázku 5*. Tento tvar byl zvolen pro ověření zjišťování tvaru všemi čtyřmi pohyby.

Roboti, vytvoření pro zjištění tvaru jsou situováni po celém obvodu virtuální desky, tak jak ukazuje *obrázek 17*. Proces zjišťování začíná s pohybem robotů vlevo. Když robot narazí na překážku, tedy hranu objektu, nemůže se hnout dál a tím se zaznamená hrana do paměti. Obdobným způsobem pokračuje hledání hran pohybem doprava, poté pohybem dolů a nakonec i nahoru.

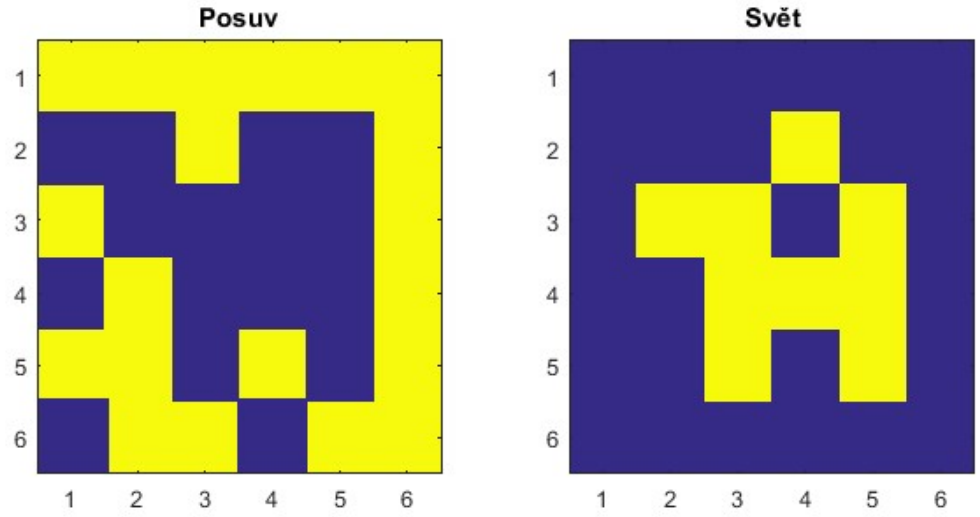


Obrázek 17: Vizualizace objektu a startovní pozice robotů

Veškeré zaznamenané hrany se ukládají do oblasti, která byla nazvána *svět*. V této oblasti se při každém kroku objevuje nová hrana, v případě, že na ní některý robotů narazí.

Vizualizace postupu tohoto algoritmu, s výše zmíněným objektem, je přiložena k práci v příloze E.

Konečný krok tohoto algoritmu je vyobrazen na *obrázku 18*, z něhož je patrné, že algoritmus uspěl v nalezení všech hran.

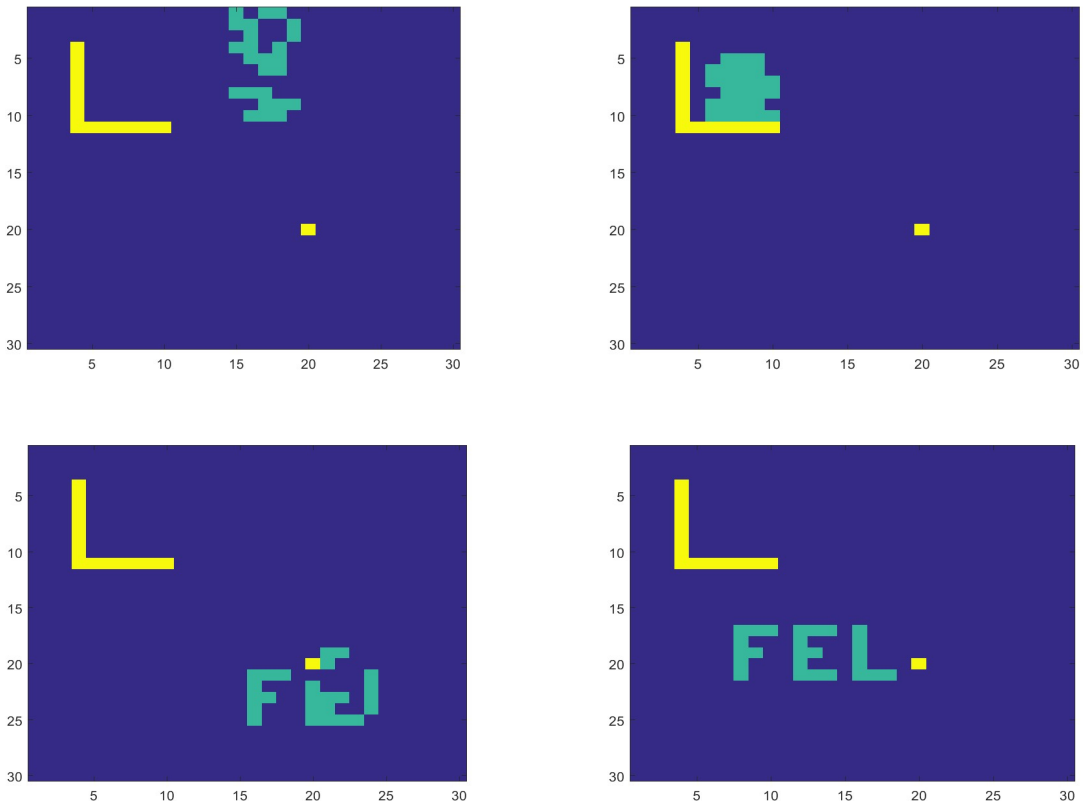


Obrázek 18: Vizualizace nalezených hran a koncové polohy robotů

4.4.5 Využití překážek jako ovládacích prvků

Další algoritmus využívá poznatků z předchozích algoritmů ve spojení s modelem A.T. Beckera [12], který je popsán výše v kapitole 3.2.1. Tento algoritmus využívá statických překážek k ovládání hejna robotů, jedním globálním vstupem. Jako ukázkový příklad toho přístupu k paralelnímu ovládání několika robotů byla vytvořena zkratka „FEL“. Tento algoritmus je přiložen k práci v příloze F.

