UNIVERZA V LJUBLJANI
Fakulteta za elektrotehniko

Željko Jotanovič

UMETNA INTELIGENCA PRI VODENJU GOSPODINJSKIH SESALNIKOV

DIPLOMSKO DELO UNIVERZITETNEGA ŠTUDIJA

Mentor: doc. dr. Boštjan Murovec

Ljubljana, 2010

Zahvala

Zahvaljujem se mentorju doc. dr. Boštjanu Murovcu za pomoč, strokovne nasvete ter potrpežljivo pregledovanje diplomskega dela. Zahvala gre tudi mojim staršem, ki so mi omogočili študij ter me pri tem podpirali. Zahvaljujem se tudi štipenditorju Slovenske železnice, ker me je finančno podpiral v času študija. In nenazadnje, hvala tudi punci Nini in prijateljem za podporo in spodbudo v času študija.

Povzetek

V diplomski nalogi je opisana simulacija avtonomnega robotskega sesalnika, ki je sposoben samostojnega odločanja in načrtovanja poti v inteligentnem okolju. Pri tem uporaba algoritmov umetne inteligence omogoča napredno, inteligentno in energijsko varčno strategijo čiščenja.

Diplomsko delo je razdeljeno na tri dele. Prvi zajema pregled trenutnega stanja tržišča inteligentnih sesalnikov. Sledi definicija inteligentnega okolja, ki temelji na omrežju robotskih sistemov in raziskava možnih izboljšav, ki jih ponuja inteligentno okolje na področju storitev čiščenja. Sodelovanje različnih robotskih sistemov v inteligentnem okolju prikažemo na primeru robotskih naprav v gospodinjstvu.

V drugem delu opisujemo algoritma za iskanje poti in odločanje. Odločanje inteligentnega agenta realiziramo z mehko logiko na osnovi različnih kriterijev kot je umazanost okolja in njegova oddaljenost. Za iskanje poti v diskretiziranem okolju uporabimo algoritem A*, ki temelji na Dijsktrovem algoritmu.

Zadnji del naloge nas seznani z Microsoftovim programskim paketom XNA Game Studio, katerega uporabljamo za izdelavo simulacije. Slednjo prikažemo v tridimenzionalnem navideznem okolju, opravimo analizo algoritma za iskanje poti, podamo nastavitve parametrov mehke logike in rešitev problemov, ki so se pojavili med izdelavo algoritma. Primere delovanja inteligentnega agenta prikažemo v treh različnih diskretiziranih okoljih.

Ključne besede: umetna inteligenca, avtonomni robotski sesalnik, omrežje robotskih sistemov, iskanje poti, algoritem A*, Dijsktrov algoritem, mehka logika, diskretizacija, Microsoft XNA Game Studio.

Abstract

This thesis discusses the simulation of autonomous robotic cleaner, capable of decision making and pathfinding in the intelligent environment. We use the artificial intelligence algorithms, which provide us advanced and energy efficient cleaning strategy.

The thesis consists of three major parts. The first one includes an insight into the current available intelligent cleaners on the market. It is followed by the definition of intelligent environment, based on the network robotic system. Cooperation of the different robotic systems in the intelligent environment is shown on the example in the domestic environment.

In the second part we describe the algorithms for pathfinding and decision-making. The decision-making of the intelligent agent is based on the fuzzy logic using multiple criteria, for instance pollution of the environment and the distance of it. For pathfinding we use algorithm A*.

The last part of the thesis describes the basic properties of XNA Game Studio, which was used for the 3D simulation of the environment. We analyze the pathfinding algorithm, the settings for fuzzy logic and the solutions for the problems, which occurred during the making of the thesis. The example of intelligent agent is shown in three different discretisized environments.

Keywords: artificial intelligence, autonomous robotic cleaner, network robotic system, pathfinding, decision-making, fuzzy logic, A* algorithm, Dijsktra algorithm, discretization, Microsoft XNA Game studio.

Kazalo

1.Uvod 1

2. Avtonomni robotski sesalniki 3

2.1 Definicija in delitev avtonomnih robotskih sesalnikov 3

3. Omrežje robotskih sistemov 7

3.1 Definicija omrežja robotskih sistemov 7

3.2 Primeri omrežij robotskih sistemov 8

3.2.1 Projekt URUS 8

3.2.2 Japonski projekt omrežja robotskih sistemov 9

3.2.3 PEIS ekologija 10

4. Umetna inteligenca v robotiki 19

4.1 Iskanje poti 19

4.1.1 Predstavitev problemov v prostoru stanj 20

4.1.2 Iskalni Graf 21

4.2.3 Diskretizacija območja 23

4.2.4 Iskalni algoritmi 26

4.2.5 Dijkstrov algoritem 27

4.2.6 Algoritem A* 30

4.2 Mehka logika 35

4.2.1 Mehke množice 35

4.2.2 Lastnosti mehkih množic 39

4.2.3 Osnovne operacije nad mehkimi množicami 39

4.2.4 Mehka pravila in sklepanje 41

4.2.4.1 Mehka pravila 41

4.2.4.2 Mehko sklepanje 43

5. Simulacija inteligentnega agenta 47

5.1 Microsoft XNA Game Studio 47

5.2 Analiza algoritma A* 49
5.3 Odločanje inteligentnega agenta z mehko logiko..... 53
5.3.1. Definicija problema in potek modeliranja sistema mehkega odločanja 54
5.3.2 Sestava baze pravil mehkega odločanja 56
5.3.3 Določanje vhodnih in izhodnih lingvističnih spremenljivk..... 57
5.3.4 Dodatne modifikacije algoritma inteligentnega agenta ter nastavitve simulacije ... 61
5.3.5 Analiza uporabe mehke logike kot sistema za odločanje 62
6. Zaključek..... 69
7. Literatura..... 71

Kazalo slik

Slika 1: iRobot Roomba 610. 5
Slika 2: Electrolux Trilobite. 6
Slika 3: ST82 R VARIOTECH Hefter Cleantech v veleblagovnici. 6
Slika 4: študija PEIS ekolgije povezuje tri področja. 11
Slika 5: avtonomni robotski sesalnik v PEIS okolju. 12
Slika 6: PEIS ekologija je heterogena glede na strojno opremo, programsko opremo in način komunikacij. 13
Slika 7: PEIS mizica. 14
Slika 8: a) StarLITE kamera in brezžična oznaka in b) primer pozicioniranja s sistemom StarLITE. 15
Slika 9: a) zemljevid okolja in b) diskretiziran zemljevid okolja. 16
Slika 10: primer algoritma za iskanje poti. 16
Slika 11: PEIS mizica v realnem okolju. 17
Slika 12: a) pot v realnem okolju, b) diskretizacija okolja in c) transformacija okolja v graf. 20
Slika 13: a) neusmerjen graf in b) usmerjen graf. 22
Slika 14: a) graf z uteženimi povezavami in b) predstavitev geometrije okolja z grafom. 23
Slika 15: a) mnogokotniška delitev okolja in b) primeri mnogokotniških mrež. 24
Slika 16: a) vozlišča v središču mnogokotnikov in b) vozlišča na središču stranic mnogokotnika. 24
Slika 17: a) konveksen in b) konkaven mnogokotnik. 25
Slika 18: izvedba algortima Hertel – Mehlhorn nad mnogokotnikom. 26
Slika 19: prva iteracija Dijkstrovega algortima nad začetnim vozliščem A. 28
Slika 20: druga iteracija Dijkstrovega algortima: izbrano je vozlišče B z najmanjšo ceno poti. 28
Slika 21: nadgradnja vozlišča iz seznama »OPEN«. 29
Slika 22: a) h(sj) z razdaljo Manhattan in b) h(sj) z Evklidovo razdaljo. 32
Slika 23: primer algoritma A*. 33
Slika 24: primer algoritma A*. 33
Slika 25: primer algoritma A*. 34
Slika 26: graf pripadnostne funkcije za klasično množico visokih ljudi. 37
Slika 27: graf pripadnostne funkcije za mehko množico visokih ljudi. 37
Slika 28: graf pripadnostnih funkcij za mehke množice višinskih razredov. 38

Slika 29: grafi pripadnostnih funkcij: a) trikotna, b) logistična in c) Gaussova. 39
Slika 30: graf pripadnostnih funkcij za mehko množico težnostnih razredov. 42
Slika 31: graf pripadnostnih funkcij za primernost skakalca v višino. 43
Slika 32: tipični sistem mehkega sklepanja. 44
Slika 33: obrezovanje grafov pripadnostnih funkcij pri doseženi aktivaciji in tvorba skupnega območja. 45
Slika 34: metoda maksimalne pripadnosti. 46
Slika 35: diagram poteka simulacijskega teka. 48
Slika 36: a) zemljevid zasedenosti in b) prikaz ovir v navideznem okolju. 49
Slika 37: a) vozlišče s štirimi povezavami in b) z osmimi. 50
Slika 38: a) hevristika Manhattan in b) Evklidova hevristika. 51
Slika 39: a) Čebiševa in b) diagonalna hevristika. 51
Slika 40: iskanje poti s kvadrirano Evklidovo hevristiko. 53
Slika 41: a) hevristika Manhattan in b) Evklidova kvadrirana hevristika. 53
Slika 42: postopek fuzifikacije in defuzifikacije posamezne smeti na seznamu vseh smeti. 54
Slika 43: vhodna mehka spremenljivka 'umazanost'. 57
Slika 44: prikaz smeti z barvami v navideznem okolju. 57
Slika 45: vhodna mehka spremenljivka 'cena poti'. 58
Slika 46: izračun cene poti do vseh smeti v eni iteraciji. 58
Slika 47: vhodna mehka spremenljivka 'cena poti'. 59
Slika 48: a) tloris stanovanjske enote in b) diskretizacija. 59
Slika 49: trajektorija poti inteligentnega agenta. 61
Slika 50: a) prikaz gruče smeti in b) agent v postopku odločanja. 61
Slika 51: trajektorija pri a) ne-upoštevanju spremembe in b) upoštevanju spremebe smeri. 62
Slika 52: a) prikaz umazanega okolja in b) prikaz prioritete v sivinskih barvah. 62
Slika 53: diskretizirano in 'umazano' okolje. 63
Slika 54: trajektorija agenta s a) hevristiko Manhattan in b) Evklidovo hevristiko. 63
Slika 55: trajektorija agenta z a) diagonalno in b) Čebiševo hevristiko. 64
Slika 56: a) tloris stanovanjske enote in b) diskretizacija. 64
Slika 57: trajektorija agenta s a) hevristiko Manhattan in b) Evklidovo hevristiko. 65
Slika 58: trajektorija agenta z a) diagonalno in b) Čebiševo hevristiko. 65
Slika 59: a) tloris gostinske enote in b) diskretizacija. 66
Slika 60: trajektorija agenta s a) hevristiko Manhattan in b) Čebiševo hevristiko. 66
Slika 61: trajektorija agenta z a) Evklidovo in b) diagonalno hevristiko. 66

Kazalo tabel

Tabela 1: primerjava časa izračunavanj, cene iskanih poti ter število obiskanih vozlišč.....51
Tabela 2: primerjava časa izračunavanj, cene iskanih poti ter število obiskanih vozlišč za zemljevid dimenzij 93×32. 52
Tabela 3: primerjava časa izračunavanj, cene iskanih poti ter število obiskanih vozlišč.....53
Tabela 4: posamezni koraki odločanja inteligentnega agenta. 60
Tabela 5: cene poti pri posameznih hevristikah. 64
Tabela 6: cene poti pri posameznih hevristikah. 65
Tabela 7: cene poti pri posameznih hevristikah. 67

1.Uvod

S stališča genetike so bili ljudje že pred 100.000 leti skoraj enaki sodobnim ljudem. Pred približno 40.000 leti sta se pri človeku začela razvijati iznajdljivost in domiselnost. Ne ve se natančno, kaj je spodbudilo razvoj mentalnih sposobnosti, vendar so nam slednje tako pomembne, da smo sami sebe poimenovali »sodobni, misleči človek« (*lat. Homo Sapiens Sapiens*). Že tisočletja se sprašujemo in poskušamo razumeti koncept razmišljanja, področje umetne inteligence pa gre korak naprej, saj ima za cilj ustvariti inteligetne entitete.

Korenine umetne inteligence segajo že v grško mitologijo. Ideja »inteligetnih robotov« se prvič pojavi v grškem mitu Hefajst in Pigmalion. V moderni dobi je pionir na tem področju Alan Turing (1912-1954), ki je med drugim znan po »Turingovem testu«. S filozofskim vprašanjem na začetku članka »Ali stroji lahko mislijo?« je Turing leta 1950 podal postopek testa, ki oceni stopnjo inteligentnosti stroja. Za pravi začetek umetne inteligence smatramo konferenco v Dartmouthu v ZDA leta 1956, kjer je John McCarthy predlagal dvomesečni študij umetne inteligence. Tedaj se je prvič pojavil pojem umetna inteligenca [1].

Umetna inteligenca (ang. *artificial intelligence*) je obsežno in raznoliko področje računalništva. Opišemo jo kot študijo inteligentnih agentov (ang. *intelligent agents*), ki zaznavajo in sprejemajo podatke iz okolice ter z implementirano funkcijo izvršujejo akcije. Z agenti predstavimo algoritme različnih področij, kot so reševanje problemov z iskanjem, avtomatsko sklepanje, planiranje, verjetnostno sklepanje, klasifikacija, razpoznavanje itd. [2].

Zadnje desetletje opažamo znaten razvoj robotike na področju gospodinjskih naprav. Naprave, kot so avtonomni robotski sesalniki, samostojno počistijo domače okolje in nam olajšajo vsakdanja domača opravila. Proizvajalci se trudijo z različnimi algoritmi umetne inteligence doseči čim bolj inteligenten način sesanja ter s tem energijsko varčno in učinkovito strategijo čiščenja prostorov. Delež inteligentnih sesalnikov na trgu storitev čiščenja ni več tako zanemarljiv, saj je bilo v zadnjih desetih letih prodanih več milijonov enot pametnih sesalnikov [3].

Robot kot izolirana avtonomna mobilna enota je v pasivnem okolju omejen. Alternativna vizija, ki prihaja iz področja inteligentnega okolja, je vzpostavitev omrežja inteligentnih naprav, ki so sposobne z nami in med seboj komunicirati ter opravljati razna opravila namesto nas. Omenjen pristop ima ekološki pogled na okolje robota, v katerem sta robot in okolje obravnavana kot del istega sistema, ki imata simbiotično razmerje v smislu doseganja skupnega cilja ali ravnotežnega stanja. Robotske naprave so porazdeljene po okolju v obliki senzorjev, stikal, kamer, predmetov označenih z RFID (ang. *radio freqeuny identificaton*) oznakami, enostavnih mobilnih ali stacionarnih robotov itd. Namen opisanega sistema je povezava heterogene skupine naprav, ki medsebojno komunicirajo in sodelujejo. Avtonomni robotski sesalnik ima v takem okolju dostop do vseh funkcionalnosti ostalih robotskih enot (npr. prepoznavanje ljudi in ovir preko video kamer, natančno globalno pozicioniranje ...). Z omenjenim pristopom se maksimizira kriterij učinkovitosti v primerjavi s konvencionalnimi pametnimi sesalniki [4].

Diplomska naloga nas seznani z definicijo inteligentnega agenta v obliki avtonomnega robotskega sesalnika. V nadaljevanju podajamo definicijo omrežja robotskih sistemov ter primere inteligentnih okolij, ki temeljijo na omenjenem konceptu. Pri razlagi izpostavljamo inteligento okolje v notranjih prostorih z robotskimi gospodinjskimi napravami. Temu sledi opis algoritmov umetne inteligence za iskanje poti in odločanja.

V zadnjem delu naloge z uporabo algoritmov, kot so mehka logika in iskanje poti z algoritmom A*, prikazujemo rešitev strategije čiščenja avtonomnega robotskega sesalnika. Za prikaz umetne inteligence smo v tridimenzionalnem navideznem okolju ustvarili inteligentnega agenta, katerega naloga je odločanje, načrtovanje poti in premik. Implementacijo algoritmov in predstavitev simulacije izdelamo z Microsoftovim programskim paketom XNA Game Studio v programskem jeziku C#. Prikazujemo simulacijo v različnih oblikah diskretiziranega okolja z različnimi parametri algoritmov ter podajamo analizo rezultatov. V okolju pred pričetkom simulacije nastavimo stopnjo umazanosti posameznih delov okolja.

2. Avtonomni robotski sesalniki

Izum sesalnikov je posledica napredka v znanosti in tehnologiji v času industrijske revolucije. Do sredine 19. stoletja so tovarne proizvedle na tisoče proizvodov, vključno s tonami nesnage. Umazanija in dim sta bila prisotna povsod. V tem času je znanstvenik Louis Pasteur razvil teorijo, da nalezljive bolezni povzročajo mikroorganizmi. Z razvojem te teorije in reakcijo ljudi proti industrijskemu onesnaževanju se je pričelo osredotočanje na higieno in čistočo. Pred izumom električnih sesalnikov je večina gospodinj uporabljala metlo in smetišnico. Uporaba sesalnikov je izboljšala saniteto, splošno zdravje in kakovost življenja [5].

Mejniki:

- 1860 - Daniel Hess patentira sesalnik, prvi zapis.
- 1899 - John S.Thurman izumi sesalnik na gorivo.
- 1901 - Hubert Cecil Booth iz Londona izumi električni sesalnik.
- 1905 - Chapman in Skinner iz San Francisca izdelata prenosni (~ 42 kg) električni sesalnik.
- 1920 - AirWay Sanitizor iz Ohia predstavi prvi sesalnik z zamenljivimi vrečkami.
- 1950 - Prvi pokončni sesalnik.
- 1979 - Black & Decker's predstavi brezžični ročni sesalnik "Dustbuster".
- 1991 - Hitachi, Sanyo predstavi prvi prototip avtonomnega robotskega sesalnika.
- 1996 - Hefter Cleantech/Siemens AG razvijejo prototip industrijskega robotskega sesalnika za uporabo na večjih površinah (supermarketi, letališča, dvorane...).
- 1996 - Electrolux – predstavi prvi komercialni avtonomni robotski sesalnik (v prodaji šele leta 2001)
- 2002 - iRobot predstavi Roombo – trenutno najbolj prodajan robotski sesalnik [6].