Algoritmus začíná vygenerováním náhodného hejna robotů. V tomto případě o velikosti 25 členů, jelikož předem víme, kolik robotů je potřeba pro vytvoření cílové formace. Dalším krokem je přesun robotů do překážky ve tvaru písmena „L“ kde se nárazy do překážky, díky platnosti pravidla o tvoření front zmíněného výše, shromáždí. Po shromáždění robotů do skupiny se přesunou do blízkosti další překážky. Tato překážka je o velikosti jednoho robota, tento rozměr je klíčový pro úspěšné provedení rekonfigurace robotů. Opětnými nárazy do překážky a trvající platnosti pravidla o tvoření front, je možné vytvořit obrazec ve tvaru zkratky „FEL“. Obrázek 19 demonstruje několik ukázkových kroků tohoto procesu.



Obrázek 19: Demonstrace kroků algoritmu využití překážek k ovládání
(1) Iniciační krok – vygenerování 25 robotů
(2) Shromáždění robotů do lépe ovladatelné skupiny
(3) Několikátý krok při tvorbě obrazce
(4) Hotový obrazec „FEL“

4.4.6 Zhodnocení výsledků a možné zlepšení

Provedená simulace úspěšně zaznamenala veškeré hrany zadaného objektu. Nevýhodou tohoto algoritmu je fakt, že dochází pouze k rozpoznání hran objektu, tudíž není možné s jistotou určit, zda je objekt zmapovaný celý. Naproti tomu, rozpoznávání hran probíhá rychle, což se dá připsat velkému počtu podílejících se robotů.

Mezi možná zlepšení rozhodně patří užití menšího počtu robotů ke splnění tohoto úkolu. Dále by se dala zlepšit vizualizace výsledků, které jsou pro tuto chvíli řešeny pouze příkazem *spy*. Nicméně vypovídací schopnost je i takto na dostatečné úrovni.

Jako slibný se jeví způsob využití překážek jako ovládacích prvků robotů ovládaných jedním globálním vstupem. V této práci je tento způsob sice použit, ale je pouze ovladatelný uživatelem. Velkým přínosem by byla implementace umělé inteligence, která by roboty za použití překážek posunula do určené formace sama, bez dalších příkazů od uživatel.

5 Závěr

Práce popisuje ve svých dvou hlavních částí teoretická fakta a praktickou část.

V teoretické části bylo popsáno, kromě základních pojmů i několik algoritmy pro ovládání hejn a paralelního polohování robotů. V praktické části byl popsán způsob ovládání robotů z permanentních magnetů za pomoci aplikace magnetického pole, omezující podmínky z toho vyplývající a dále simulace, která je předmětem této práce. Simulace byla popsána a zhodnocena.

V teoretické části jsou popsány hejnové algoritmy, nicméně v simulaci není hejnový algoritmus aplikovaný přímo, všechny algoritmy přiložené k práci v m-file formátech jsou hejnovými algoritmy pouze inspirovány. Jedná se o předem určené činnosti v předem určeném sledu.

Simulace systému je v práci provedena pomocí několika algoritmů. Konkrétně jednotlivých posuvů do čtyř směrů, kde je na uvedeném příkladu vidět pravidlo o tvoření front v praxi. Dále byla provedena rotace proti směru hodinových ručiček. Poté bylo provedeno zjišťování tvaru objektu vloženého do pole obklopeného roboty. Tento algoritmus je založen na narážení robotů do objektu, přičemž svou nehybností u hrany objektu, tuto hranu ukládají do paměti. Roboti jsou nastaveni tak, že při každém pohybu projdou desku do poloviny, tudíž se nemůže stát, že by se nějaká hrana nedetekovala. Poslední provedenou simulací je algoritmus demonstrující možnost ovládání robotů pomocí překážek, kdy lze roboty seskupovat do snadněji ovladatelných skupin, pomocí překážek určitých tvarů a rozměrů, nebo naopak separaci těchto skupin do menších.

Pro další vývoj prototypů je třeba provést další simulace, například více využívajících možnosti spolupráce několika robotů na daném úkolu. Toho může být docíleno jak implementací jakéhosi instinktu do robotů, čímž by se jejich paralelní ovladatelnost výrazně zjednodušila, nebo využitím formy umělé inteligence pro řízení hejna robotů za použití překážek jako ovládacích prvků.

Seznam použité literatury

[1] M. Brambilla, E. Ferrante, a M. Birattari, „Swarm robotics : a review from the swarm engineering", s. 1–41, 2013.

[2] C. R. Kube a H. Zhang, „Collective Robotics : From Social Insects to Robots 1 Introduction 2 Social Insects", 1993.

[3] J. Tharin, *Kilobot*, č. August. 2013.

[4] M. Le Goc, L. H. Kim, A. Parsaei, J.-D. Fekete, P. Dragicevic, a S. Follmer, „Zooids: Building Blocks for Swarm User Interfaces", *Proc. 29th Annu. Symp. User Interface Softw. Technol. - UIST '16*, s. 97–109, 2016.

[5] D. R. Frutiger, B. E. Kratochvil, K. Vollmers, a B. J. Nelson, „Magmites - Wireless resonant magnetic microrobots", *Proc. - IEEE Int. Conf. Robot. Autom.*, s. 1770–1771, 2008.

[6] J. Bishop, „Stochastic diffusion search", *Scholarpedia*, roč. 2, č. 8, s. 3101, 2007.

[7] M. Dorigo, M. Oca, a A. Engelbrecht, „Particle swarm optimization", *Scholarpedia*, roč. 3, č. 11, s. 1486, 2008.

[8] G. Giftson Samuel a C. Christober Asir Rajan, „Hybrid: Particle Swarm Optimization-Genetic Algorithm and Particle Swarm Optimization-Shuffled Frog Leaping Algorithm for long-term generator maintenance scheduling", *Int. J. Electr. Power Energy Syst.*, roč. 65, č. February, s. 432–442, 2015.

[9] M. Birattari, P. Pellegrini, a M. Dorigo, „On the invariance of ant colony optimization", *IEEE Trans. Evol. Comput.*, roč. 11, č. 6, s. 732–742, 2007.