2.1 Definicija in delitev avtonomnih robotskih sesalnikov

Inteligentni sesalnik opišemo kot avtonomni mobilni robot, ki je zmožen samostojno čistiti okolje. Ker tak sesalnik uporablja napredne računalniške algoritme umetne inteligence za različne strategije čiščenja, ga označimo kot inteligentnega agenta. Pod pojmom inteligentni agent si predstavljamo predvsem programskega agenta, ki izkazuje obliko inteligence, kar daje agentu sposobnost prilagajanja in učenja. Pojem agent razumemo kot poskus

povezovanja na zunaj zelo različnih tem, ki so predmet obravnavane umetne inteligence. V praksi se pojavlja širok spekter lastnosti, ki naj bi jih imel inteligentni agent, med katerimi izpostavimo naslednje [2].

- *Avtonomnost*. Agent je pri svojem delovanju samostojen in ne zahteva posegov upravljalca.
- *Racionalnost*. Racionalni agent deluje tako, da za vsako možno zaporedje zaznanih predmetov izbere akcijo, ki čimbolj maksimizira nek kriterij učinkovitosti.
- *Prilagodljivost*. Agent se avtomatsko prilagaja spremembam v okolju.
- *Komunikativnost*. Agent sodeluje v kompleksnih komunikacijah z drugimi agenti, vključno z ljudmi, da bi pridobil informacije, ki mu pomagajo pri izpolnitvi ciljev.
- *Sodelavanje*. Agent ne uboga ukazov slepo, temveč ima možnost spreminjati zahteve, postavljati vprašanja za razjasnitve ali zavračati določene zahteve.
- *Fleksibilnost*. Agentove akcije niso določene s scenarijem. Agent je zmožen dinamično izbirati akcije v odvisnosti od stanja okolja.

V danem trenutku je izbira akcije agenta odvisna od celotnega zaporedja zaznavnih predmetov, opazovanih do tega trenutka. Obnašanje agenta je opisano z njegovo funkcijo, ki preslika dano zaporedje zaznavanih predmetov v akcijo. Agenta opišemo tudi s kriterijem učinkovitosti, okoljem, aktuatorji in senzorji. Primer takega agenta je fizični objekt – npr. sesalnik v domačem okolju. Kriterij učinkovitosti je stopnja pospravljenosti okolja. Z različnimi senzorji zaznavamo okolje, razdalje do ovir... Aktuatorji premikajo sesalnik in ob določenih akcijah sprožijo različne mehanizme čiščenja.

Evropski trg je na področju storitev čiščenja ocenjen na sto milijard dolarjev letno [3], zaradi česar v zadnjem desetletju opažamo znaten razvoj na področju avtonomnih mobilnih sesalnikov, tako industrijskih kot tistih za zasebno uporabo. Na najvišji ravni delimo avtonomne robotske sesalnike glede na njihovo uporabo v dve kategoriji:

- Industrijski čistilni roboti so opazno večji in težji od domačih sesalnikov. Cena je večja tudi za faktor 100. Razlika v ceni je posledica zapletenih čistilnih mehanizmov, ki jih vsebuje industrijski sesalnik (npr. škropljenje, čiščenje, sesanje). Običajno so taki sesalniki opremljeni z napredno navigacijsko opremo, ki zmore natančno zaznavati okolje, se izogibati oviram in se pozicionirati. Nekateri sesalniki so opremeljeni tudi z naprednimi senzorji, ki so sposobni zaznavati umazanost tal in izračunati optimalno pot čiščenja v delovnem prostoru.

- Čistilni roboti za domačo uporabo so pogosto manjše enote, sestavljene iz cenovno dostopnih sestavnih delov. Njihov senzorski sistem je ponavadi okrnjen do te mere, da sesalnik opravlja le enostavne strategije čiščenja. Idealni so za gospodinjstva z veliko odprte talne površine (laminati, preproge...) in malo ovirami.

Primeri nekaterih čistilnih robotov za zasebno in industrijsko uporabo:

- **iRobot Roomba** [6]**:** Podjetje iRobot je ustanovljeno leta 1991 in je specializirano za razvoj robotov, ki pomagajo ljudem opravljati vsakdanje naloge. Do danes je iRobot prodal več kot tri milijone enot pametnih sesalnikov Roomba. Podjetje razvija tudi robote za vojaške namene. Obstaja več serij in modelov pametnega sesalnika Roomba. Eden izmed njihovih naprednejših modelov je Roomba 610 (slika 1).

Slika 1: iRobot Roomba 610.

Sesalnik je opremljen z inteligentnim sistemom, ki se glede na podatke iz niza IR senzorjev odloči za različne profile vedenja. Senzorji sesalnik usmerjajo med ovirami in preprečujejo padec po stopnicah ter omogočajo, da se sesalnik dlje časa zadrži na bolj umazanih območjih. Vsebuje sistem navideznega zidu (tehnologija Virtual Wall Lighthouse z IR senzorji), ki preprečuje prehod Roombe v drugo sobo, preden ta počisti sobo, v kateri se trenutno nahaja. Sesalnik se po sesanju samodejno vrne k bazni postaji, kjer se prične polniti.

- **Electrolux Trilobite** [7]**:** Robotski sesalnik Trilobite (slika 2) je proizvod švedskega podjetja Electrolux. Prototip je bil predstavljen leta 1996. Trilobite je prvi komercialni avtonomni robotski sesalnik, predstavljen kot proizvod leta 2001. Sesalnik druge generacije (Trilobite 2.0) ima več kot 200 izboljšav. Za razliko od večine ostalih pametnih sesalnikov Trilobite za navigacijo po prostoru uporablja ultrazvočne

senzorje. Med čiščenjem si gradi zemljevid okolja in določi energijsko varčno pot sesanja. Z ultrazvočnimi senzorji pri frekvenci 60 kHz zazna ovire. Z IR senzorji zazna stopnice in kable. Ko zaključi s sesanjem, najde pot do bazne postaje in se prične polniti. Z magnetnimi zaščitnimi trakovi označimo področja, na katerih ne želimo sesanja.

Slika 2: Electrolux Trilobite.

- **ST82 R VARIOTECH Hefter Cleantech** [3]**:** Siemens AG in Hefter Cleantech so razvili industrijski robotski sesalnik (slika 3) za čiščenje večjih površin (veleblagovnice, letališča,...). Sesalnik deluje kljub prisotnosti ljudi. Poleg številnih mehanizmov čiščenja je robot opremljen z naprednim navigacijskim sistemom SINAS (Siemens), ki vsebuje navigacijski sistem vsebuje laserski senzor, niz ultrazvočnih senzorjev, giroskop in odometrični sistem. Sesalniku prvič pokažemo (naučimo) pot po kateri želimo, da počisti tla. V tem procesu sesalnik izdela zemljevid okolja, ki ga pozneje uporabi za avtonomno navigacijo. Sesalnik je zmožen nadgrajevati zemljevid med samostojnim delovanjem.

Slika 3: ST82 R VARIOTECH Hefter Cleantech v veleblagovnici.

3. Omrežje robotskih sistemov

Avtonomni robotski sesalniki, ki so trenutno na tržišču, opravljajo dobro lokalno pozicioniranje in izogibanje oviram, vendar to ni dovolj za zanesljivo globalno pozicioniranje in zapleteno strategijo čiščenja na večjem območju. Če prostor opremimo z različnimi senzorji in ostalimi avtonomnimi roboti ter vključimo v skupno rabo funkcionalnosti posameznih enot, lahko inteligentni sesalnik ali kakšna druga naprava izkorišča dodatne funkcije in opravlja delo učinkoviteje. V ta namen je razvit koncept, povezovanja robotov v omrežje za delitev funkcionalnosti.

3.1 Definicija omrežja robotskih sistemov

Dosežki na področju robotike in umetne inteligence so omogočili zasnovo sistema, ki je sposoben različnih interakcij s človekom preko množice sodelujočih avtonomnih inteligentnih enot. Tehnologija se imenuje omrežje robotskih sistemov (ang. *network robot systems*) [8] in vključuje naslednje elemente:

- Fizično utelešenje: Vsako omrežje robotskih sistemov mora vsebovati vsaj enega robota v fizični obliki, ki vključuje strojno in programsko opremo.
- Avtonomne zmogljivosti: Fizični robot mora imeti avtonomne zmogljivosti, da ga obravnavamo kot osnovni element mreže robotskih sistemov.
- Omrežno sodelovanje: Roboti, senzorji okolja in ljudje sodelujejo in komunicirajo preko omrežja.
- Senzorji in aktuatorji: Poleg robotovih senzorjev mora okolje vsebovati tudi druge senzorje (npr. kamere, merilnike razdalje) in aktuatorje (npr. zvočniki).
- Interakcije: Če želimo sistem obravnavati kot omrežje robotskih sistemov, mora imeti povezane akcije v smislu človek – robot.

Omrežje robotskih sistemov je definiral tehnični odbor IEEE RAS (ang. *Robotics and Automation Society)* maja 2004, kot posledico predhodnega dela in razvoja upravljanja robotov na daljavo (preko spleta). Strokovni odbor društva IEEE RAS je za omrežje robotskih sistemov podal naslednjo definicijo.

Mrežni robot je naprava, povezana v omrežje, kot sta internet ali LAN. Mreža je lahko osnovana različnih protokolih (npr. TCP, UDP, 802.11...), žična ali brezžična. Obstajata dva podrazreda omrežnih robotov:

- Roboti, vodeni na daljavo, kjer človek pošilja ukaze in dobi povratne informacije preko omrežja.
- Avtonomni roboti, kjer poteka izmenjava podatkov med roboti in senzorji prek omrežja. V takšnih sistemih mreža senzorjev učinkovito razširja zaznavanje robotov in jim omogoča medsebojno komunikacijo na daljše razdalje z namenom usklajevanja svojih dejavnosti.

Omrežje lahko vsebuje tri vrste robotov, in sicer:

- Vidni roboti, ki se pojavljajo v različnih oblikah, npr. hišne živali, sesalniki ali humanoidi.
- Virtualni roboti, ki so predstavljeni v kibernetskem prostoru (npr. inteligentni agent (avatar) na mobilnem telefonu ali osebnem računalniku).
- Nezavedni tip robota označuje kamera ali senzor. Taki roboti so vgrajeni v ceste, stavbe, parke ali sobe. Uporabniki se ne zavedajo prisotnosti takega tipa robota.

Preprostejšo definicijo je predlagala evropska študijska skupina za raziskavo omrežij robotskih sistemov (ang. *european robotics research network - EURON*):
Omrežje robotskih sistemov je množica avtonomnih inteligentnih sistemov, ki izkoriščajo prednosti brežične komunikacije za komunikacijo med seboj ali z inteligentnim okoljem ter opravljanje določene naloge. Omrežje robotskih sistemov združuje več področij, kot so robotika, zaznavanje (senzorni sistemi), vseprisotno računalništvo, umetna inteligenca in komunikacijsko omrežje. Nekatera izmed ključnih vprašanj, ki jih je treba obravnavati pri oblikovanju mreže robotskih sistemov so lokalizacija in navigacija več enot, skupinsko dojemanje okolja, skupno grajenje zemljevida, dodelitev nalog, interakcije človek-robot, omrežja in komunikacije.

3.2 Primeri omrežij robotskih sistemov

V nadaljevanju opisujemo tri največje projekte [8], ki trenutno potekajo na področju razvoja omrežij robotskih sistemov. Izpostavljamo projekt PEIS ekologija, ki temelji na inteligentnem okolju v domovih.

3.2.1 Projekt URUS

Splošni cilj projekta URUS (ang. *Ubiquitous Robotics Network System for Urban Settings*) [9] je razvoj novih načinov sodelovanja med omrežnimi roboti, ljudmi in/ali okoljem na

urbanih območjih, da bi učinkovito opravili naloge, ki so drugače zapletene, dolgotrajne in predrage. Projekt je usmerjen v mestna območja s pešci. To je pomembna tema v Evropi, kjer obstaja večje zanimanje za zmanjšanje števila avtomobilov na ulicah in izboljšanje kakovosti življenja. Omrežni roboti so pomemben instrument za reševanje teh vprašanj.

Projekt URUS se osredotoča na načrtovanje in razvijanje omrežja robotov, ki z medsebojnim sodelovanjem komunicirajo z ljudmi in okoljem pri nalogah vodenja in pomoči, prevozih blaga in nadzora v mestnih območjih.

Glavni znanstveni in tehnološki izzivi projekta so:

- navigacija in koordinacija gibanja robotov,
- medsebojno zaznavanje okolja,
- medsebojno sodelovanje pri izdelavi in posodobitvah zemljevida,
- interakcija človek – robot,
- brezžična komunikacija med uporabniki, okoljem in roboti.

Projekt URUS se preizkuša v mestnem okolju (na 10,000 m^2 površine v severnem kampusu tehnične univerze v Barceloni), kjer so predvideni trije scenariji: prevoz oseb, usmerjanje in evakuacija ljudi. V projekt URUS je vključenih enajst evropskih partnerjev iz šestih različnih držav.

3.2.2 Japonski projekt omrežja robotskih sistemov

Japonski projekt omrežja robotskih sistemov (ang. *Japan NRS project*) [10] se je začel leta 2004. Cilj projekta je omogočiti uporabniku prijazno interakcijo med ljudmi in inteligentim okoljem. Projekt je sestavljen iz štirih glavnih vidikov: komunikacija med roboti, omrežna platforma, vsepovsod prisotno računalništvo, interakcija med človekom in robotom. Avtonomni mobilni roboti (npr. humanoid, hišni ljubljenček itd.) opravljajo vlogo zagotavljanja informacij ljudem, medtem ko opazovanje in prepoznavanje ljudi večinoma opravljajo v okolje vgrajeni senzorji.

Japonski projekt ima dve značilnosti:

- Usmerjen je na interakcijo človek-robot. Roboti ne opravljajo fizičnih nalog, temveč sodelujejo pri nalogah komuniciranja z uporabniki.

- Poudarek je v socialni umestitvi robota v realno okolje. Eden od pomembnejših primerov raziskave v realnem okolju je robot - muzejski vodnik.

Japonski projekt financirajo štiri večja japonska podjetja: telekomunikacijska družba NTT, Toshiba, Mitsubishi Heavy Industries in ATR, raziskovalno usmerjeno podjetje za telekomunikacijo in socialno robotiko.

3.2.3 PEIS ekologija

Projekt PEIS ekologija (ang. *PEIS - physically embeded intelligent systems ecology*) je bil razvit v okviru raziskovalnega projekta in sodelovanja med univerzo Orebro na Švedskem in ETRI (ang. *Electronic and Telecommunication Research Institute*) v Koreji. Projekt traja od oktobra 2004.

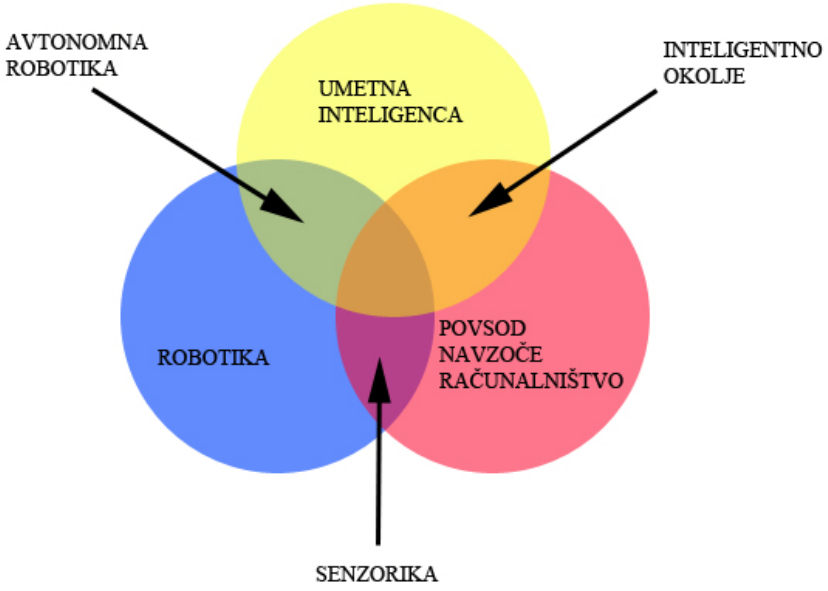
Pristop razvijalcev PEIS ekologije ni v izgradnji zelo naprednega in izoliranega robota, ampak v izgradnji omrežja integriranih inteligentnih naprav, ki komunicirajo med seboj. PEIS ekologija je sestavljena iz številnih preprostih robotskih delov.

Koncept

Rešitev, ki prihaja iz področja inteligentnega okolja je vzpostavitev omrežja inteligentnih naprav, ki so sposobne komunicirati z nami in med seboj ter opravljati razna opravila namesto nas. S tem namenom so raziskovalci predlagali koncept ekologije. Prednosti predlaganega sistema so v vzpostavitvi povezav med heterogeno skupino naprav [4].

V prihodnjih desetletjih bodo fizično integrirani inteligentni sistemi postali del našega vsakdana z namenom izboljšave kakovosti življenja, posebej tistim ljudem, ki potrebujejo pomoč v domačem okolju (npr. starejši občani, invalidi). Primeri takšnih sistemov so gospodinjske inteligentne naprave s pametnim uporabniškim vmesnikom in namenske naprave (roboti), ki so sposobne izvajati različna opravila. Robotske naprave oziroma PEIS enote so porazdeljene po delovnem prostoru v obliki senzorjev, stikal, pametnih naprav, enostavnih robotov itd. Vse PEIS enote komunicirajo in sodelujejo druga z drugo. V ekosistem lahko vključimo tudi človeka, ki predstavlja vrsto PEIS-a znotraj ekosistema. Razvoj PEIS ekologije temelji na povezovanju treh področij (slika 4):

- robotika,
- umetna inteligenca in
- povsod navzoče računalništvo (ang. *ubiquitous computing*).



Slika 4: študija PEIS ekolgije povezuje tri področja.

Posamezne PEIS enote temeljijo na tehnologiji moderne robotike. Plast umetne inteligence je potrebna za procesiranje negotovih informacij, saj v skladu z algoritmi spremeni vedenje robota glede na trenutne razmere in naloge. Z algoritmi umetne inteligence odkriva in odpravlja napake. Kognitivnost ni prisotna le znotraj posamezne PEIS enote, ampak se kot rezultat medsebojnega sodelovanja pojavlja v celotnem sistemu. Individualne PEIS enote so povezane v omrežni sistem. Preko mehanizmov, ki skrbijo za povezave in komunikacijo, množica PEIS enot predstavlja PEIS ekosistem.

PEIS ekologijo opišemo z naslednjimi definicijami [11]:

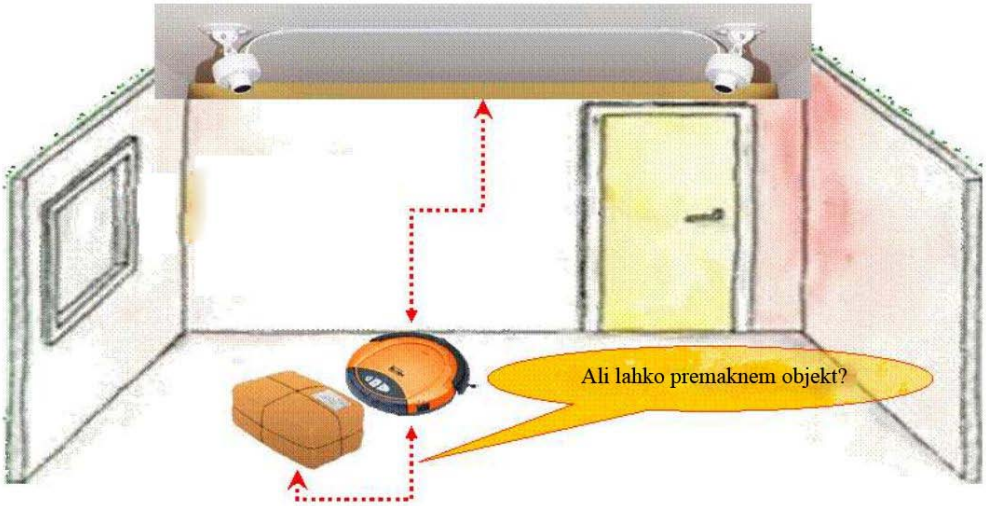
- Vsak robot v okolju je mišljen kot uniformna enota PEIS sistema. Izraz »robot« nastopa v svoji najbolj splošni definiciji kot naprava, ki omogoča računanje in komunikacijo ter vpliva na okolje z aktuatorji in zaznava le-tega s senzorji (ali stikali). PEIS enota je lahko kuhinjski opekač ali kompleksen humanoiden robot. Na splošno definiramo PEIS enoto kot sklop med seboj povezanih programskih plasti znotraj ene fizične enote.
- Vse PEIS enote so povezane z enotnim modelom komunikacije. V praksi sistem uporablja porazdeljen model komnunikacije z mehanizmom dogodkov.

- Vse PEIS enote v ekosistemu delujejo po enotnem modelu sodelovanja, ki temelji na povezovanju funkcionalnih komponent. Vsak PEIS v ekosistemu uporablja funkcije drugega PEIS-a (programske plasti, senzorji, aktuatorji), da dopolni svojo funkcionalnost. V tem primeru funkcionalnost definiramo kot modul, ki odda ali sprejme informacijo in pride v stik z okoljem preko senzorjev in aktuatorjev.

Na osnovi zgornjih definicij zaključimo, da je PEIS ekologija množica povezanih PEIS enot, ki so vgrajene v isto fizično okolje.

Za ilustracijo zgornjega koncepta je na sliki 5 prikazan avtonomni robotski sesalnik. Le-ta je PEIS enota, ki vsebuje funkcionalne komponente (sesanje, navigacija) in kot samostojna enota ne more izvesti zahtevnejših akcij, kot je zapletena strategija čiščenja večjega ali več prostorov.

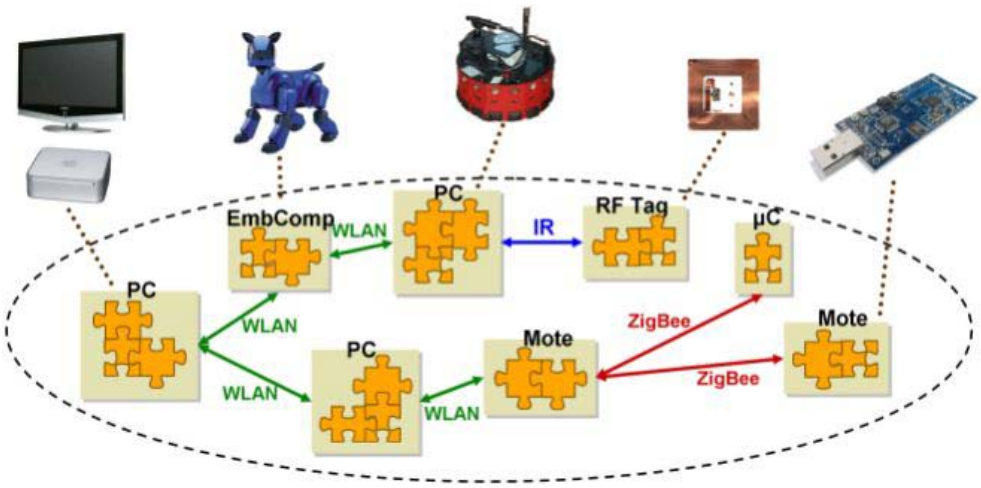
Z video nadzornim sistemom opremljen prostor nam zagotavlja slike iz statičnih kamer, razporejenih po celotnem prostoru. Z obdelavo slik določimo stanje okolja in natančno pozicijo sesalnika. Sesalnik in video sistem združimo v preprost PEIS ekosistem in dobimo funkcionalno globalno lokalizacijo sesalnika. Sesalnik tako izkorišča dodatne funkcije. Če sesalnik pripelje pred oviro (npr. roža), s pomočjo RFID oznake, ki jo ima roža, določi oziroma se odloči glede na njeno težo in velikost ali jo bo premaknil in počisti tla pod njo ali pa jo bo obšel. Ker je roža opremljena z oznako RFID, ki ima informacije o njeni vsebini, jo razumemo kot PEIS enoto in se lahko pridruži ekologiji, ter posreduje informacije robotu. Primer ilustrira sodelovanje več PEIS enot.



Slika 5: avtonomni robotski sesalnik v PEIS okolju.

Glede na dani PEIS ekosistem definiramo konfiguracijo okolja kot sklop komponent v ekosistemu skupaj z nizom povezav med njimi z namenom izvajanja določene naloge. Nad okoljem uporabimo tudi drugačno konfiguracijo, kar je odvisno od okoliščin, trenutne naloge, stanja okolja in razpoložljivih virov. V prejšnjem primeru uporabimo spremembo konfiguracije, ko robota ni več v vidnem področju kamer. Takrat robot uporabi ostale vire za lokalizacijo (npr. odometrični sistem ali kamero na sebi).

PEIS ekologija vključuje različne naprave (slika 6), ki temeljijo na različnih programskih platformah. Komunikacijski model omogoča izmenjavo velikega števila podatkov in možnost, da se lahko vsaka PEIS enota pridruži ali zapusti ekosistem v kateremkoli trenutku. PEIS jedro je skupna programska knjižica, ki je povezana s katerimkoli funkcionalnim delom vseh PEIS enot v sistemu. PEIS jedro ustvari in vzdržuje popolnoma decentralizirano omrežje ter dinamično usmerja sporočila med PEIS enotami. Jedro je prenosljivo med različnimi operacijskimi sistemi in deluje na različnih arhitekturah procesorjev in mikrokrmilnikov.



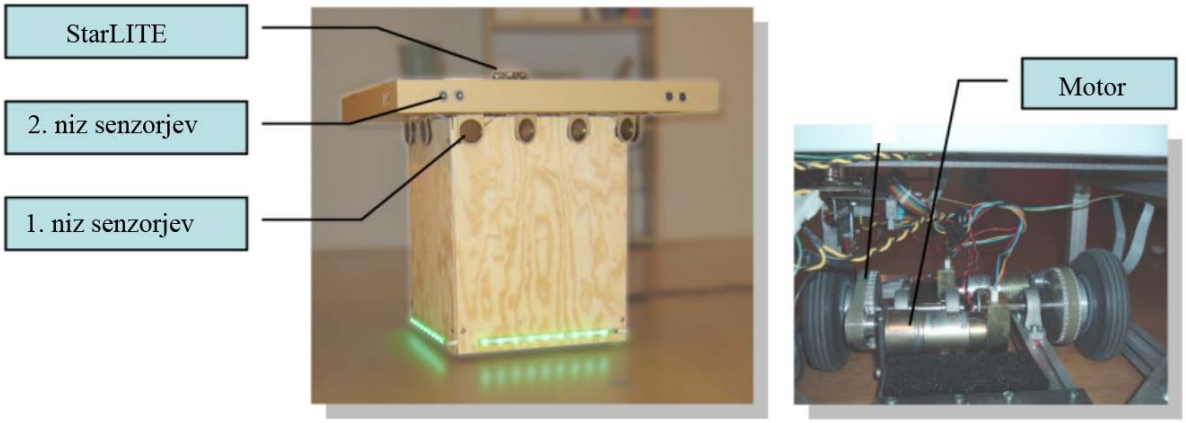
Slika 6: PEIS ekologija je heterogena glede na strojno opremo, programsko opremo in način komunikacij.

Primer:Avtonomna robotska mizica v domačem okolju

V naslednjem primeru opisujemo strojne in programske zahteve za avtonomno robotsko PEIS mizico, ki je del domačega gospodinjstva; opravlja avtonomno navigacijo v domačem okolju. PEIS mizica je del PEIS ekologije in skupaj z ostalimi PEIS enotami opravlja kompleksna vsakdanja opravila [12]. Koncept in realizacija tehnik odločanja in navigacije, prikazan na primeru PEIS mizice, je analogen tudi ostalim napravam s podobnimi zahtevami; v našem primeru imamo podobne zahteve pri avtonomnem robotskem sesalniku.

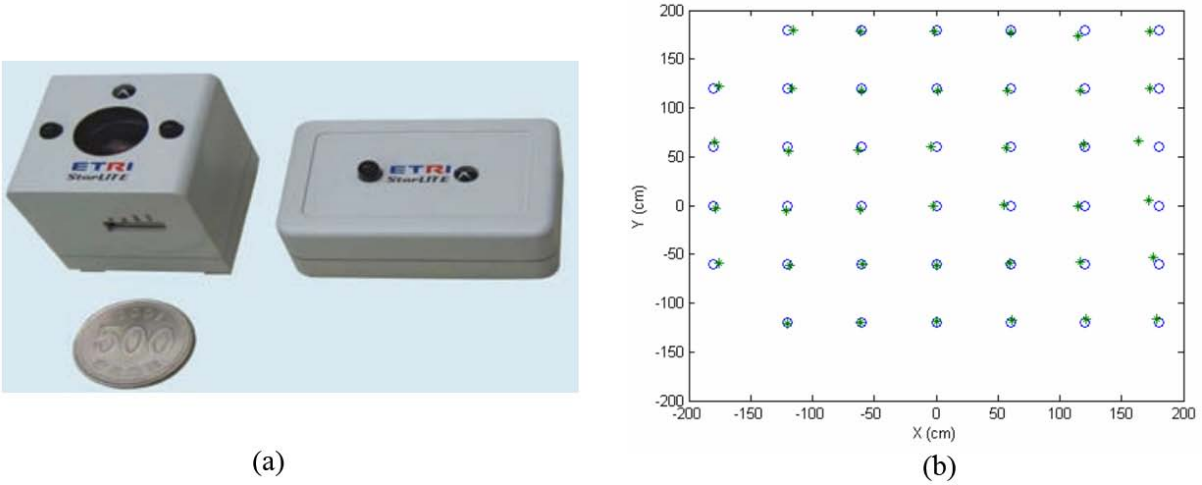
Delovanje PEIS mizice opisuje naslednji primer. Študent si zlomi nogo in je prisiljen počivati v postelji. PEIS ekologijo prosi, da mu prinese vodo. Mizica se premakne v kuhinjo k hladilniku, kjer hladilnik pošlje zahtevo za plastenko vode. Hladilnik odpre vrata in naloži plastenko na mizico, ta pa se vrne nazaj k postelji, da izpolni nalogo. V zgornjem primeru je vključeno sodelovanje treh PEIS enot, kamera, robotski hladilnik in avtonomna robotska mizica.

PEIS mizica je zgrajena iz komercialne lesene mizice (slika 7). Vanjo je vgrajen mobilni robot ActivMedia (AmigoBot) ter PC arhitektura z operacijskim sistemom Linux. Komunikacija poteka preko brezžičnega protokola 802.11. Na spodnjem robu mizice so montirane led diode, ki z barvo prikazujejo trenutni status mizice. Ob straneh sta dodani pogonski kolesi, vsako s svojim motorjem.



Slika 7: PEIS mizica.

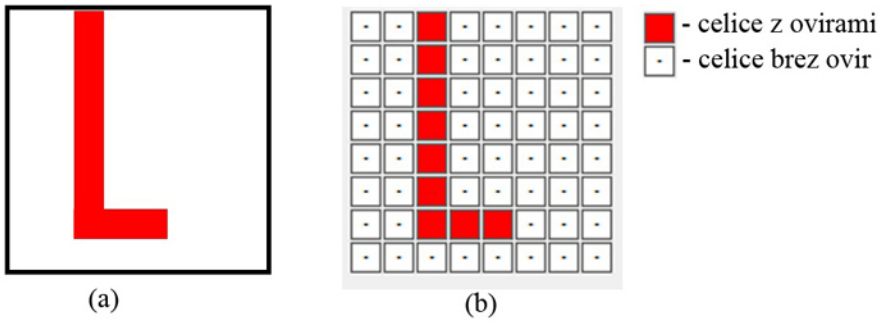
Sistem senzorjev je sestavljen iz dveh nizov. Prvi niz vsebuje osem senzorjev, ki zagotavljajo zanesljive meritve razdalje do 2,5 m. Primerni so za izgradnjo zemljevida okolja in izogibanje oviram. Drugi niz je sestavljen iz štirih ločenih ultrazvočnih senzorjev, ki merijo razdalje do 25 mm. V primerjavi s prvim nizom senzorjev ima drugi večjo natančnost pri manjših razdaljah. Drugi niz senzorjev se uporablja za popravljanje pozicije v primerjavi z referenčno, ko ima mizica zahtevnejše manevrske premike (npr. približevnje objektu).



Slika 8: a) StarLITE kamera in brezžična oznaka in b) primer pozicioniranja s sistemom StarLITE.

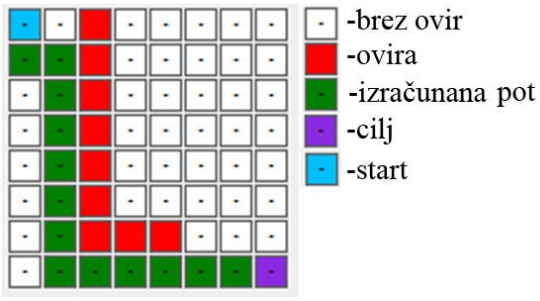
V PEIS okolje je za za natančno notranjo globalno lokalizacijo nameščen Sistem StarLITE. Sistem StarLITE [13] je razvil ETRI za namen navigacije robotov v notranjih prostorih. Sistem je sestavljen iz brezžičnih oznak (slika 8a, desno) in detektorja (slika 8a, levo). Detektor je IR kamera in modul za prepoznavanje oznak; nameščen je na robotu. Oznake so sestavljene iz IR LED diod. Pritrjene so na strop in imajo svojo lastno identiteto. Sistem zagotavlja pozicioniranje z natančnostjo ± 4 cm in ± 1° orientacije. Na sliki 8b je prikazan poizkus lokalizacije v sobi velikosti 4×4 m^2. Na stropu so nameščeni trije IR oddajniki, enakomerno oddaljeni 120 cm. Informacije o lokacijah so izračunane za mrežo točk, ki so enakomerno razporejene po prostoru z oddaljenostjo 60 cm ena od druge. Srednja napaka natačnosti pozicioniranja je 4.1 cm s standardnim odklonom 2.9 cm. Največja napaka znaša 17.1 cm (pozicija 180, 60) zaradi ukrivljenih tal v prostoru.

Mizica deluje v domačem okolju, ki se do določene mere dinamično spreminja. To povzroči številne zahteve glede navigacije in krmiljenja, ki ga izvedemo s programskim delom. Mizica mora biti varna med premikanjem, zato mora zaznavati ljudi (ali druge ovire) in se izogibati trkom. Vsebuje zemljevid okolja z ovirami in izvaja globalno lokalizacijo z namenom načrtovanja poti med dvema pozicijama v okolju. Mizica dinamično spreminja zemljevid z novimi ovirami. Okolje je diskretizirano (podpoglavje 4.2.3) v enakomerno razporejene celice. Ustrezna celica vsebuje informacijo o zasedenosti okolja (slika 9).