[10] M. Dorigo, „Ant colony optimization", *Scholarpedia*, roč. 2, č. 3, s. 1461, 2007.

[11] B. Bachir, A. Ali, a M. Abdellah, „Multiobjective Optimization of an Operational Amplifier by the Ant Colony Optimisation Algorithm", *Electr. Electron. Eng.*, roč. 2, č. 4, s. 230–235, srp. 2012.

[12] A. Becker, G. Habibi, J. Werfel, M. Rubenstein, a J. McLurkin, „Massive uniform manipulation: Controlling large populations of simple robots with a common input signal", *IEEE Int. Conf. Intell. Robot. Syst.*, s. 520–527, 2013.

[13] J. B. Kuthan, „Elektromagnetický systém pro polohování magnetických těles", Západočeské univerzita, 2017.

Přílohy

Příloha A – skripty jednotlivých posunů

Pohyb doleva

```
function [ matrix ] = posun_doleva ( matrix )
help_matrix = zeros( size ( matrix ) );
j = 1; i = 1;
for ( i = 1 : size( matrix , 1 ) )
    if ( matrix ( i , j ) == 1)
        help_matrix ( i , j ) = 1;
        for ( j = 2 : ( size( matrix , 1 ) ) )
            if( ( matrix ( i , ( j - 1) ) == 0 ) && ( matrix ( i , j  ) ==
1 ) )
                help_matrix ( i , j ) = matrix ( i , ( j - 1 ) );
                help_matrix ( i , ( j - 1 ) ) = matrix ( i , j );
            else
                help_matrix ( i , j ) = matrix ( i , j );
            end
        end
    else
        help_row = matrix(i,:);
        help_row ( : , 1 ) = [];
        help_matrix ( i , : ) = [ (help_row ( 1 , : )) 0 ];
    end
    j=1;
end
     matrix = help_matrix;
end
```

Pohyb doprava

```
function [ matrix ] = posun_doprava ( matrix )
help_matrix = zeros( size( matrix ) );
j = size( matrix , 1 ); i = 1;
for ( i = 1 : size( matrix , 1 ) )
    if ( matrix ( i , j ) == 1)
        help_matrix ( i , j ) = 1;
        for ( j = ( size( matrix , 1 ) - 1 ) : -1 : 2 )
            if ( ( matrix ( i , ( j - 1 ) ) == 1 ) && ( matrix ( i , j ) ==
0 ) )
                help_matrix ( i , j  ) = 1;
                help_matrix ( i , ( j - 1 ) ) = 0;
            else
                help_matrix ( i , j ) = matrix ( i , j );
            end
        end
    else
        help_row = matrix ( i , : );
        help_row ( : , size( matrix , 1 ) ) = [];
        help_matrix ( i , : ) = [ 0 (help_row ( 1 , : )) ];
    end
    j=size( matrix , 1 );
end
    matrix = help_matrix;

end
```

Pohyb nahoru

```
function [ matrix ] = posun_nahoru ( matrix )
help_matrix = zeros( size( matrix ) ); i = 1;
done = zeros ( 1 , size( matrix , 2 ) );
help_column = [ 1 ; size( matrix , 1 ) ];
help_row = zeros ( 1 , size( matrix , 1 ) );
matrix_zeros = zeros( 1 + size( matrix , 1 ) , size( matrix , 1 ) );
matrix_zeros = [ matrix ; help_row ];
for j = 1 : size( matrix , 2 )
    if ( matrix ( i , j ) == 1 && done ( 1 , j ) ~= 1 )
        help_matrix ( i , j ) = 1;
        for i = ( size( matrix , 1 ) : -1 : 2 )
            if ( matrix ( i , j ) == 0 && help_matrix ( i , j ) ~= 1 )
                help_matrix ( i , j ) = 0;
            elseif ( matrix ( i , j ) == 1 && matrix ( ( i - 1 ) , j ) == 0
&& help_matrix ( i , j ) ~= 1 )
                help_matrix ( ( i - 1 ) , j ) = matrix ( i , j );
                help_matrix ( i , j ) = matrix ( ( i - 1 ) , j );
            else
                help_matrix ( i , j ) = 1 ;
            end
        end
    else
        help_column = matrix_zeros ( : , j );
        help_column ( 1 , : ) = [];
        help_matrix ( : , j ) = help_column;
        done ( 1 , j ) = 1;
    end
    i = 1 ;
end
matrix = help_matrix;
end
```

Pohyb dolu

```
function [ matrix ] = posun_dolu ( matrix )
help_matrix = zeros( size( matrix ) );
i = size( matrix , 1 ); j = size( matrix , 1 );
done = zeros ( 1 , size( matrix , 1 ) );
for ( i = size( matrix , 1 ) : -1 : 1 )
  for ( j = size( matrix , 1 ) : -1 : 1 )
    if ( matrix ( i , j ) == 1 && 1 ~= done ( 1 , j ) )
      help_matrix ( i , j ) = 1;
      if ( ( i + 1 ) <= ( size( matrix , 1 ) ) && matrix ( ( i + 1 ) , j )
    == 0 )
        help_matrix ( i , j ) = matrix ( ( i + 1 ) , j );
        help_matrix ( ( i + 1 ) , j ) = matrix ( i , j );
      else
        help_matrix ( i , j ) = matrix ( i , j );
      end
    elseif ( ( i == size( matrix , 1 ) ) && matrix ( i , j ) == 0 )
      done ( 1 , j )= 1;
      help_row = zeros( 1 , size( matrix , 1 ) );
      A_zeros = [ help_row ; matrix ];
      help_column = A_zeros ( : , j );
      help_column ( ( size( matrix , 1 ) + 1) , : ) = [];
      help_matrix ( : , j ) = help_column;
    end
  end
    end
```

```
    matrix = help_matrix;
end
```