Slika 9: a) zemljevid okolja in b) diskretiziran zemljevid okolja.

Sistemu vnaprej posredujemo zemljevid, vključno s konstantnimi ovirami, s čimer robot že vnaprej določi topološko pravilne poti še preden razišče prostor. Modul za načrtovanje poti z algoritmom A* za iskanje poti (podpoglavje 4.2.4) izračuna najkrajšo pot med dvema točkama na osnovi diskretiziranega zemljevida realnega okolja (slika 10).



Slika 10: primer algoritma za iskanje poti.

Nadzor nad vedenjem mizice je realiziran z mehko logiko (podpoglavje 4.2). Profile različnih obnašanj opišemo z različnimi pravili. Na osnovi odčitkov senzorjev se glede na pravila določi premikanje mizice. Če je zaznana nevarna situacija, se mizica vede v skladu s pravili za izogibanje oviram. Avtorji so predvideli štiri glavna načine vedenja:

- premikanje po izračunani poti,
- izogibanje oviram,
- približevanje predmetom in
- oddaljevanje od predmetov.

Na sliki 11 je prikazana mizica pri navigaciji v sobi in pri približevanju objektu (robotski hladilnik).

Slika 11: PEIS mizica v realnem okolju.

PEIS ekologijo lahko obravnavamo kot ekologijo v skladu z običajnim pomenom tega izraza, tj. "študija odnosov med živimi organizmi in njihovim okoljem". Na podoben način je za PEIS ekologijo značilen tak odnos med enotami PEIS (skozi komunikacijo in sodelovanje) ter razmerja med enotami in njihovim okoljem (s pomočjo zaznavanja in delovanja). PEIS ekologija je modularna in prilagodljiva. Uporabniki po potrebi dodajajo nove robotske komponente, npr. začenši s preprostim robotskim sesalnikom, ki mu dodajajo nove PEIS enote v skladu s svojimi potrebami in željami. Tak pristop zagotovlja cenovno dostopno in sprejemljivo robotsko tehnologijo v vsakdanjem okolju. Ker vsaka nova PEIS enota združuje svoje funkcije z obostoječimi PEIS enotami v sistemu, je vrednost celotne PEIS ekologije večja od vsote vrednosti posameznih PEIS enot v njem.

4. Umetna inteligenca v robotiki

Načrtovanje poti je ena od osnovnih problematik v mobilni robotiki. Okolje robota pretvorimo v graf in uporabimo tehnike iskalnih algoritmov, ki temeljijo na sistematičnem preiskovanju grafov. Najbolj pogosto uporabljen je algoritem **A***, ki je nadgrajena različica **Dijkstrovega algoritma**.

Avtonomnost, racionalnost in prilagodljivost robota realiziramo z algoritmi odločanja. Naloga **algoritmov odločanja** je določanje najbolj primerne akcije v danem trenutku. Ena izmed tehnik odločanja temelji na **negotovem sklepanje** (ang. *uncertain reasoning*). V določenih situacijah je določen načrt ali sklep potrebno sprejeti kljub pomanjkanju informacij, ki bi sicer omogočile sklepanje in planiranje z zanesljivimi dejstvi. Pomanjkanju informacij pravimo tudi negotovost. Iz področja negotovega znanja in sklepanja se uporablja »**mehka logika«** (ang. *fuzzy logic*), kjer je vir negotovosti nedoločenost neke izjave v njenem opisu.

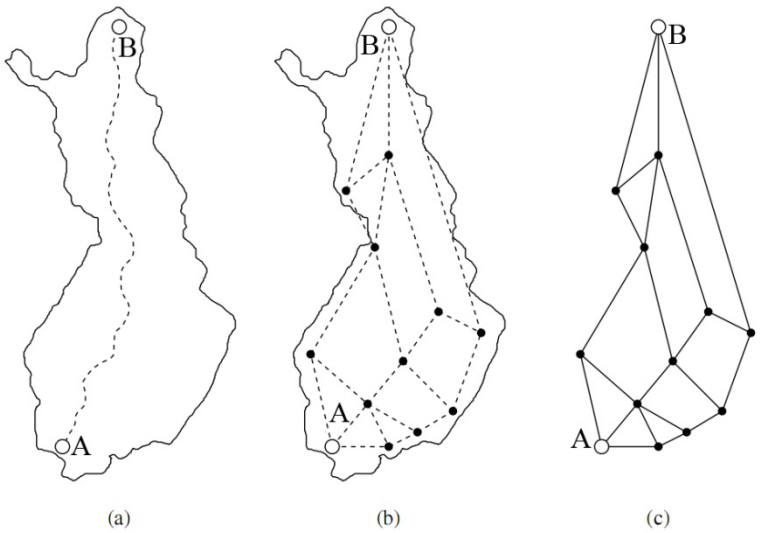
4.1 Iskanje poti

Ljudje znamo poiskati pot na zemljevidu brez težav, če le obstaja. Iskanje poti po zemljevidu je analogen problem navigaciji robota med fiksnimi točkami v prostoru, pri katerih želimo izvesti avtomatiziran premik robota. Primer predstavlja inteligentnega agenta, s katerim lahko predstavimo algoritem iskanja poti. Algoritem, ki išče pot, potrebuje precej procesorske moči za izračun rešitve, če je okolje kompleksno, obsežno, in se dinamično spreminja. Iskanje poti se izvrši v dveh korakih [14]:

- diskretizacija okolja,
- iskanje najkrajše poti.

Dano okolje (slika 12a) diskretiziramo oziroma razdelimo na končno število odsekov in vsakemu odseku dodelimo vozlišče (ang. *node*) (slika 12b). Če obstaja prehodna pot med sosednjim vozliščema, sta vozlišči med seboj povezani. Okolje, v katerem iščemo pot, tako transformiramo v graf (slika 12c) (pod-poglavje 4.1.2), ki nam problem opiše na matematično rigorozen način, primeren za izvedbo algoritmov iskanja.

Z dobljenim grafom in na njem osnovanem algortimu iščemo najprimernejšo pot od točke *A* do točke *B*.



Slika 12: a) pot v realnem okolju, b) diskretizacija okolja in c) transformacija okolja v graf.

4.1.1 Predstavitev problemov v prostoru stanj

Veliko problemov je možno definirati kot množico stanj, v katerih se problem nahaja, in akcij oziroma potez, ki predstavljajo prehode med različnimi stanji problema [2]. Med tovrstne probleme štejemo še:

- navigacijo,
- razni problemi na igralni plošči (sudoku, N kraljic, dama, ...),
- dokazovanje matematičnih teoremov,
- sestavljanje kompleksnih objektov iz elemntarnih sestavnih delov,
- optimizacija načrtov,
- spletno iskanje.

Prostor stanj je četverica $<N,A,S,G>$ kjer so:

- N množica vseh stanj, v katerih se problem lahko nahaja,
- A množica vseh akcij (potez, korakov, pravil), ki so dovoljene pri reševanju problema in spreminjajo trenutno stanje problema
- $S \in N$ začetno stanje problema in
- $G \subset N$ neprazna množica vseh končnih in ciljnih stanj problema.

Opis stanja je podatkovna struktura, ki vsebuje vse potrebne informacije za enolično določitev trenutnega položaja v postopku reševanja problema. Trenutno stanje je katerokoli stanje iz množice N. Pri tem ločimo delni in popolni opis stanja. Delno stanje predstavlja nepopolno

rešitev problema, ki jo dopolnjujemo v postopku reševanja problema. Popolna stanja so v celoti oblikovana stanja, ki pa ne izpolnjujejo nujno pogojev za ciljna stanja. V določenem stanju problema je dovoljena samo podmnožica vseh možnih akcij *A*, ki povzročajo spremembo trenutnega stanja in prehod v eno izmed »sosednjih« stanj. Namesto seznama akcij, ki so možne v nekem stanju $n \in N$, podajamo možne naslednike od *n* s pomočjo naslednostne funkcije (ang. *succesor function*), ki vrne množico vseh možnih stanj naslednikov stanja *n*. Začetno stanje problema, v katerem se nahajamo pred pričetkom reševanja, je poljubno, a je eno samo. Ciljna stanja so tista stanja problema, v katerih se reševanje problema zaključi.

Glede na vrsto problema ločimo naslednje načine podajanja ciljnih stanj:

- v obliki eksplicitnega opisa ciljnega stanja,
- v obliki pogojev, ki jih mora ciljno stanje izpolnjevati ali
- v obliki pogojev, ki jih mora izpolnjevati pot od začetnega do ciljnega stanja.

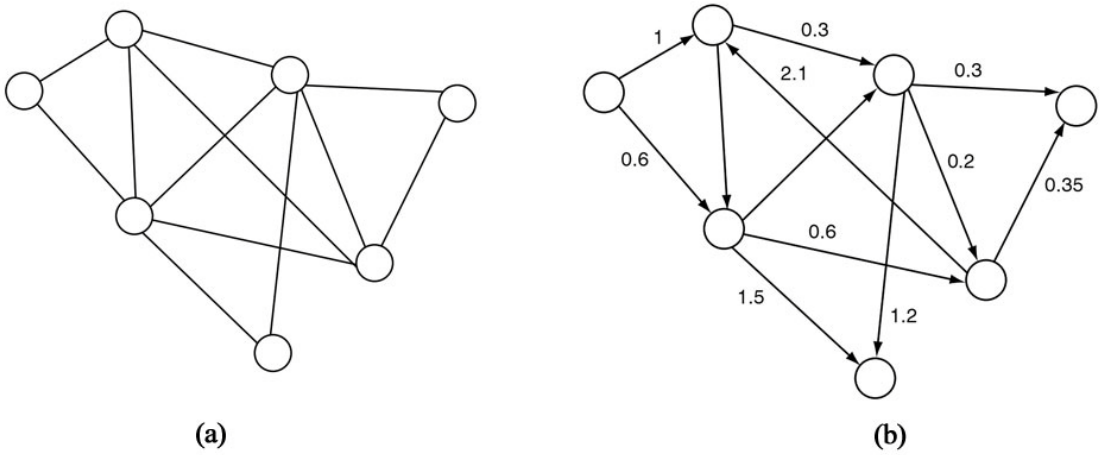
V prvo skupino spadajo problemi, katerih iskana rešitev predstavlja pot oziroma zaporedje akcij, ki vodi od začetnega do katerega izmed ciljnih stanj. Med omenjene probleme spada tudi iskanje poti med kraji na zemljevidu. Podana je mreža cestnih povezav med kraji na zemljevidu, prav tako izhodišče in cilj. Iščemo pot med njima kot zaporedje premikov v sosednje kraje. Opis stanja pri problemih iskanja fizične poti je kraj trenutnega nahajanja. Komponente opisa v prostoru stanj so:

- množica *N* vsebuje vse kraje v podanem omrežju,
- množica *A* vsebuje vse cestne povezave v podanem omrežju,
- začetno stanje *S* je kraj izhodišča in
- množica *G* vsebuje oznako kraja, kamor želimo priti.

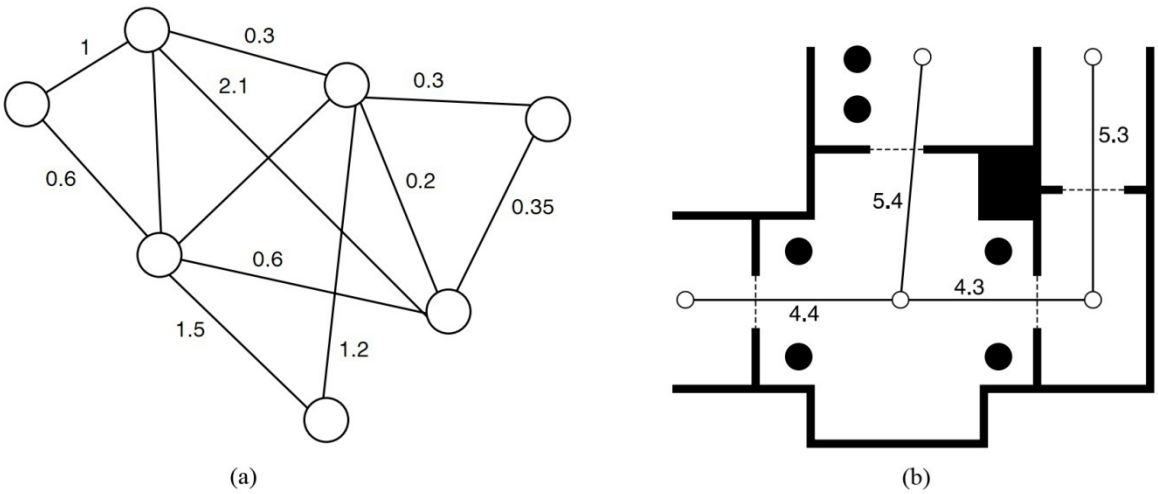
4.1.2 Iskalni Graf

Prostor stanj nekega problem najlažje predstavimo z vizualizacijo njegove topologije v obliki grafa. Stanje problema v tem primeru postane vozlišče grafa, akcije pa usmerjene povezave med vozlišči. Začetna in ciljna stanja problema ustrezno označimo. Reševanje problema si sedaj predstavljamo kot iskanje poti med začetnim in ciljnim vozliščem v grafu [2]. Za lažjo razlago Dijkstrovega in A* algoritma je nujno poznavanje osnovnih definicij o matematični teoriji grafov [15].

- Graf je urejena dvojica *(N,B),* kjer je $N = \{1,2,...,q\}$ neprazna končna množica vozlišč in $B = \{b_1,b_2,...,b_p\}$ množica povezav med vozlišč.
- Grafe grafično ponazorimo z diagrami (slika 13a).
- Vsaka povezava iz množice *B* je definirana s krajišči povezave, ki sta vozlišči iz množice *N*. Primer: oznaka $b_k=(i,j)$ pomeni, da obstaja povezava b_k med vozliščema *i* in *j*. Če velja *i=j*, pravimo povezavi zanka.
- Usmerjen graf je dvojica *(N,B),* kjer je $N = \{1,2,...,q\}$ neprazna končna množica vozlišč grafa in *B* enosmerna relacija nad $N \times N$. Elemente $B: b_1,b_2,...,b_p$, imenujemo povezave. Prvo krajišče povezave imenujemo začetek, drugo pa konec povezave. Povezava je usmerjena od začetka h koncu povezave (slika 13b).
- Graf je označen (utežen), če priredimo vsakemu vozlišču in/ali vsaki povezavi nek znak (utež) iz slovarja $V=\{v_1,v_2,...,v_n\}$. Znak je kvalitativnega ali kvantitativnega značaja (slika 14a).
- Vozlišči sta sosednji, če sta povezani s povezavo.
- Število sosedov vozlišča grafa imenujemo stopnja vozlišča *r(i),* kjer je $i \in N$.
- Sprehod v grafu je tako zaporedje vozlišč 1,2,...,*n,* da je vozlišče 1 sosed vozlišču 2, 2 sosed 3, ..., *n-1* sosed *n-ju*. Vozlišče 1 je začetek, vozlišče *n* pa konec sprehoda.
- Enostavni sprehod je sprehod, v katerem se vozlišča lahko ponavljajo, povezave pa ne.
- Pot je sprehod, v katerem so sama različna vozlišča grafa.
- Graf je povezan, če za poljubni vozlišči grafa obstaja sprehod, ki ima ti vozlišči za krajišči.



Slika 13: a) neusmerjen graf in b) usmerjen graf.



Slika 14: a) graf z uteženimi povezavami in b) predstavitev geometrije okolja z grafom.

Glede na smer preiskovanja grafa ločimo naslednje primere.

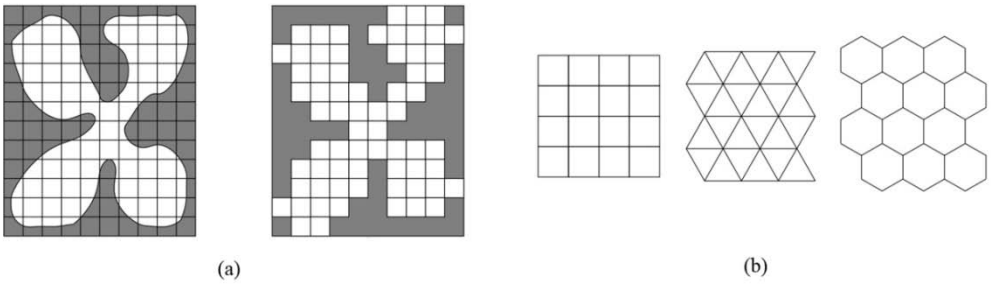
- Iskanje z veriženjem naprej (ang. *forward chaining*), pri katerem pričnemo z iskanjem v začetnem vozlišču in napredujemo v smeri proti ciljnemu vozlišču. Takšno iskanje imenujemo tudi podatkovno vodeno iskanje (ang. *data driven search*).
- Iskanje z veriženjem nazaj (ang. *backward chaining*), pri katerem pričnemo z iskanjem v končem vozlišču in razvijamo pot v smeri nazaj proti začetnem vozlišču. Za vsako vozlišče/stanje pri razvoju ugotavljamo, iz katerih stanj prispemo vanj. Ta stanja postanejo pod-cilji za iskanje, iz katerih se vračamo proti začetenemu vozlišču. Takšno iskanje imenujemo ciljno vodeno iskanje (ang. *goal driven search*).
- Iskanje z veriženjem v obeh smereh (ang. *bidirectional chaining*), pri katerem vzporedno izvajamo iskanje iz začetnega vozlišča proti končnemu in v obratni smeri ter iščemo stične točke obeh iskanj.

4.2.3 Diskretizacija območja

Pred iskanjem poti, je potrebno prostor diskretizirati. Trije najpogostejši načine diskretizacije so naslednji [16].