Příloha B – skript rotace proti směru hodinových ručiček

Pohyb proti směru hodinových ručiček

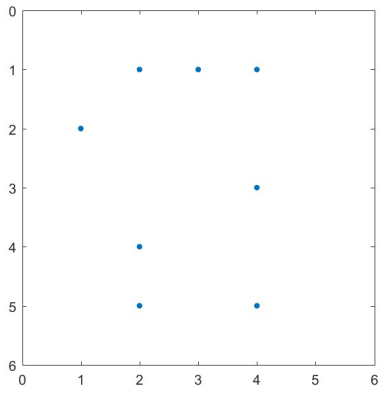
```
clear all
clc

n = 5 ; m = 5 ;
A1 = rand(n,m) < 0.5 ;
A2 = round(rand(n,m));

N0 = ceil(2*n*m/3) ;
A = ones(n,m) ;
A(1:N0) = 0 ;
A(randperm(numel(A))) = A;
done = 0 ;
count = 1;

% Seřazení do levého horního rohu
while done == 0
    for i = 1 : size(A,1)
        for j = 2 : size(A,2)
            if ( A( i , j - 1 ) == 0 && A( i , j ) == 1 )
                A = posun_zobrazeni ( A , [ 3 ] , count);
                count = count + 1;
            else
                done = 1;
            end
        end
    end
    i = 1;
    j = 1;
end

pause(1)

while done == 1
    for i = 1 : ( size(A,1) - 1 )
        for j = 1 : size(A,2)
            if ( A( i , j ) == 0 && A( (i + 1) , j ) == 1 )
                A = posun_zobrazeni ( A , [ 1 ] , count);
                count = count + 1;
            elseif ( A( i , j ) == 1 && A( (i + 1) , j ) == 0 )
                A = posun_zobrazeni ( A , [ 1 ] , count);
                count = count + 1;
            else
                done = 0;
            end
        end
    end
    i = 1;
    j = 1;
end
step = 0 ;
```

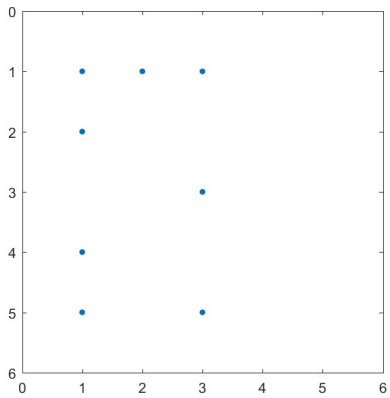
```
% Rotace po směru hodinových ručiček
while step <= 4
    for i = size(A,1) : -1 : 2
        for j = 1 : size(A,2)
            if ( A( i , j ) == 0 && A( (i - 1) , j ) == 1 )
                A = posun_zobrazeni ( A , [ 2 ] , count);
                count = count + 1;
            elseif ( A( i , j ) == 1 && A( (i - 1) , j ) == 0 )
                A = posun_zobrazeni ( A , [ 2 ] , count);
                count = count + 1;
            else
                done = 0;
            end
        end
    end
    i = 1;
    j = 1;
    step = step + 1;
    for i = 1 : size(A,1)
        for j = 1 : (size(A,2)-1)
            if ( A( i , j + 1 ) == 0 && A( i , j ) == 1 )
                A = posun_zobrazeni ( A , [ 4 ] , count);
                count = count + 1;
            end
        end
    end
    i = 1;
    j = 1;
    step = step + 1;
    for i = 1 : ( size(A,1) - 1 )
        for j = 1 : size(A,2)
            if ( A( i , j ) == 0 && A( (i + 1) , j ) == 1 )
                A = posun_zobrazeni ( A , [ 1 ] , count);
                count = count + 1;
            elseif ( A( i , j ) == 1 && A( (i + 1) , j ) == 0 )
                A = posun_zobrazeni ( A , [ 1 ] , count);
                count = count + 1;
            else
                done = 0;
            end
        end
    end
    i = 1;
    j = 1;
    step = step + 1;
    for i = 1 : size(A,1)
        for j = 2 : size(A,2)
            if ( A( i , j - 1 ) == 0 && A( i , j ) == 1 )
                A = posun_zobrazeni ( A , [ 3 ] , count );
                count = count + 1;
            end
        end
    end
    i = 1;
    j = 1;
    step = step + 1;
end
```

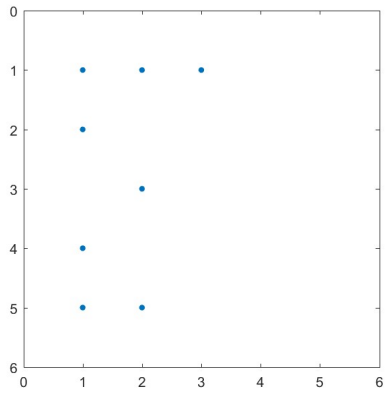
Funkce posun_zobrazení

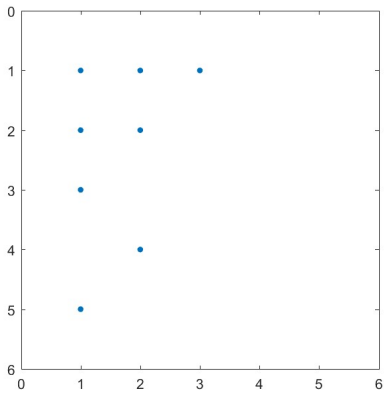
```
function [ matrix ] = posun_zobrazeni ( matrix , vector , save_num)

    for ( i = 1 : length ( vector ) )

        if (vector ( 1 , i ) == 1)
            matrix = posun_nahoru (matrix);
        elseif (vector ( 1 , i ) == 2)
            matrix = posun_dolu (matrix);
        elseif (vector ( 1 , i ) == 3)
            matrix = posun_doleva (matrix);
        elseif (vector ( 1 , i ) == 4)
            matrix = posun_doprava (matrix);
        end

        fig = figure;
        spy(matrix)
        fname = sprintf('SavedPlot%d', save_num);
        print(fig, fname ,'-dpng');
    end
end
```