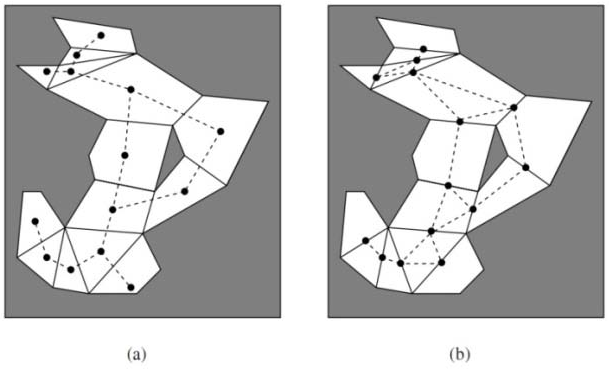
Vozliščna metoda (ang. *waypoints*) je najbolj enostavna diskretizacija. V danem okolju označimo vsak odsek kot vozlišče, poti med vozlišči pa utežimo z razdaljo (slika 14b). Tak pristop zahteva veliko »ročnega dela«, kjer moramo sami označiti poti in vozlišča. Slabost pristopa je, da se objekt giblje precej predvidljivo oziroma po »navidezni tirnici«. Na sliki 15b vidimo, da obstaja samo ena povezava med sosednjima vozliščema in objekt se vedno giba samo po eni možni poti [17].

Diskretizacija mreže : deli okolje (ang. *tessellation*) na enako velike, dotikajoče se mnogokotnike (slika 15a). Ob delitvi so označeni mnogokotniki, ki predstavljajo oviro. Središče vsakega mnogokotnika predstavlja vozlišče, povezave med vozlišči pa so odvisne od lastnosti sosednjih mnogokotnikov. Mnogokotniško delitev izvedemo z mrežo, ki vsebuje kot glavni element kvadrate, trikotnike ali šestkotnike (slika 15b) [16].



Slika 15: a) mnogokotniška delitev okolja in b) primeri mnogokotniških mrež.

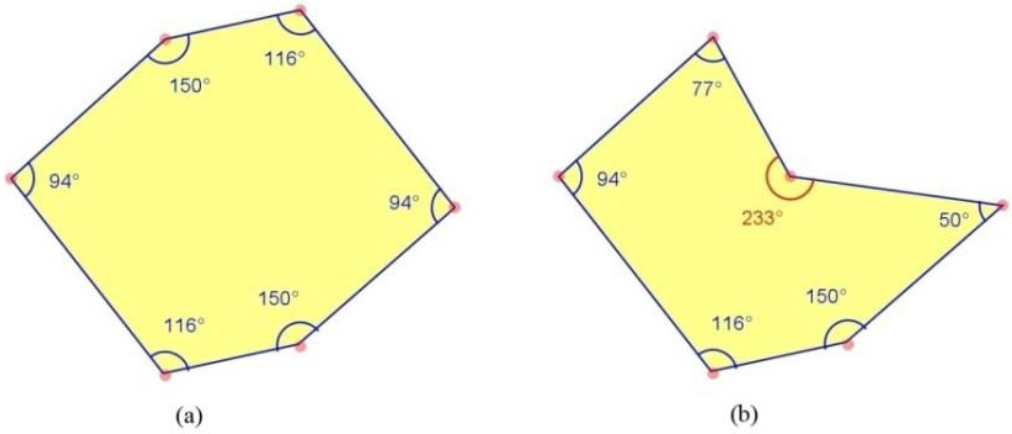
Diskretizacija okolja v navigacijsko mrežo (ang. *navigation mesh*): Za diskretizacijo uporabimo mnogokotniško mrežo okolja, ki jo z ustrezno obdelavo pretvorimo v navigacijsko mrežo. Ločimo mnogokotnike, ki predstavljajo ovire, in tiste mnogokotnike, ki predstavljajo podlago, po katerih se objekt premika. Vozlišče predstavlja središče mnogokotnika, ali pa središče roba mnogokotnika (slika 16) [14].



Slika 16: a) vozlišča v središču mnogokotnikov in b) vozlišča na središču stranic mnogokotnika.

Če je mnogokotnik konveksen, se lahko premaknemo v ravni črti znotraj mnogokotnika. Mnogokotniško mrežo obdelamo tako, da vse konkavne mnogokotnike pretvorimo v konveksno obliko (slika 17). To storimo s pomočjo algoritma, ki sta ga predlagala avtorja Hertel in Mehlhorn [16].

Za konveksen mnogokotnik velja, da izmed celotne množice notranjih kotov $A = \{\alpha_1, \alpha_2, \dots, \alpha_i\}$, ni večjega kota od $\alpha_i \leq 180°$ (slika 17a). Za konkaven mnogokotnik velja, da je izmed celotne množice notranjih kotov $B = \{\beta_1, \beta_2, \dots, \beta_i\}$, vsaj en kot večji od $\beta_i > 180°$ (slika 17b).



Slika 17: a) konveksen in b) konkaven mnogokotnik.

Hertel – Mehlhorn sta predlagala algoritem, ki s triangulacijo mnogokotnika (slika 18a) in zatem odstranjevanja nepotrebnih povezav v mnogokotniku določi minimalno število konveksnih mnogokotnikov, potrebnih, da pokrijejo izvorni mnogokotnik (slika 18b). Algoritem zagotavlja največ štirikrat večjo delitev mnogokotnika od optimalne delitve [16]. V naslednjem odstavku je prikazana psevdo koda Hertel – Mehlhorn algoritma.

Postopek

Trianguliraj mnogokotnik *P*.

Za vse diagonale $e_i \in P$, ki so ustvarjene s postopkom triangulacije **naredi**

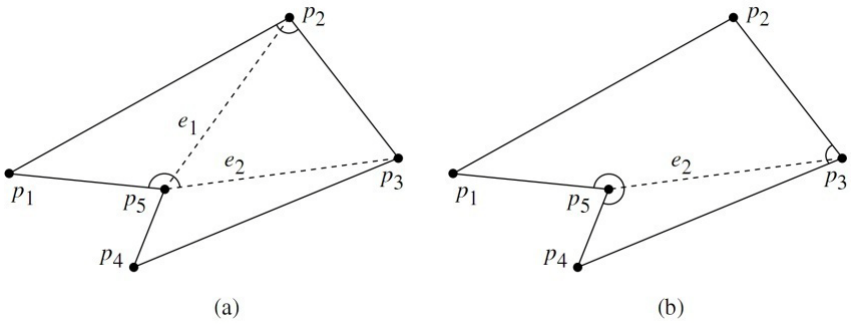
Če odstranitev diagonala e_i ne ustvari konkaven mnogokotnik v mnogokotniku *P* **naredi**

Odstrani diagonalo e_i;

Konec če

Konec za

Konec postopka



Slika 18: izvedba algortima Hertel – Mehlhorn nad mnogokotnikom.

Po triangulaciji mnogokotnika začne Hertel – Mehlhorn algoritem odstranjevati nepomembne diagonale (slika 18a) . Diagonala e_1 deli konveksna kota pri točkah p_2 in p_5. Ko diagonalo odstranimo, ima rezultirajoč mnogokotnik (p_1, p_2, p_3, p_5) lastnost konveksenega mnogokotnika (slika 18b).

4.2.4 Iskalni algoritmi

Pri implementaciji iskalnega algoritma smo pozorni na naslednji dve stvari [2].

- Zaželena lastnost algoritma je, da je terminalen, kar pomeni, da se algoritem prej ali slej zaključi. V neskončnih prostorih stanj je tej zahtevi načeloma nemogoče zadostiti, razen z eksplicitno omejitvijo dolžine poti. Tudi v končnih prostorih stanj preprečujemo iskanje v ciklih, ki podaljšujejo pot v neskončnost.
- Določena zaporedja akcij vodijo v nerešljive položaje, iz katerih se rešimo samo z razveljavljanjem že izvedenih akcij. Temu pravimo vračanje (ang. *backtracking*). Vračanje uporabimo tudi, če v postopku iskanja naletimo na omejitev dožine poti.

Če pri izbiri naslednjega vozlišča za razvoj uporabljajo problemsko specifično znanje »hevristiko«, delimo iskalne strategije na naslednje.

- Neformalne iskalne strategije (ang. *uninformed*) ne uporabljajo nobene dodatne informacije, ki bi jim pomagala pri določanju najbolšega naslednjega vozlišča za razvoj. Graf pregledujejo sistematično, v problemsko neodvisnem vrstnem redu, kar imenujemo slepo iskanje (ang. *blind search*).
- Formalne iskalne strategije (ang. *informed*). Zaradi velika števila vozlišč grafa je tudi časovna zahtevnost iskanja lahko velika. To neučinkovitost odpravimo z uporabo problemsko specifičnega znanja, kar imenujemo hevristično iskanje. Dobra hevristika usmerja iskanje proti cilju in zavrača tista nadaljevanja, ki niso obetavna.

4.2.5 Dijkstrov algoritem

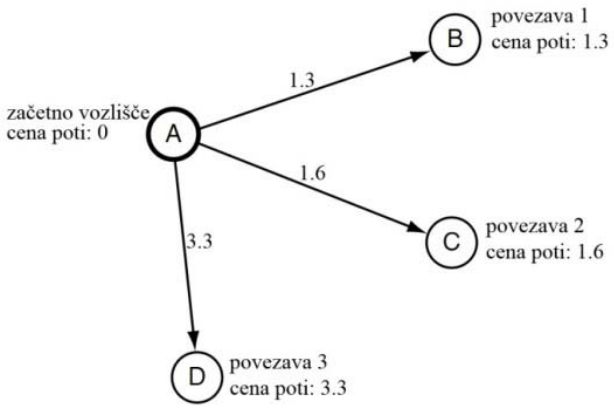
Dijkstrov algoritem je poimenovan po matematiku Edsgerju Dijsktri. Algoritem je bil razvit za reševanje najkrajše poti v teoriji matematičnih grafov in spada med neformalne iskalne strategije. Algoritem uporabljamo na različnih področjih, nekateri primeri uporabe so:

- za izračun najkrajše razdalje na zemljevidu,
- uporaba Dijsktrovega algoritma pri generiranju naključnih kromosomov,
- internetna protokola RFC 1142 in OSPF uporabljata Dijkstrov algoritem.

Za iskanje poti srečamo Dijsktrov algoritem redko, le v zelo kompleksnih simulacijah. V vseh ostalih manj zahtevnih problemih zadostuje spremenjen Dijkstrov algoritem, imenovan tudi algoritem A*. Dijsktrov algoritem je strategija iskanja z izbiro najboljšega. Naloga je, da v usmerjenem grafu z ne-negativnimi utežmi in začetnim ter končnim vozliščem v množici vozlišč poiščemo tako pot med dvema vozliščema, da bo imela najmanjšo ceno izmed vseh možnih poti. Zaradi lažjega razumevanja algoritma A*, v naslednjih odstavkih opisujemo postopek Dijkstrovega algortima [14].

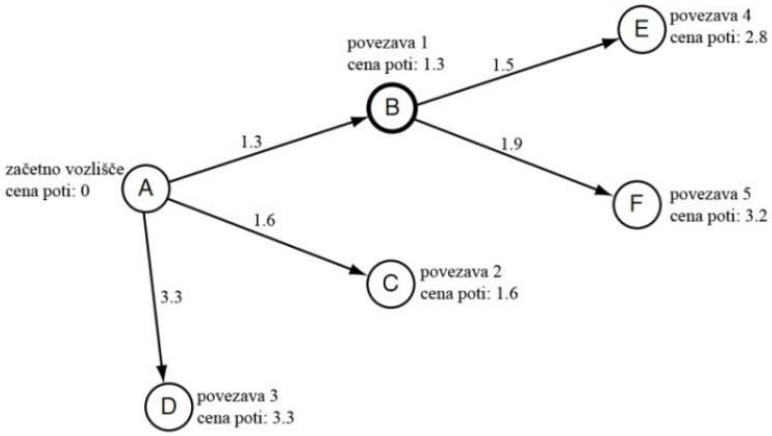
Vzemimo s_j kot poljubno vozlišče na najboljši poti med začetnim vozliščem s_Z in končnim vozliščem s_K. $g(s_j)$ je utež najboljše poti med začetkom poti in poljubnim vozliščem na najboljši poti s_j. Za vsako vozlišče na najboljši poti, razen za s_Z, velja $g(s_j)>0$. Naj bo c_{jk} utež povezave vozllišča s_j z njegovim naslednikom s_k [15]. Če označimo množico vozlišč z V in množico povezav z E v grafu G, je časovna zahtevnost algoritma enaka $O(|E|\log|V|)$ [18].

Dijkstrov algoritem začne graf analizirati v začetnem vozlišču s_z in nadaljuje analizo po celotnem grafu preko povezav do sosednjih vozlišč s_j. Za vsako vozlišče s_j shrani ceno poti $g(s_j)$ (ang. *cost-so-far*) od začetnega vozlišča s_z. Po celotni analizi grafa določi optimalno pot iz izhodiščnega s_z do končnega vozlišča s_k. Algoritem deluje iterativno. V vsaki iteraciji analizira vozlišče s_j in vse uteži c_{jk} povezav naslednikov (ang. *connection cost*), ki izhajajo iz izbranega vozlišča s_j. Algoritem opravi prvo iteracijo nad začetnim vozliščem s_z in izračuna ceno poti $g(s_j)$ do sosednjih vozlišč (slika 19).



Slika 19: prva iteracija Dijkstrovega algortima nad začetnim vozliščem A.

Za naslednjo iteracijo (slika 20) oziroma, za naslednje izbrano vozlišče s_j, izračuna zopet ceno poti c_{jk} med sosednjimi vozlišči. Cena poti, ki jo shranimo za vozlišče s_j, je seštevek vseh cen poti $g(s_j)$ iz začetnega vozlišča s_z.

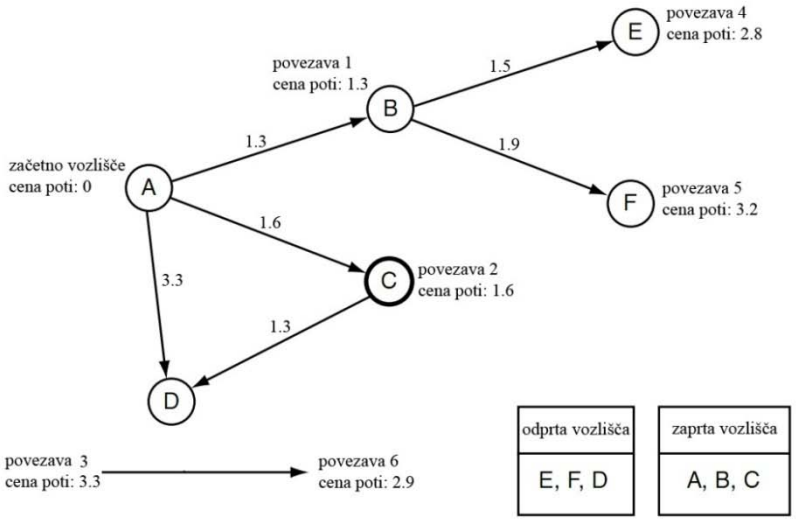


Slika 20: druga iteracija Dijkstrovega algortima: izbrano je vozlišče B z najmanjšo ceno poti.

Algoritem vsebuje seznam vozlišč. Ta so razdeljena na seznam odprtih »OPEN« in zaprtih »CLOSED« vozlišč (slika 21). Na seznamu odprtih vozlišč so vsa tista, ki še niso imela svoje iteracije, vendar jih je algoritem že analiziral v iteraciji enega izmed sosednjih vozlišč. Na seznamu zaprtih vozlišč, so vsa tista, ki so že imela svojo iteracijo. Za vsa ostala vozlišča pravimo, da so neobiskana. Postopek se začne s prvo iteracijo nad začetnim vozliščem A, in doda na seznam »OPEN« vozlišča B, C in D, ki so direktni nasledniki vozlišča A (slika 19). Algoritem si za naslednjo iteracijo vedno izbere vozlišče iz odprtega seznama in za obravnavano vozlišče doda na seznam »OPEN« vsa vozlišča, ki so direktni nasledniki. Prvo izbrano vozlišče iz seznama »OPEN« je vedno tisto vozlišče, ki ima najmanjšo ceno poti $g(s_j)$, zato na sliki 20 izbere za naslednjo iteracijo vozlišče B. Iskanje z izbiro najboljšega ali iskanje

najboljši naprej (ang. *best-first search*) je splošno ime za iskalne strategije, pri katerih na vsakem koraku razvijemo vozlišče s_j z najmanjšo ceno $g(s_j)$ v seznamu »OPEN«. To dosežemo tako, da je seznam »OPEN« v vsakem trenutku urejen po naraščajočih vrednostih g.

Ne velja vedno, da bodo vsa vozlišča, ki so nasledniki določene iteracije, neobiskana. Če v določeni iteraciji dobimo povezavo z vozliščem, ki je že na odprtem seznamu, pomeni, da smo tako vozlišče že analizirali v eni izmed predhodnjih iteracij. V takem primeru dodelimo odprtemu vozlišču, tisto ceno poti, ki je manjša (slika 21). Nad vozliščem C opravljamo iteracijo, zato dodamo vsa vozlišča, ki so nasledniki, na seznam »OPEN«. D je že na seznamu prejšnje iteracije in ker je $g(C)<g(D)$, nadgradimo vozlišče D z novo vrednostjo in potjo.



Slika 21: nadgradnja vozlišča iz seznama »OPEN«.

Iskanje poti se zaključi, ko je seznam »OPEN« prazen. To pomeni, da je algoritem opravil iteracije nad vsemi vozlišči in jih prestavil v seznam »CLOSED«. V analizi nas zanimajo tiste poti, ki vodijo do končnega vozlišča. Pot, ki je naboljša, ima najmanjšo ceno poti. Izbrano pot od končnega do začetnega vozlišča identificiramo z vzvratnim sledenjem shranjenih povezav za vsako vozlišče [14].

4.2.6 Algoritem A*

Problem, ki ga rešuje algoritem A*, je identičen problemu, ki ga rešuje Dijkstrov algoritem. Algoritem spada med formalne iskalne strategije. Pri iskanju uporablja specifično znanje oziroma hevristiko. Rešitve algoritma A* imajo večjo ceno poti v primerjavi z Dijsktrovim algoritmom vendar je časovna zahtevnost iskanja manjša.

Kot v prejšnjem poglavju vzemimo, da je s_j poljubno vozlišče na najboljši poti med začetnim vozliščem s_Z in končnim vozliščem s_K. Naj bo utež (cena) najboljše poti med s_Z in s_K vsota najboljše poti med s_Z in s_j ter med s_j in s_K [15]:

$$f(s_j)=g(s_j)+h(s_j) \tag{4.1}$$

kjer so:

- $g(s_j)$*:* utež najboljše poti med začetkom poti in poljubnim vozliščem na najbolši poti s_j. Za vsako vozlišče na najboljši poti, razen za s_z, velja $g(s_j)>0$,
- $h(s_j)$*:* Osnovna komponenta ocenitvene funkcije f je hevristična ocenitvena funkcija (ang. *heuristic*) $h(s_j)$, ki podaja oceno cene poti od vozlišča s_j do cilja. Hevristična funkcija je lahko poljubna, omejitev je le, da za vsak s_j, ki je ciljno vozlišče, velja $h(s_j) = 0$.
- $f(s_j)$*:* utež najbolše poti, ki je prisiljena iti skozi vozlišče s_j.

Funkcija $g(s_j)$ vzdolž poti monotono narašča, $h(s_j)$ pa monotono pada. Funkcijo $g(s_j)$ (utež najboljše poti med s_Z in s_j) izračunamo kot seštevek uteži povezav vozlišč na poti s_Z in s_j. Funkcije $h(s_j)$, ki je utež optimalne poti med s_j in s_K,v splošnem ne poznamo. Ocena funkcije $h(s_j)$ je odvisna od naloge, ki jo rešujemo. Neodvisno od naloge pa zahtevamo, da je ta ocena najboljša, to je, da velja:

$$\hat{h}(s_j) - \hat{h}(s_k) \leq c_{jk}\,, \tag{4.2}$$

kjer je c_{jk} utež (cena, dolžina) povezave vozlišča s_j z njegovim naslednikom s_k. Iz te zahteve sledi, da ocena uteži najboljše poti med vozliščema s_j in koncem poti s_k ne more biti večja od prave vrednosti.

V algoritmu A* zato izračunamo utež najboljše poti iz s_Z v s_K, ki je prisiljena iti skozi vozlišče s_j, kot vsoto uteži najboljše poti med s_Z in s_j ter ocene uteži poti med s_j in s_K, to je:

$$\hat{f}(s_j) = g(s_j) + \hat{h}(s_j), \tag{4.3}$$

kjer je:

$$\hat{f}(s_j) \leq g(s_j). \tag{4.4}$$

Postopek

Vpiši začetno vozlišče poti s_Z v seznam OPEN in izračunaj fukcijo $\hat{f}(s_Z)$. Odpri seznam CLOSED, ki je na začetku prazen.

dokler »seznam OPEN ni prazen«, **naredi**

»prenesi s seznama OPEN na seznam CLOSED vozlišče s_j, ki ima med vsemi vozlišči na seznamu OPEN najmanjšo vrednost $\hat{f}(s_j)$«;

če »s_j je končno vozlišče s_K«, **potem**

»končaj postopek. To je uspešen konec postopka. Vozlišča, ki sestavljajo iskano pot, so na seznamu CLOSED. Pot od končnega do začetnega vozlišča vzpostavimo z vzvratnim sledenjem postavljenih kazalcev«

sicer

»Določi potomce vozlišča s_j in za vse potomce izračunaj funkcije $\hat{f}$. Vpiši v seznam OPEN vse tiste potomce, ki še niso bili vpisani na seznam OPEN ali na seznam CLOSED, in postavi njihove kazalce tako, da kažejo na s_j. Za vse potomce, ki so že v seznamu OPEN (pomeni, da imajo le-ti več prednikov), preusmeri kazalce na vozlišče s_j le v primeru, da je funkcija $\hat{f}$ sedaj manjša od prej izračunane funkcije $\hat{f}$,«

konec-če

konec dokler

končaj postopek. To je neuspešen konec postopka, ker pot med začetnim in končnim vozliščem v grafu ne obstaja.

Konec potopka

Funkcijo $h(s_j)$ ocenimo na več načinov. V simulacijah je najbolj pogost način ocenjevanja z Evklidovo, Manhattan in Čebiševo razdaljo. Glede na izbran način merjenja je razdalja $h(s_j)$ najkrajša razdalja med poljubno izbranim vozliščem s_j in končnim vozliščem s_K [17].

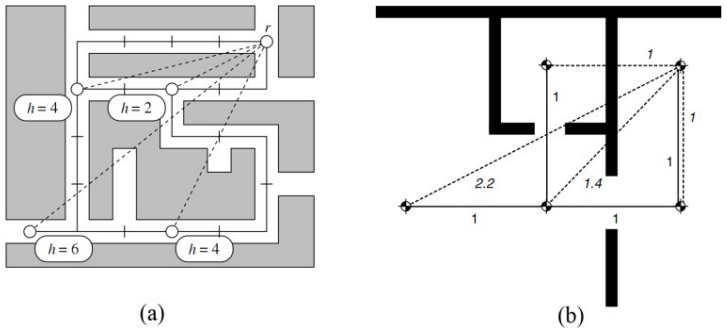
- **Razdalja Manhattan** je med dvema točkama definirana kot vsota razdalj vzdolž osi, ki sta si med seboj pravokotni. Slika 22-a prikazuje merjenje hevristične funkcije z

razdaljo Manhattan med dvema vozliščema. Enačba za razdaljo med točkama $X = \{x_1, y_1\}$ in $Y = \{x_2, y_2\}$ je:

$$D = |x_1 - x_2| + |y_1 - y_2| \tag{4.5}$$

- **Evklidova razdalja:** Slika 22b prikazuje merjenje hevristične funkcije $h(s_j)$ med dvema vozliščema. Enačba za razdaljo med točkama $X = \{x_1, y_1\}$ in $Y = \{x_2, y_2\}$ je:

$$D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{4.6}$$

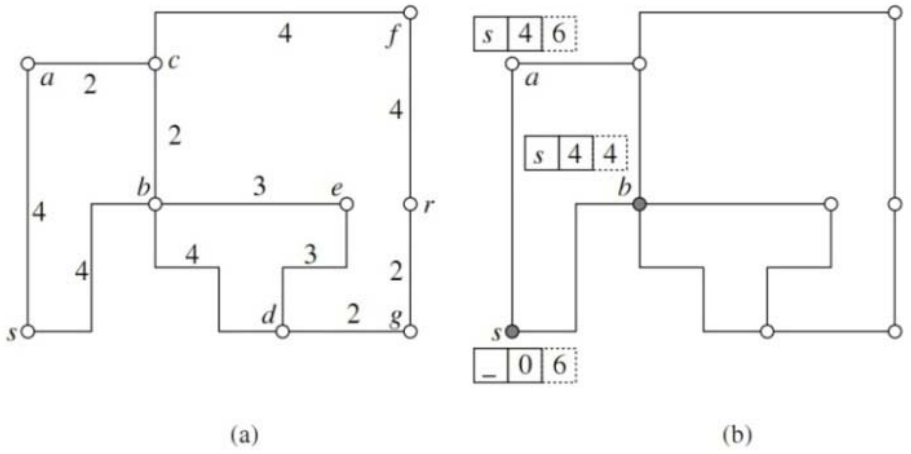


Slika 22: a) h(sj) z razdaljo Manhattan in b) h(sj) z Evklidovo razdaljo.

- **Čebiševa razdalja**: Enačba za razdaljo med točkama $X = \{x_1, y_1\}$ in $Y = \{x_2, y_2\}$:

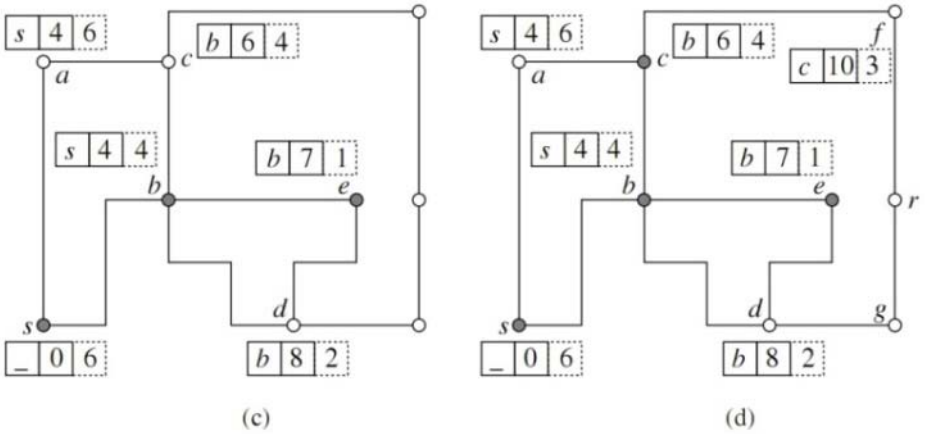
$$D = \max(|x_1 - x_2|, |y_1 - y_2|) \tag{4.7}$$

Naslednji primer prikazuje delovanje algoritma A*. Povezave do vozlišč in hevristična funkcija so utežene glede na razdaljo Manhattan. Naloga je, da pridemo iz začetnega vozlišča *s* v končno vozlišče *r* (slika 23a). Algoritem A* v prvi iteraciji analizira vozlišče *s* in povezave obravnavanega vozlišča do ostalih vozlišč, to sta vozlišči *a* in *b* (slika 23b). Sedaj sta vozlišči *a* in *b* dodani na seznam »OPEN«. Algoritem A* določi hevristično funckijo tako, da izmeri horizontalne povezave do končnega vozlišča. V naslednji iteraciji vzamemo vozlišče iz seznama »OPEN«, to sta vozlišči *a* in *b* in za obe vozlišči določimo hevristično funkcijo. Ker po enačbi 4.1 velja $f(b)<f(a)$, si izberemo iteracijo nad vozliščem *b* (slika 23b).



Slika 23: primer algoritma A*.

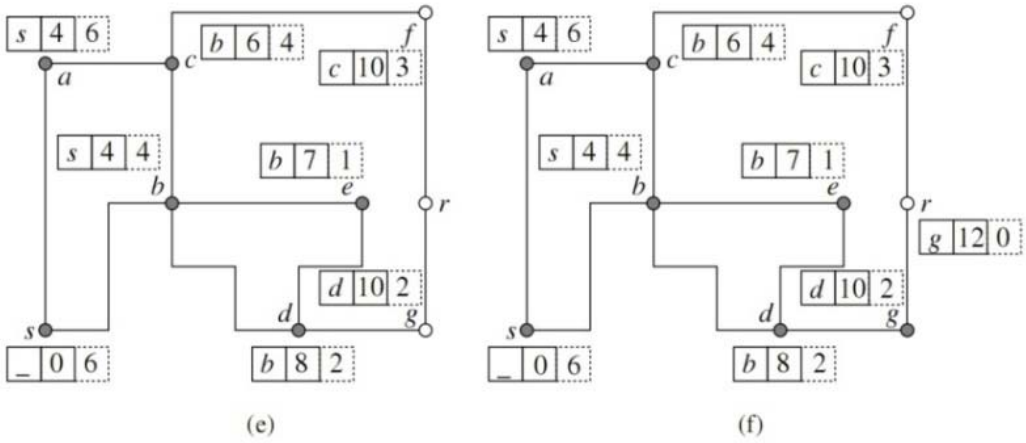
Zaradi iteracije vozlišča b dodamo na seznam »OPEN« vozlišča *c, d* in *e* (slika 24a). Sedaj imamo na seznamu »OPEN« vozlišča *a, c, d* in *e*. Naslednjo iteracijo izvedemo nad vozliščem, ki ima najboljšo oceno poti $f(s_j)$ iz seznama »OPEN«, zato izberemo vozlišče *e* (slika 24b). Vozlišče *e* nima ne-obiskanih naslednikov, ker je vozlišče *d* že na seznamu »OPEN«, zato izberemo drugo vozlišče iz seznama. Vsa ostala vozlišča na seznamu imajo isto vrednost funkcije $f(s_j)$, zato si izberemo poljubno vozlišče, naprimer *c* (slika 24b). Iteracija vozlišča *c* doda na seznam »OPEN« vozlišče *f*.



Slika 24: primer algoritma A*.

V naslednji iteraciji si zopet izberemo najbolj primerno vozlišče iz seznama »OPEN«. Izbiramo med *a, d* in *f*. Najboljšo oceno funkcije $f(s_j)$ ima vozlišče *a*, vendar tako kot pri vozlišču *e*, ima *a* za svojega naslednika vozlišče, ki je že obiskano (slika 25a). Naslednje najbolj primerno vozlišče za iteracijo je *d*. Zaradi iteracije vozlišča *d*, dodamo na seznam »OPEN« še vozlišče *g*, ki ima boljšo oceno poti kot vozlišče *f*, ki je še ostalo na seznamu

»OPEN«. Iz vozlišča *g* pridemo v končno vozlišče *r* (slika 25b). Po končani iteraciji, je bilo vsako vozlišče dodano na seznam »CLOSED«. Pot od končnega do začetnega vozlišča vzpostavimo z vzvratnim sledenjem postavljenih kazalcev [16].



Slika 25: primer algoritma A*.

Učinkovitost iskanja *algoritma A** je zagotovljena, če so izpolnjeni naslednji pogoji:

- Hevristična ocena $h(s_j)$ je sprejemljiva, kar pomeni, da nikoli ne preceni dejanske cene do cilja. Če je $h^*(s_j)$ minimalen dejanska cena poti od vozlišča s_j do cilja, potem mora za vse *n* veljati $h(s_j) \leq h^*(s_j)$.
- Beležimo najkrajšo pot do vsakega vozlišča v *open* ali *closed* seznamu. Če do nekega vozlišča s_j v seznamu *open* pridemo po novi poti, ki je krajša od zabeležene, jo popravimo in ohranimo samo en zapis vozlišča v *open*. V primeru, da je takšno vozlišče s_j že v seznamu *closed*, pot rekurzivno popravimo tudi vsem njegovim naslednikom.

Učinkovitost iskanja *algoritma A** je možno izpolniti tudi tako

- Zagotovimo, da do vsakega vozlišča pridemo najprej po najkrajši poti. To nam zagotavlja konsistenca oziroma monotona hevristična ocena, za katero velja $h(s_j) \leq c(s_j, s_j') + h(s_j')$ za vsa vozlišča s_j in njihove naslednike s_j'. Pri zgornji zahtevi gre za posebno obliko trikotniške neenakosti, saj zahtevamo, da je ocena poti od nekega vozlišča s_j do cilja manjša ali enaka vsoti cene poti do kateregakoli naslednika s_j' od s_j in hevristične ocene od s_j' do cilja. Vsaka konsistentna hevrsitika je hkrati tudi sprejemljiva, medtem ko obratno ne velja.

Učinkovitost algoritma A*, ki jo merimo v številu razvitih vozlišč, izboljšamo z uporabo močnejše hevristike. Slabost algoritma A* je, da število znotraj preiskovalne meje še vedno raste eksponentno z dolžino rešitve. Zato je v nekaterih problemih bolje iskati rešitev, ki je samo blizu optimalne in ne vztrajati na optimalni rešitvi. Še bolj kot časovna je problematična prostorska zahtevnost algoritma A*, saj ta hrani vsa razvita vozlišča v pomnilniku. Zato so bile razvite izboljšave A*, ki so prostorsko bolj učinkovite [2]:

- Algoritem A* s postopnim poglabljanjem (ang. *iterative deepeing A* -IDA**) – uporablja vedno višjo mejo za oceno *f*, kar pa ni praktično za realne vrednosti cene,
- Rekurzivno iskanje z izbiro najboljšega (ang. *recursive best-first search – RBFS*): je varianta algoritma A* s prostorsko zahtevnostjo, ki je linearna v številu razvitih vozlišč. Algoritem deluje tako, da si zapomni oceno druge najboljše poti za razvoj. Če trenutno vozlišče preseže shranjeno oceno najboljše alternative, se iskanje rekurzivno odvije (razvita vozlišča se brišejo iz pomnilnika) in vsem vozliščem se po poti nazaj priredi najboljša ocena *f* od njihovih naslednikov.

4.2 Mehka logika

Osnovo mehke logike predstavlja teorija mehkih množic, ki je zasnovana v šestdesetih letih [*Zadeh, 1965*]. V začetku je bila deležna obsežnih kritik, v katerih je prednjačil tedaj že uveljavljeni Kalman, ki enostavno ni videl smisla v "poenostavljenem" obravnavanju problemov, saj natačnost res ne more in ne sme "škodovati" pri njihovem reševanju. V sedemdesetih letih pa se je najbolj po zaslugi japonskih raziskovalcev teorija uveljavila, predvsem na področju procesne kontrole, procesiranja signalov, telekomunikacij, medicine, pa tudi financ, trgovine in sevisiranja. Danes so njeni rezultati prisotni v večini sodobnih produktov bele tehnike, audio-video multimedijskih naprav, optičnih in telekomunikacijskih izdelkov itd. Mehki sistemi omogočajo opisovanje zapletenih problemov z simboličnimi (lingvističnimi) izrazi, podobno, kot jih v naravnem jeziku opisuje človek. Skupaj s postopki mehkih pravil, mehkega sklepanja, odločitev in krmiljenja predstavljajo nov način reševanja problemov, ki sledi človeški obravnavi, sklepanju in odločanju [19].