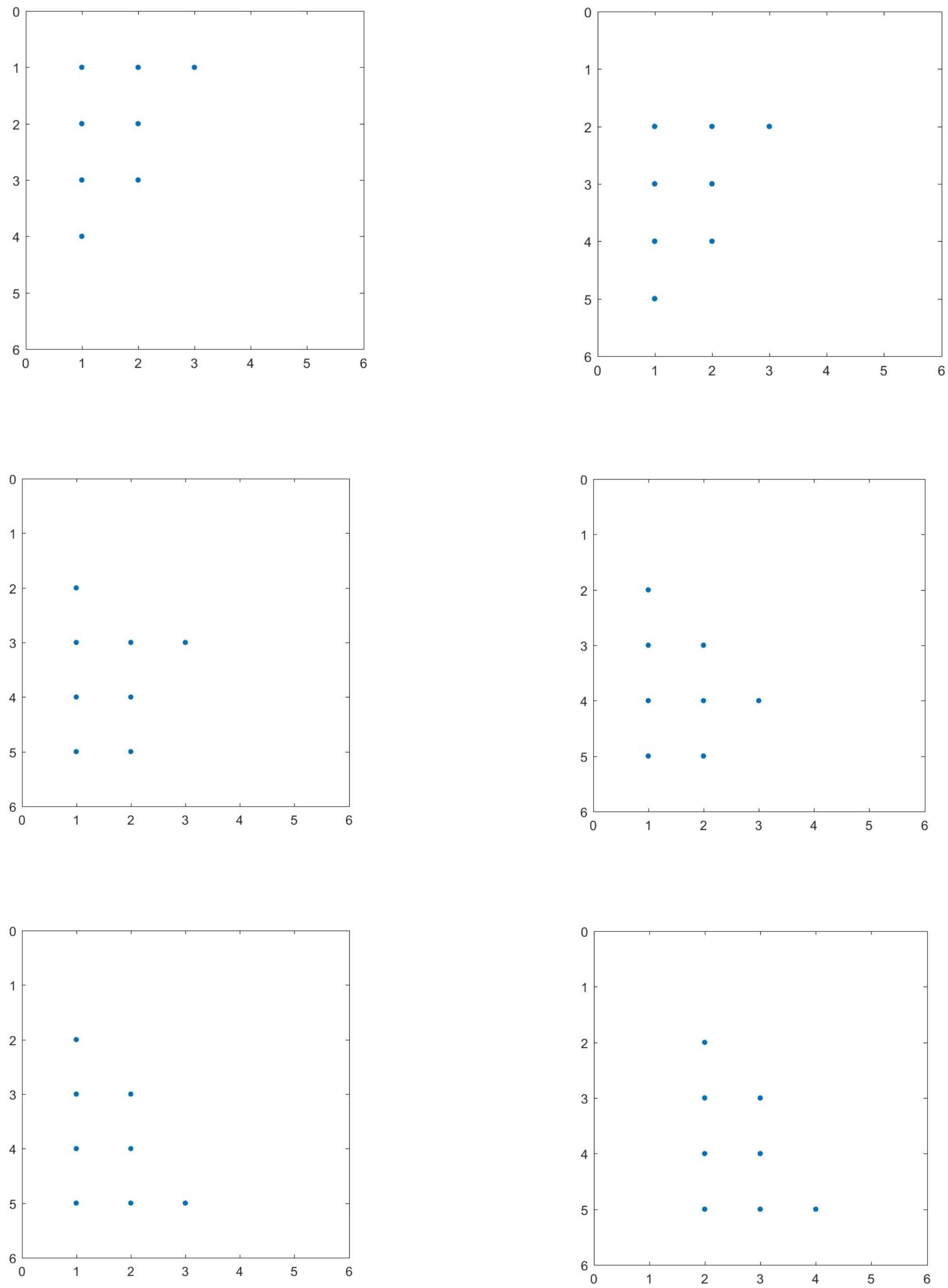
Příloha C – postup robotů v rotaci proti směru h. ručiček