4.2.1 Mehke množice

Izjavna in predikatna logika operirata s stavki, ki zasedejo le dve logični vrednosti – *true* ali *false*. Tudi verjetnosti *P(a)* neke negotove izjave *a* ne vpliva na njeno veljavnost. Ta je še

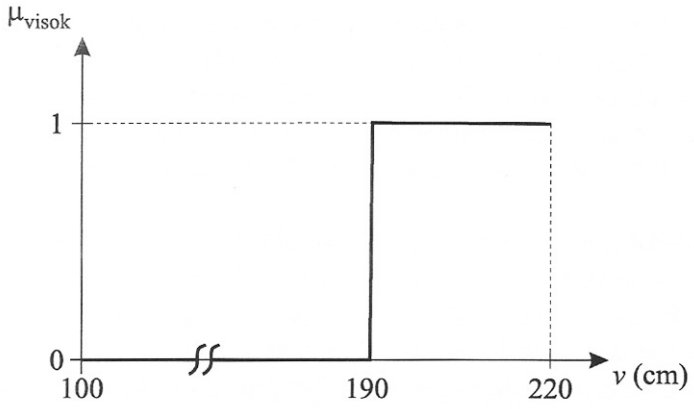
vedno lahko samo resnična ali neresnična. V realnem svetu je mogoče le malo stvari označiti s tema skrajnima vrednostima. Vzemimo primer osebe *A* z višino 180 *cm*. Kako bi ovrednotili resničnost izjave "Oseba *A* je visoka."? Odgovor na to vprašanje je seveda subjektiven in odvisen od tega, kaj za nekoga pomeni lastnost "biti visok". K negotovosti pripomore tudi naravni jezik, ki je včasih premalo natančen in dvoumen, kot na primer v izjavi "Oseba *A* je še dokaj mlada.". Kljub temu znamo ljudje s takšnimi izjavami dobro operirati, se sporazumevati ter z njimi izvajati sklepanje [2].

Binarna logika je omejeno uporabna za sklepanje v domenah, kjer nastopajo razmerja in lastnosti, kot je zgoraj opisana višina. V tem podpoglavju opisujemo obliko logike, ki operira z izjavami, ki so lahko le delno resnične oziroma veljavne. Govorimo o negotovem sklepanju, vir negotovosti je nejasnost izjave (ang. *vagueness*) oziroma nedoločenost (ang. *fuzziness*) v njenem opisu. V slovenščini se je za to vrsto logike uveljavil izraz mehka logika (ang. *fuzzy logic*).

V primeru telesne višine definirajmo množico visokih oseb z M_{visok}. Za elemente te množice imamo vse osebe, katerih višina *v* je večja ali enaka določeni mejni višini *m*. To pomeni, da oseba *A* pripada množici M_{visok} natanko tedaj, ko velja $v(A) \geq m$. Denimo, da je izbrana mejna višina m_v = 190 *cm*. Oseba *A* z višino $v(A) = 189\,cm$ potem ne pripada množici visokih ljudi, medtem ko osebo *B* z višino $v(B) = 190\,cm$ štejemo med visoke ljudi. Značilnost klasične teorije množic je, da objekti pripadajo neki množici v celoti (pripadnost *1*) ali pa ji sploh ne pripadajo (pripadnost *0*).

Stopnjo vključenosti elementov *x* iz domene *X* v množico *M* lahko podamo v obliki pripadnostne funkcije (ang. *membership function*) $\mu_M(x): R \rightarrow \{0,1\}$. Pripadnostna funkcija preslika numerično vrednost opazovane lastnosti v pripadalnost ustrezni množici. Slika 26 prikazuje stopničasti graf pripadnostne funkcije za množico M_{visok} v odvisnosti od telesne višine, ki mu pripada naslednji funkcijski zapis:

$$\mu_{visok}(x) = \begin{cases} 0, & če\ v(x) < m \\ 1, & sicer \end{cases} \tag{4.8}$$

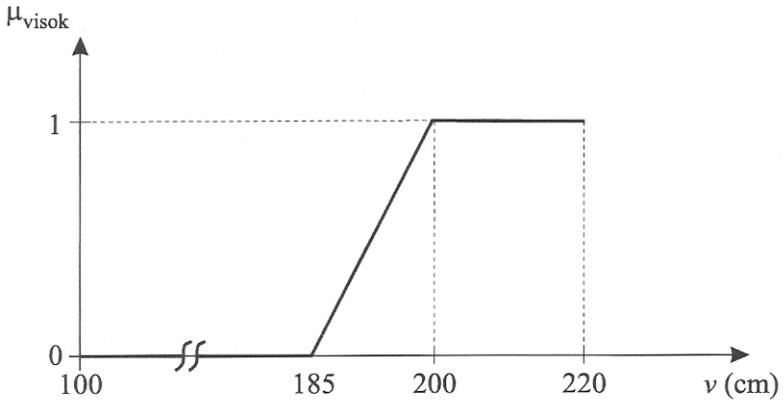


Slika 26: graf pripadnostne funkcije za klasično množico visokih ljudi.

Predstavitve visokih ljudi s klasično množico realno ni sprejemljiva. Osebi *A* in *B*, ki se v višini razlikujeta za praktično zanemarljiv 1 cm, se po pripadnosti visokim razlikujeta za maksimalnih *1*. Po drugi strani med osebo *B* in osebo *C* z višino $v(C) = 220\,cm$ v pripadnosti visokim ni nobene razlikem, čeprav je višinska razlika med njima kar *30 cm*. Tudi če postavimo katerokoli drugo mejno višino, s tem težave ne rešimo. Potrebujemo množice, katerih elementi so le delno vključeni oziroma imajo delno pripadnost. Takšnim množica pravimo mehke množice (ang. *fuzzy sets*).

Za mehke množice je značilno, da je zaloga vrednosti pripadnostne funkcije celoten realni interval [0,1] in ne njegovi skrajni vrednosti, tj. $\mu_M(x): R \rightarrow \{0,1\}$. Slika 27 prikazuje novo obliko pripadnostne funkcije za množico visokih ljudi, ki vključujejo dve mejni višini – spodnjo mejo m_1 = 185 cm in zgornjo mejo m_2 = 200 cm. Njen funkcijski zapis je naslednji:

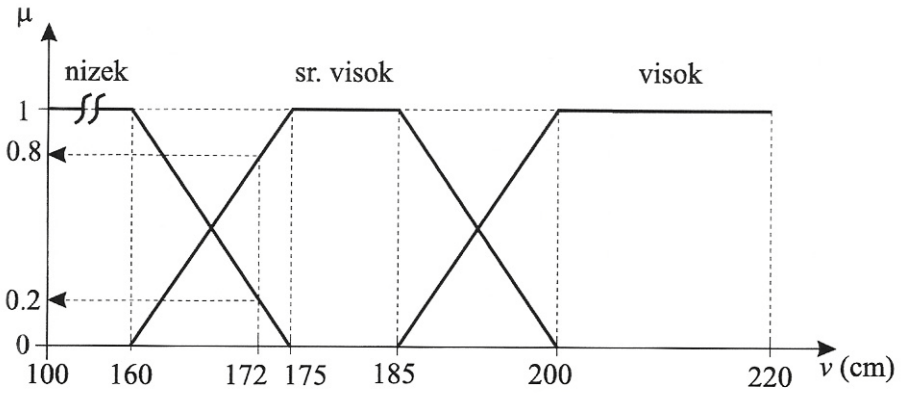
$$\mu_{visok}(x) = \begin{cases} 0, & \check{c}e\ v(x) \leq m_1 \\ \frac{v(x)-m_1}{m_2-m_1}, & \check{c}e\ m_1 < v(x) \leq m_2 \\ 1, & sicer \end{cases} \qquad (4.9)$$



Slika 27: graf pripadnostne funkcije za mehko množico visokih ljudi.

Sprememba pripadnosti je v tem primeru linearno povezana s spremembo v višini, kar je veliko bližje naravnemu načinu razmišljanja.

Naš primer sedaj razširimo tako, da uvedemo tri opisne razrede za višino osebe – nizek, srednje visok in visok. Vsakemu razredu pripada ustrezna mehka množica M_{nizek}, $M_{srednje\_visok}$ in M_{visok}. Grafi pripadnostnih funkcij za vse tri mehke množice so na sliki 28.

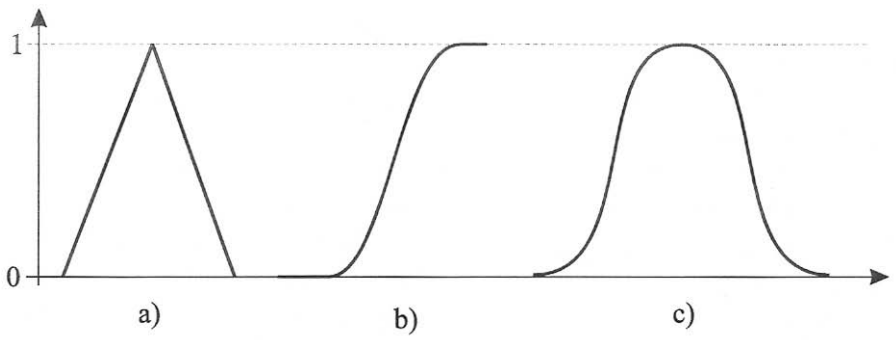


Slika 28: graf pripadnostnih funkcij za mehke množice višinskih razredov.

Definicijsko območje za višino je na slikah 26–28 omejeno na interval 100 cm – 220 cm in je odvisno od vrste problema. Slika 28 prikazuje, kako odčitamo pripadnost posameznim razredom za določeno osebo z znano višino v kot višino presečišč ustreznih grafov z navpičnico pri v. Za osebo višine 172 cm ugotovimo, da je 0,2 nizka, 0,8 srednje visoka in 0 visoka.

Značilnost prikazanih grafov pripadnostnih funkcij za množice, ki označujejo pomensko sosednje razrede določene lastnosti, je, da se sekajo na višini 0,5. V točkah, kjer prične pripadnost eni množici upadati, prične pripadnost sosednji množici naraščati in obratno. Na tak način se vsota pripadnosti vseh elementov ohranja in je enaka 1, kar je v praksi pogosto zaželeno.

Grafi pripadnostnih funkcij na sliki 28 so trapezoidne oziroma odsekoma linearne oblike, uporabne pa so tudi nekatere druge oblike. Slika 29 prikazuje trikotno, logistično in Gaussovo funkcijo, ki se poleg trapezoidne v praksi najpogosteje uporabljajo.



Slika 29: grafi pripadnostnih funkcij: a) trikotna, b) logistična in c) Gaussova.

4.2.2 Lastnosti mehkih množic

Teorijo mehkih množic obravnavamo kot posplošitev klasične teorije množic [19].

- Podpora (ang. *support*) *S(A)* mehke množice *A* je podmnožica univerzalne množice *U*, katere vsak element ima stopnjo pripadnosti različno od *0*. Npr. podpora množici 'srednja temperatura' je interval [10, 30] v stopinjah Celzija.
- Kardinalnost (ang. *cardinalty*) klasične množice je število elementov v množici, pri mehki pa je definirana kot :

$$M(A) = \sum_{u \in U} \mu_A(u) \tag{4.10}$$

- Potenčna množica (ang. *power set*) mehke množice *A* je množica vseh mehkih podmnožic množice *A*.
- Normalna mehka množica *A* ima vsaj eno stopnjo pripadnosti z vrednostjo *1*.
- Mehki posameznik (ang. *fuzzy singleton*) je mehka množica, katere podpora vsebuje eno samo točko $\mu \in U$, pri kateri je $\mu_A(u) = 1$.

4.2.3 Osnovne operacije nad mehkimi množicami

Klasične množice so poseben primer mehkih množic, pri katerih sta le dve stopnji pripadnosti ali vrednosti pripadnostne funkcije, *0* in *1*. Zato morajo vse definicije mehkih množic, veljati tudi za klasične množice (ang. *Crisp set*). V nadaljevanju so podane osnovne mehke operacije nad dvema mehkima množicama *A* in *B*, definirane nad univerzalno množico *U* [19].

- Unija mehkih množic *A* in *B* s skupno domeno je mehka množica $A \cup B$ s pripadnostno funkcijo $\mu_{A \cup B}(x) = s\{\mu_A(x), \mu_B(x)\}$, kjer je funkcija *s(a,b)* s-norma. V praksi najpogosteje uporabljeni obliki s-norme sta:

$$\mu_{A \cup B}(x) = max\{\mu_A(x), \mu_B(x)\}, \tag{4.11}$$

$$\mu_{A\cup B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x)$$

- Presek mehkih množic A in B s skupno domeno je mehka množica $A \cap B$ s pripadnostno funkcijo $\mu_{A\cap B}(x) = t\{\mu_A(x), \mu_B(x)\}$, kjer je funkcija *t(a,b)* t-norma. V praksi najpogosteje uporabljeni obliki t-norme sta:

$$\mu_{A\cap B}(x) = min\{\mu_A(x), \mu_B(x)\}, \quad (4.12)$$
$$\mu_{A\cap B}(x) = \mu_A(x)\mu_B(x)$$

- Komplement $\rightharpoonup A$:

$$\mu_{\rightharpoonup A}(x) = 1 - \mu_A(x), za\ vse\ x \in U \quad (4.13)$$

- Algebraični produkt $A \cdot B$:

$$\mu_{AB}(x) = \mu_A(x) \cdot \mu_B(x), za\ vse\ x \in U \quad (4.14)$$

- Algebraična vsota $A + B$:

$$\mu_{A+B}(x) = \mu_A(x) + \mu_B(x), za\ vse\ x \in U \quad (4.15)$$

- Enakost *A= B:*

$$\mu_A(x) = \mu_B(x), za\ vse\ x \in U \quad (4.16)$$

- Podmnožica $A \subset B$:

$$\mu_A(x) \leq \mu_B(x), za\ vse\ x \in U \quad (4.17)$$

Za operacije nad mehkimi množicami veljajo naslednje lastnosti: asociativnost, komutativnost in distributivnost. Pomembna lastnost mehkih množic (v primerjavi s klasičnimi) je v neveljavnosti pravila izločene sredine (ang. *excluded middle*) in pravila protislovja (ang. *contradiction*):

$$A \cup \rightharpoonup A \neq U \quad (4.18)$$
$$A \cup \rightharpoonup A \neq \{\}$$

Unija mehke množice in njenega komplementa torej ne dajeta nujno celotne univerzalne množice *U* in njun presek ni nujno prazna množica.

4.2.4 Mehka pravila in sklepanje

Mehka logika je odlično orodje za implementacijo znanja "zdravega razuma" (ang. *commomn sense knowledge*), nejasnega znanja v računalniških programih in za posledično izvajanje določenih zaključkov. Mehka logika izhaja iz mehkih relacij in mehkih trditev (ang. *fuzzy propositions*), ki so definirane na osnovi mehkih množic [19].

4.2.4.1 Mehka pravila

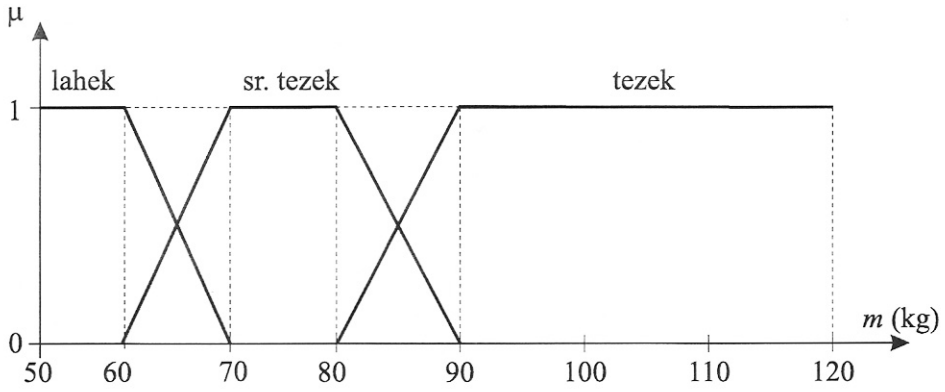
Definirana lastnost višina je primer lingvističnih, jezikovnih ali mehkih spremenljivk (ang. *linguistic variable, fuzzy variable*). Vrednosti lingvistične spremenljivke so izrazi v naravnem jeziku (npr. nizek, srednje visok ali visok). Naboru možnih vrednosti lingvistične spremenljivke pravimo lingvistične ali mehke vrednosti (ang. *linguistic value, fuzzy value*). Vsaka mehka vrednost ustreza eni mehki množici. Lingvistične spremenljivke in vrednosti predstavljajo enega izmed elementov mehke logike, ki je zato posebej primerna za sklepanje z izrazi v naravnem jeziku [2].

Obravnavajmo osebo z višino $v = 190\,cm$ in maso m = 67 kg. Iz slik 28 in 30 lahko razberemo, da je pripadnost te osebe različnim mehkim množicam naslednja:

$$\mu_{nizek}(A) = 0{,}6$$
$$\mu_{srednje\ \_visok}(A) = 0{,}4$$
$$\mu_{visok}(A) = 0$$
$$\mu_{lahek}(A) = 0{,}3$$
$$\mu_{srednje\ \ tezak}(A) = 0{,}7$$
$$\mu_{tezak}(A) = 0$$

Po prej opisanih operacijah izpeljimo naslednje pripadnosti:

$$\mu_{\overline{nizek}}(A) = 1 - \mu_{nizek}(A) = 0{,}4$$
$$\mu_{srednje\ \_visok\ \cap lahek}(A) = min\{\mu_{srednje\ \_visok}(A), \mu_{lahek}(A)\} = 0{,}3$$
$$\mu_{visok\ \cup \overline{tezek}}(A) = max\{\mu_{visok}(A), \mu_{\overline{tezek}}(A)\} = 1$$



Slika 30: graf pripadnostnih funkcij za mehko množico težnostnih razredov.