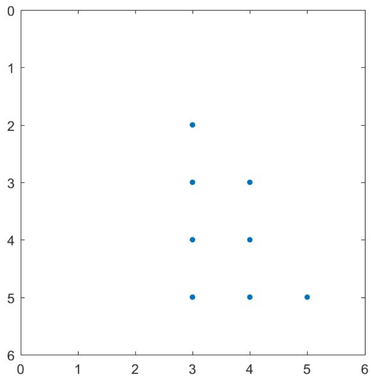


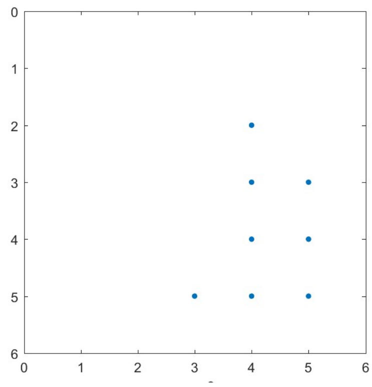


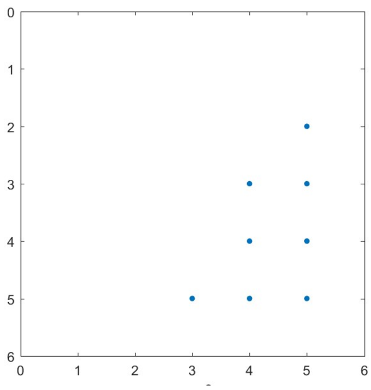


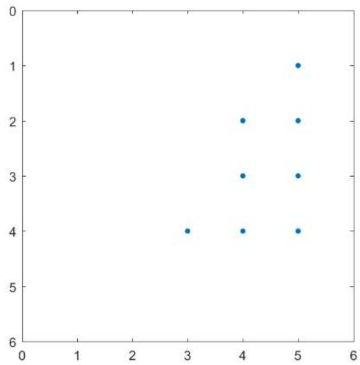


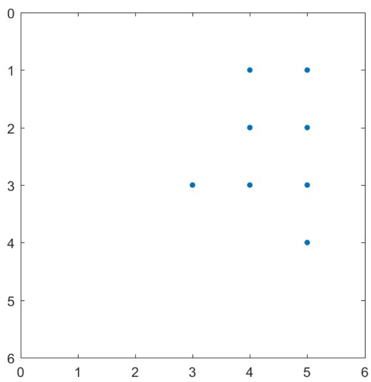

0
1
2
3
4
5
6
0 1 2 3 4 5 6

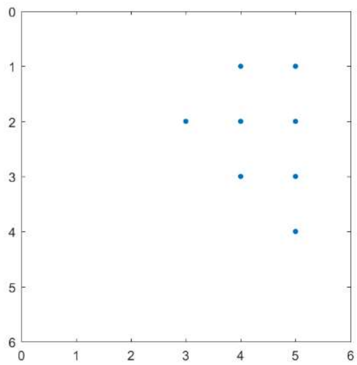


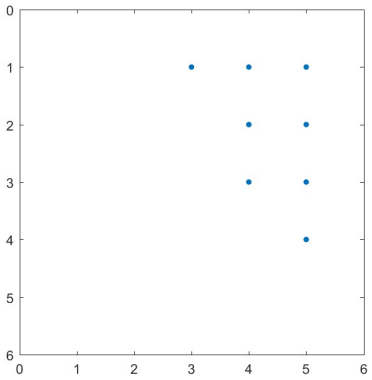












Příloha D – Skript pro rozpoznávání tvaru

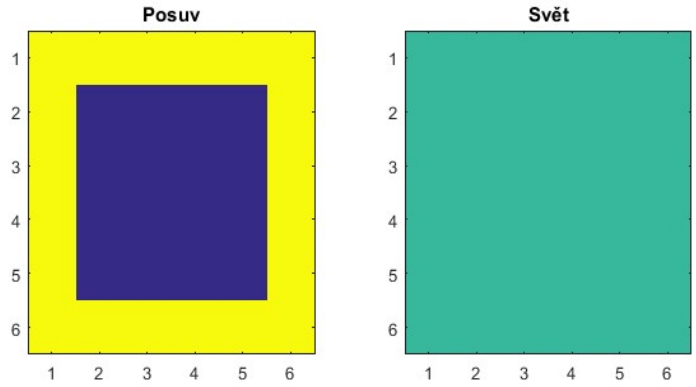
```
done = 0;
count = 1;

%------------- Naplnění matice objektu
matrix_object = [ 0 , 0 , 0 , 0 , 0 , 0 ; ...
                  0 , 0 , 0 , 1 , 0 , 0 ; ...
                  0 , 1 , 1 , 1 , 1 , 0 ; ...
                  0 , 0 , 1 , 1 , 1 , 0 ; ...
                  0 , 0 , 1 , 0 , 1 , 0 ; ...
                  0 , 0 , 0 , 0 , 0 , 0 ];

%------------- Naplnění matice robotů
matrix_robots1 = [ 1 , 1 , 1 , 1 , 1 , 1 ; ...
                   1 , 0 , 0 , 0 , 0 , 1 ; ...
                   1 , 0 , 0 , 0 , 0 , 1 ; ...
                   1 , 0 , 0 , 0 , 0 , 1 ; ...
                   1 , 0 , 0 , 0 , 0 , 1 ; ...
                   1 , 1 , 1 , 1 , 1 , 1 ];

matrix_robots = matrix_robots1;

%------------- Deklarace matice světa
matrix_world = zeros(size(matrix_robots));

figure
subplot(1,2,1); spy(matrix_object); title('Objekt');
hold on
subplot(1,2,2); spy(matrix_robots); title('Roboti');
hold off
% subplot(2,2,3); spy(matrix_robots); title('Posuv');
% hold on
% subplot(2,2,4); spy(matrix_world); title('Svět');

figure
subplot(1,2,1); spy(matrix_robots); title('Posuv');
hold on
subplot(1,2,2); spy(matrix_world); title('Svět');
hold off
```

```
%------------- Rozpoznávání objektu
while done ~= 4
    count = 1;
% %   Zkoumání posunem doleva
    while count < (size(matrix_robots,2)/2)
        help_matrix = zeros(size(matrix_robots));
        for i = 1 : size( matrix_robots , 1 )
            for j = 2 : size( matrix_robots , 2 )
                if ( ( matrix_robots ( i , j ) == 1 ) && ( matrix_robots (
i , ( j - 1) ) == 0 ) && ( matrix_object ( i , ( j - 1) ) == 0 ) )
                    help_matrix ( i , j ) = 0;
                    help_matrix ( i , ( j - 1 ) ) = 1;
                else
                    help_matrix ( i , j ) = matrix_robots ( i , j );
                end
                if (matrix_robots ( i , 1 ) == 1 )
                    help_matrix ( i , 1 ) = 1;
                end
                if ( matrix_object ( i , j ) == 1 && matrix_robots ( i , (
j + 1 ) ) == 1 )
                    matrix_world ( i , j ) = 2;
                end
            end
        end
        matrix_robots = help_matrix;
        count = count + 1;
%         vykreslení
        figure
            subplot(1,2,1); spy(matrix_robots); title('Posuv');
            hold on
            subplot(1,2,2); spy(matrix_world); title('Svět');
            hold off
    end
    matrix_robots
    % %   Zkoumání posunem doprava
    count = 1;
    done = done + 1;
    while count ~= (size(matrix_robots,2)/2)
        for i = 1 : size( matrix_robots , 1 )
            for j = ( size( matrix_robots , 2 ) - 1 ) : - 1 : 1
                if ( ( matrix_robots ( i , j ) == 1 ) && ( matrix_robots (
i , ( j + 1) ) == 0 ) && ( matrix_object ( i , ( j + 1) ) == 0 ) )
                    help_matrix ( i , j ) = matrix_robots ( i , ( j + 1 )
);
                    help_matrix ( i , ( j + 1 ) ) = matrix_robots ( i , j
);
                else
                    help_matrix ( i , j ) = matrix_robots ( i , j );
                end
                if (matrix_robots ( i , size(matrix_robots,2) ) == 1 )
                    help_matrix ( i , size(matrix_robots,2) ) = 1;
                end
                if ( matrix_object ( i , j ) == 1 && matrix_robots ( i , (
j - 1 ) ) == 1 )
                    matrix_world ( i , j ) = 2;
                end
            end
        end
```

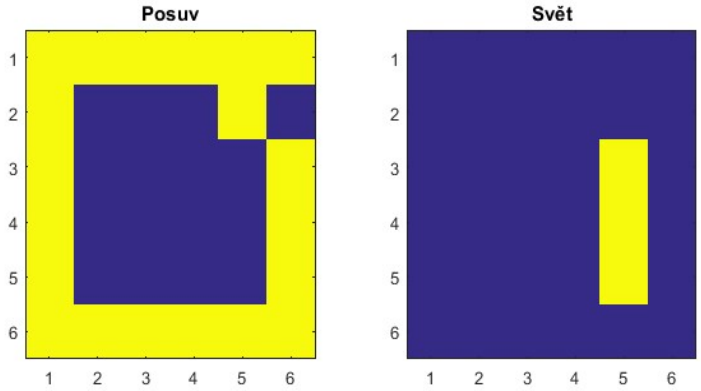
```
        matrix_robots = help_matrix;        count = count + 1;
        figure
           subplot(1,2,1); spy(matrix_robots); title('Posuv');
           hold on
           subplot(1,2,2); spy(matrix_world); title('Svět');
    end
    matrix_robots
    % %   Zkoumání posunem dolů
    count = 1;    done = done + 1;
    while count ~= (size(matrix_robots,1)/2)
        for i = ( size( matrix_robots , 1) - 1) : -1 : 1
            for j = 1 : size( matrix_robots , 2 )
                  if ( matrix_robots(i,j)==1 && matrix_robot((i+1),j)==0 &&
                       matrix_object ((i+1),j)==0)
                    help_matrix (i,j)=matrix_robots((i+1),j);
                    help_matrix ((i+1),j)= matrix_robots (i,j);
                else
                    help_matrix ( i , j ) = matrix_robots ( i , j );
                end
                if (matrix_robots ( 1 , j ) == 1)
                    help_matrix ( 1 , j ) = 1;
                end
                if ( matrix_object (i,j)==1 && matrix_robots ((i-1),j)==1)
                    matrix_world ( i , j ) = 2;
                end
            end
        end
        matrix_robots = help_matrix;
        count = count + 1;
    end
    matrix_robots
    % %   Zkoumání posunem nahoru
    count = 1;    done = done +1;
    while count ~= (size(matrix_robots,1)/2)
        for i = 2 : size( matrix_robots , 1 )
            for j = 1 : ( size( matrix_robots , 2 ) )
                  if ( matrix_robots(i,j)==1 && matrix_robots((i-1),j)==0
                       && matrix_object ( ( i - 1 ) , j ) == 0 ) )
                    help_matrix ( i , j ) = matrix_robots ( ( i - 1 ) , j;
                    help_matrix ( ( i - 1 ) , j ) = matrix_robots ( i , j);
                else
                    help_matrix ( i , j ) = matrix_robots ( i , j );
                end
                if (matrix_robots ( size(matrix_robots,1) , j ) == 1)
                    help_matrix ( size(matrix_robots,1) , j ) = 1;
                end
                if (matrix_object(i,j)==1 && matrix_robots((i+1), j)==1 )
                    matrix_world ( i , j ) = 2;
                end
            end
        end
        matrix_robots = help_matrix;        count = count + 1;
        figure
            subplot(1,2,1); spy(matrix_robots); title('Posuv');
            hold on
            subplot(1,2,2); spy(matrix_world); title('Svět');
            hold off
    end
    done = done + 1;
end
```

Příloha E – Postup skript pro rozpoznání tvaru

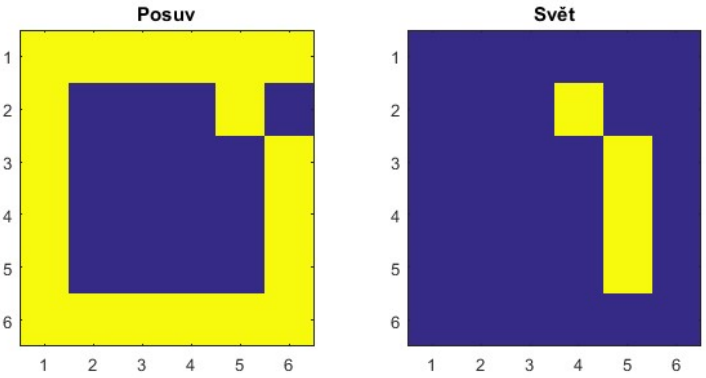
1.krok



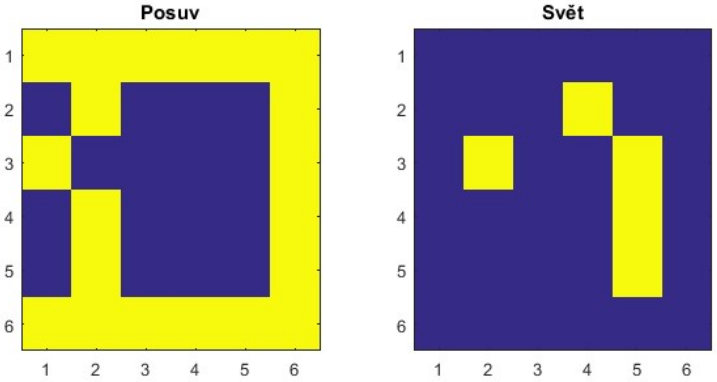
2.krok



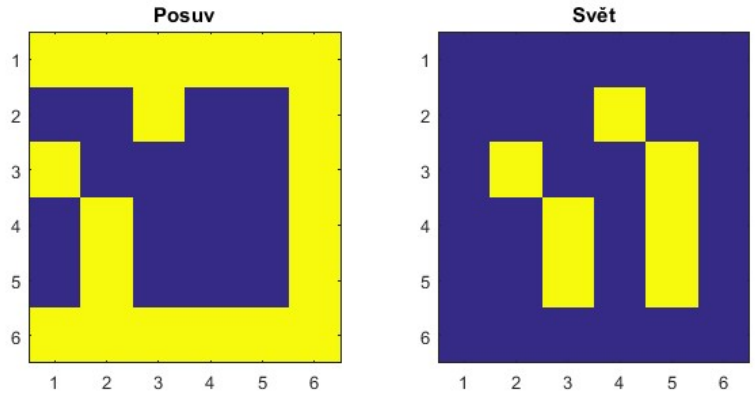
3.krok



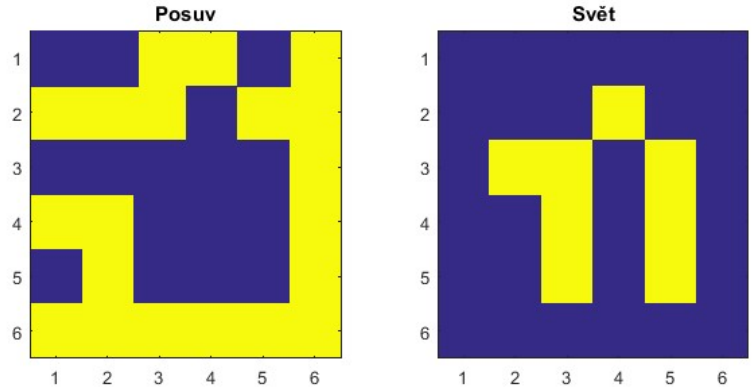
4.krok



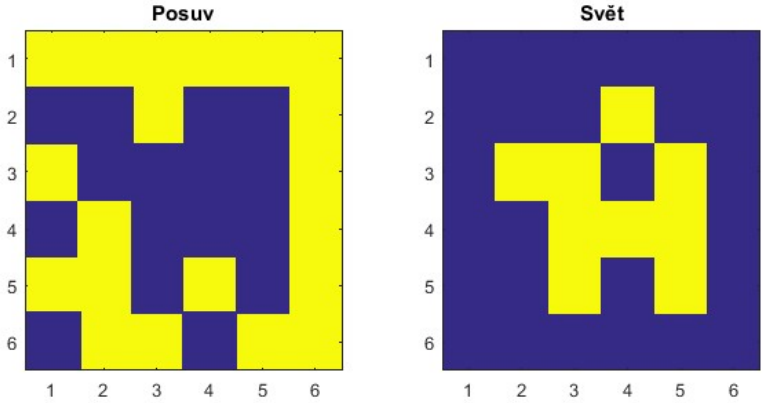
5.krok



6.krok



7.krok



Příloha F – Skript pro ovládání robotů pomocí překážek

```
matrix_world = zeros (30,30);
matrix_object = zeros (30,30);
matrix_robots = zeros (30,30);

% Načtení překážek
% % Dok
matrix_object (4,4) = 2;
matrix_object (5,4) = 2;
matrix_object (6,4) = 2;
matrix_object (7,4) = 2;
matrix_object (8,4) = 2;
matrix_object (9,4) = 2;
matrix_object (10,4) = 2;
matrix_object (11,4) = 2;
matrix_object (11,5) = 2;
matrix_object (11,6) = 2;
matrix_object (11,7) = 2;
matrix_object (11,8) = 2;
matrix_object (11,9) = 2;
matrix_object (11,10) = 2;
% % Překážka
matrix_object (20,20) = 2;

% Načtení robotů
n = 10 ; m = 5 ;
A1 = rand(n,m) < 0.5 ;
A2 = round(rand(n,m));

N0 = ceil(n*m/2) ;
A = ones(n,m) ;
A(1:N0) = 0 ;
A(randperm(numel(A))) = A;

a=0;
for i = 1 : size (A,1)
    for j = 1 : size(A,2)
        matrix_robots (i,j+15) = A (i,j);
        if A(i,j)==1
            a=a+1;
        end
    end
end
a
% Načtení do světa
for i = 1 : size(matrix_world,1)
    for j = 1 : size(matrix_world,2)
        matrix_world(i,j) = matrix_object(i,j) + matrix_robots(i,j);
    end
end
move = 0;
save_num = 1
while move ~=5
    move = input ( ' Doleva - 1, Doprava - 2 , Nahoru - 3, Dolů - 4 \n' );
    if ( move == 1 || move == 2 || move == 3 || move == 4)
        switch move
```

```
case 1
                matrix_robots =
posun_doleva_svet(matrix_robots,matrix_object);
                for i = 1 : size (matrix_world,1)
                    for j = 1 : size(matrix_world,2)
                        matrix_world(i,j) = matrix_object(i,j) +
matrix_robots(i,j);
                    end
                end
                fig = figure;                    imagesc(matrix_world)
                    fname = sprintf('SavedPlot%d', save_num);
                    print(fig, fname ,'-dpng');
                save_num = save_num + 1;
            case 2
                matrix_robots =
posun_doprava_svet(matrix_robots,matrix_object);
                for i = 1 : size (matrix_world,1)
                    for j = 1 : size(matrix_world,2)
                        matrix_world(i,j) = matrix_object(i,j) +
matrix_robots(i,j);
                    end
                end
                fig = figure;                    imagesc(matrix_world)
                    fname = sprintf('SavedPlot%d', save_num);
                    print(fig, fname ,'-dpng');
                save_num = save_num + 1;
            case 3
                matrix_robots =
posun_nahoru_svet(matrix_robots,matrix_object);
                for i = 1 : size (matrix_world,1)
                    for j = 1 : size(matrix_world,2)
                        matrix_world(i,j) = matrix_object(i,j) +
matrix_robots(i,j);
                    end
                end
                fig = figure;
                    imagesc(matrix_world)
                    fname = sprintf('SavedPlot%d', save_num);
                    print(fig, fname ,'-dpng');
                save_num = save_num + 1;
            case 4
                matrix_robots =
posun_dolu_svet(matrix_robots,matrix_object);
                for i = 1 : size (matrix_world,1)
                    for j = 1 : size(matrix_world,2)
                        matrix_world(i,j) = matrix_object(i,j) +
matrix_robots(i,j);
                    end
                end
                fig = figure;
                    imagesc(matrix_world)
                    fname = sprintf('SavedPlot%d', save_num);
                    print(fig, fname ,'-dpng');
                save_num = save_num + 1;
        end
    else
        move = input ( ' Doleva - 1, Doprava - 2 , Nahoru - 3, Dolů - 4 \n'
);
    end
end
```