Da neka oseba A pripada množici visokih ljudi M_{visok} s pripadnostjo 0,6, zapišimo v obliki naslednjega logičnega izraza:

$$visina(A) = visok_{0,6}$$

Vrednost 0,6 v tem primeru obravnavamo kot stopnjo veljavnosti ali izpolnjenosti logičnega izraza.

Mehka logika pozna klasične logične operatorje, s katerimi kombiniramo vrednosti lingvističnih spremenljivk in tvorimo bolj kompleksne izraze. Negacija logičnega izraza je enakovredna komplementu mehke množice, logična konjukcija preseku mehkih množic in logična disjunkcija uniji mehkih množic.

Negacija logičnega izraza

$$(teza = lahek_{0,3})$$

je izraz

$$\neg\,(teza = lahek_{0,3})$$

ki ga lahko po ovrednotenju komplementa zapišemo kot

$$teza = \neg\, lahek_{0,7}$$

Veljavnost konjunkcije

$$visina = visok_{0,6} \wedge \neg\,(teza = lahek_{0,3})$$

je enaka

$$\min\{0,6; 1 - 0,3\} = \min\{0,6; 0,7\} = 0,6$$

Veljavnost disjunkcije

$$\neg\,(visina = visok_{0,6}) \vee teza = lahek_{0,3}$$

je enaka

$$\min\{1 - 0,6; 0,3\} = \min\{0,4; 0,3\} = 0,4.$$

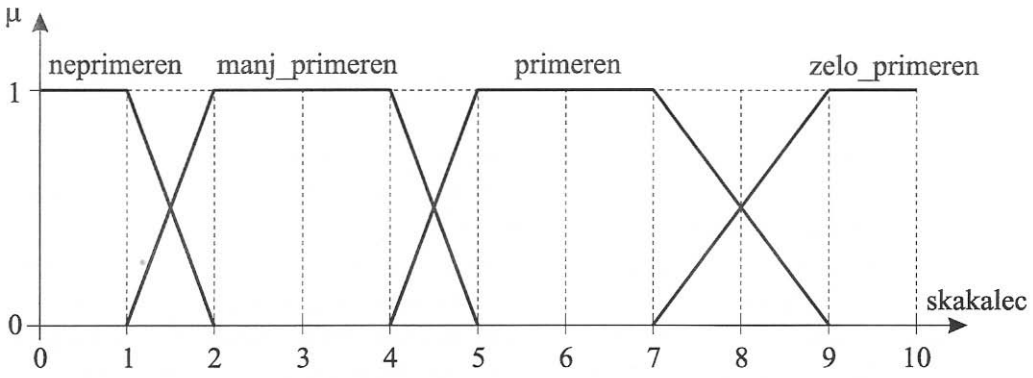
S predstavljeno sintakso zapišemo pravila, ki omogočajo sklepanje v mehki logiki. Mehko pravilo (ang. *fuzzy rule*) je vsako pravilo naslednje oblike (Zadeh – Mamdami pravilo):

$$IF\ X_1 = x_1 \wedge X_2 = x_2 \wedge \ldots \wedge X_n = x_n THEN\ Y = y,$$

Kjer so X_i in Y lingvistične spremenljivke, x_i in y pa mehke vrednosti. Izrazi v pogoju so povezani s konjunkcijo, disjunkcijo pa dosežemo z definicijo več različnih pravil, ki imajo isti sklep. Primer konkretnega pravila je naslednje pravilo:

$$IF\ visina = visok \wedge teza = tezek\ THEN\ skakalec = neprimeren,$$

kjer lingvistična spremenljivka 'skakalec' označuje primernost osebe, o kateri sklepamo za skakalca v višino. Celotno definicijo spremenljivke skakalec s štirimi mehkimi vrednostmi prikazuje slika 31, kjer smo kot razpon numeričnih ocen primernosti za skakalca v višino izbrali interval [0,10] [2].



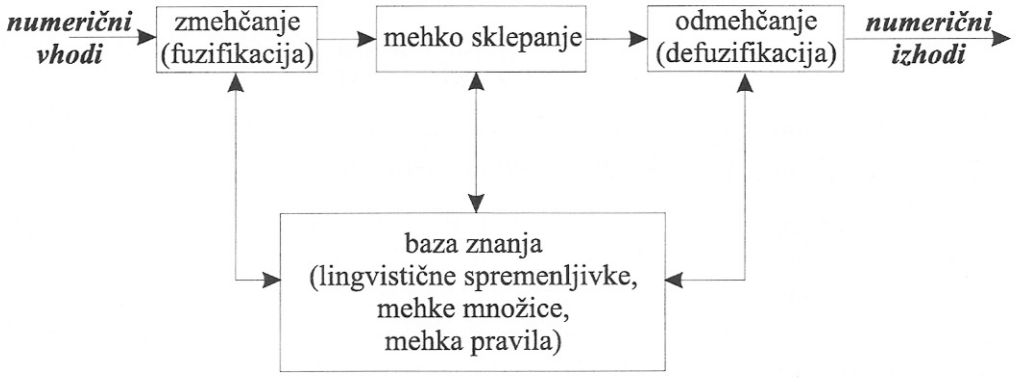
Slika 31: graf pripadnostnih funkcij za primernost skakalca v višino.

4.2.4.2 Mehko sklepanje

Veljavnost pogojev v mehkih pravilih se prenaša na sklepe, kar omogoča izvajanje sklepanja v mehki logiki. Tipični sistem sklepanja je prikazan na sliki 32 in vsebuje tri osnovne korake [Engelbrecht, 2002]:

- zmehčanje ali fuzifikacijo (ang. *fuzzification*), pri kateri iz numeričnih vhodnih podatkov (x_1,...,x_n) izračunamo pripadnosti posameznim mehkim množicam za vse lingvistične spremenljivke,
- mehko sklepanje (ang. *fuzy interface*), pri katerem izračunamo aktivacije (ang. *activation*) posameznih mehkih pravil in s tem veljavnosti pripadajočih sklepov,

- odmehčanje ali fuzifikacija (ang. *defuzzification*), pr kateri iz kombinacije mehkih sklepov sestavimo končni sklep in ga pretvorimo v izhodni numerični podatek, ki ga označimo z y.



Slika 32: tipični sistem mehkega sklepanja.

V vseh treh fazah sklepanja potrebujemo dostop do podatkovne baze oziroma baze znanja, ki jo sestavljajo definicije lingvističnih spremenljivk s pripadnostnimi funkcijami za vse mehke vrednosti ter definicije mehkih pravil.

Zmehčanje poteka tako, da s pomočjo pripadnostnih funkcij za posamezne mehke vrednosti iz numeričnih vhodnih podatkov določimo iskane pripadnosti vsem mehkim množicam. Primer takšnega izračuna smo naredili v prejšnjem razdelku za višino in težo osebe.

Mehko sklepanje poteka tako, da izvedemo logično operacijo nad pripadnostnimi mehkimi vrednostimi v pogojih in tako dobljeno aktivacijo α prenesemo na sklepe oziroma izhodne mehke vrednosti. Kadar neka mehka vrednost nastopa v več sklepih, upoštevamo maksimalno aktivacijo, ki jo je pri tem dosegla. Kot zgled vzemimo naslednjo množico pravil:

1. $IF\ visina = visok \wedge teza = tezek\ THEN\ skakalec = neprimeren$
2. $IF\ visina = nizek\ THEN\ skakalec = neprimeren$
3. $IF\ visina = sred\_visok \wedge \rightarrow teza = lahek\ THEN\ skakalec = manj\_primeren$
4. $IF\ visina = sred\_visok \wedge teza = lahek\ THEN\ skakalec = primeren$
5. $IF\ visina = visok \wedge teza = sred\_tezek\ THEN\ skakalec = primeren$
6. $IF\ visina = visok \wedge teza = lahek\ THEN\ skakalec = zelo\_primeren$

Aktivacije zgornjih dveh pravil za osebo z z višino v = 160 cm in maso m = 67 kg so potem naslednje:

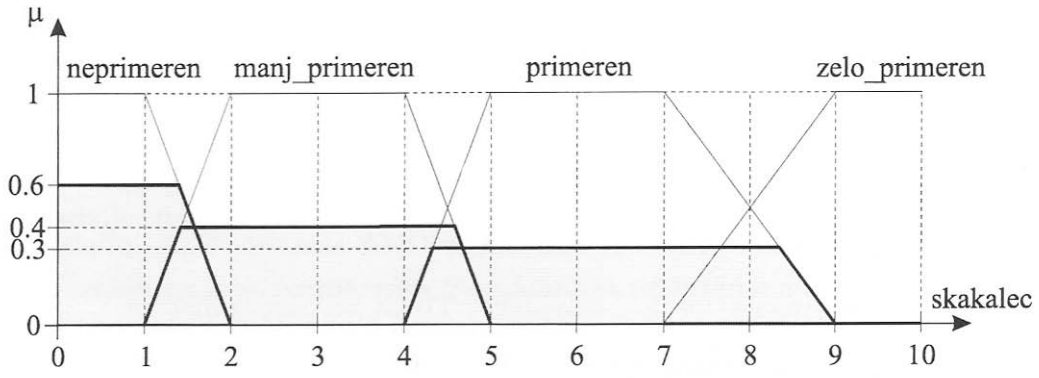
$$\alpha_1 = \min\{0; 0\} = 0$$
$$\alpha_2 = 0{,}6$$
$$\alpha_3 = \min\{0{,}4; 1 - 0{,}3\} = 0{,}4$$
$$\alpha_4 = \min\{0{,}4; 0{,}3\} = 0{,}3$$
$$\alpha_5 = \min\{0;\ 0{,}7\} = 0$$
$$\alpha_6 = \min\{0; 0{,}3\} = 0$$

Končne aktivacije za posamezne sklepe so naslednje:

- za sklep skakalec = neprimeren je aktivacija enaka $max\{\alpha_1, \alpha_2\}$=0,6,
- za sklep skakalec = manj_primeren je aktivacija enaka 0,4,
- za sklep skakalec = primeren je aktivacija enaka $max\{\alpha_4, \alpha_5\}$=0,3,
- za sklep skakalec = zelo_primeren je aktivacija enaka 0.

Vsaka izhodna mehka vrednost v procesu sklepanja doseže določeno aktivacijo. Naloga odmehčanja je iz kombinacije dobljenih sklepov tvoriti enoumen numerični odgovor, naprimer oceno primernosti skakalca v višino.

Prvi korak odmehčanja je kombinacija dobljenih sklepov. To izvedemo tako, da grafe pripadnostnih funkcij za izhodne mehke vrednosti obrežemo pri doseženih aktivacijah in načrtamo unijo območij, ki jih zajemajo pripadnostne funkcije. Za primer skakalca v višino je postopek prikazan na sliki 33 [2].



Slika 33: obrezovanje grafov pripadnostnih funkcij pri doseženi aktivaciji in tvorba skupnega območja.

V literaturi najdemo več načinov tvorbe izhodnega numeričnega podatka postopek imenujemo tudi ostrenje, med njimi v praktičnih aplikacijah prevladujejo, [2,19]:

- **metoda maksimalne pripadnosti** (ang. *makismum membership method*): izhodni numerični podatek y je položaj maksimuma pripadnosti na skupnem območju, če je maksimum na platoju, vrnemo središčno točko platoja. Za primer skakalca, si lahko izhodno funkcijo ogledamo na sliki 34.
- **sredina maksimumov**: tukaj iščemo *y*, ki ustreza maskimumu pripadajoče pripadnostne funkcije na skupnem območju. Če je maksimumov več, poiščemo njihovo povprečje:

$$y = \frac{m + M}{2} \tag{4.19}$$

  Kjer sta *m* najmanjši in *M* največji *y*, pri katerih ima pripadnostna funkcija lokalni maksimum.

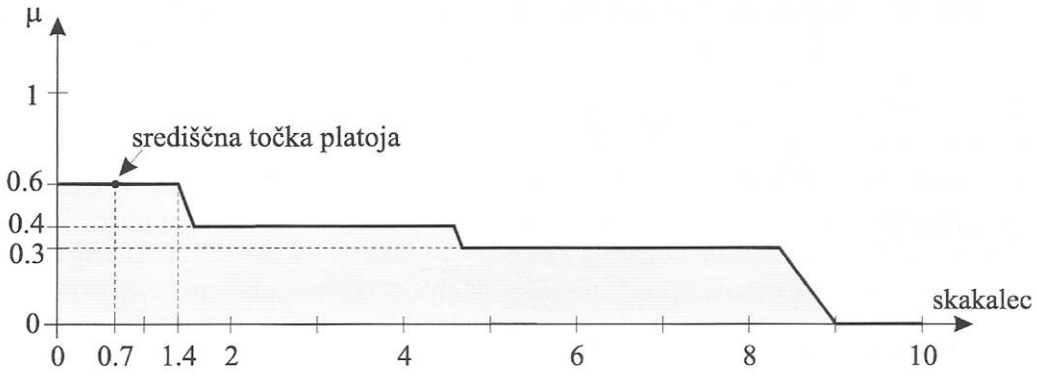
- **metoda težišča** ali **center gravitacije** (ang. *centroid method*):

$$y = \frac{\int_{y \in Y} \mu(y) y dy}{\int_{y \in Y} \mu(y) dy} \tag{4.20}$$

  Običajno vzamemo kar vsoto kot približek. Metoda je uporabna za simetrične pripadnostne funkcije:

$$y = \frac{\sum_i \mu(y_i) y_i}{\sum_i \mu(y_i)} \tag{4.21}$$

  kjer so y_i težišča grafov posameznih pripadnostnih funkcij.



Slika 34: metoda maksimalne pripadnosti.

5. Simulacija inteligentnega agenta

V tem poglavju prikazujemo simulacijo strategije čiščenja okolja z inteligentnim agentom v obliki avtonomnega mobilnega robota. Z uporabo mehke logike dosežemo, da agent izvaja racionalno in avtonomno odločanje. Računalniško simulacijo smo izdelali z namenom, da lažje analiziramo delovanje agenta, optimalno nastavimo parametre uporabljenega algoritma A* in mehke logike ter odpravimo morebitne napake. Za razvoj aplikacije smo uporabili prosto dostopno razvojno okolje Microsoft Visual Studio Express 2008 z ogrodjem XNA Game Studio. Programska koda je napisana v jeziku C#. Simulacijske teke smo opravili na računalniku s procesorjem Dual Core E5400 - 2,7 GHz, 2GB notranjim pomnilnikom in operacijskim sistemom Windows 7.

5.1 Microsoft XNA Game Studio

Navidezno okolje predstavlja vsebino določenega medija, ki obstaja le v mislih ustvarjalca ali pa je deljena z ostalimi ljudmi. Računalniško zasnovano navidezno okolje tako predstavlja opise objektov znotraj simulacije. Navidezno okolje je:

- imaginarni prostor, ki je običajno prikazan preko medija ali,
- opis množice objektov v prostoru, ter pravil in razmerij, ki vodijo te objekte.

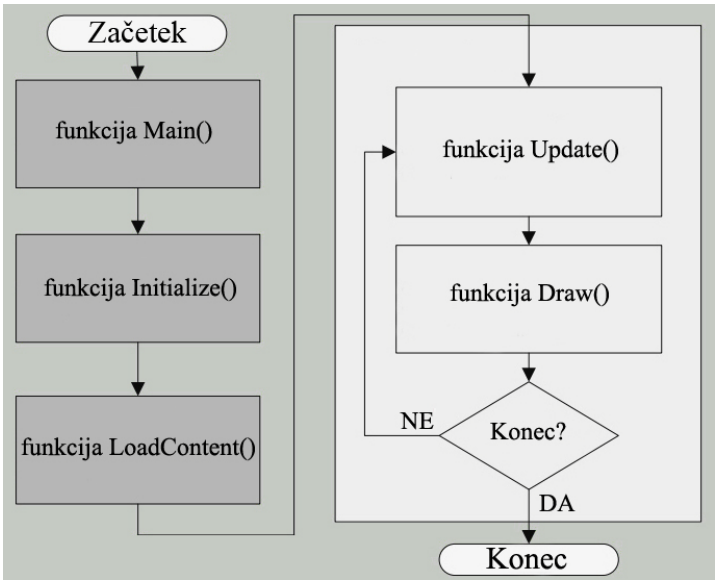
S pojmom avatar (hin. *utelešenje božanstva*) označujemo predstavitev uporabnika ali fizičnega objekta v navideznem okolju. Če fizični objekt predstvlja inteligentnega agenta, ima tak objekt sposobnosti reševanja problemov, učenja in obnašanja v navideznem svetu.

Za upodobitev tridimenzionalnega navideznega okolja je osnovna programska komponenta grafični pogon. Jedro pogona vsebuje podatkovne in datotečne strukture, operacije nad njimi, matematične strukture in operacije ter prikazovalni sistem (ang. *rendering system*). Za realističen prikaz okolja in objektov v njem skrbijo algoritmi senčenja in osvetlitve, prikaz tekstur ter ostali efekti realizma. Kot del pogona štejemo tudi programske knjižnice, ki skrbijo za večino nizko nivojskih operacij in komunikacijo z grafičnimi procesnimi enotami. Primera takih knjižic sta OpenGL in Direct3D.

Za razvoj simulacij, kjer je potrebna sprotna vizualizacija tridimenzionalnega okolja, je Microsoft predstavil programsko ogrodje XNA Game Studio [20], ki je integrirano v razvojno okolje Microsoft Visual Studio. Z njim večino težav, povezanih z gradnjo grafičnih okolij

rešujemo na standardiziran način. XNA temelji na ogrodju .NET 2.0 in knjižnicah DirectX (DirectX je zbirka multimedjiskih knjižnic vključno z Direct3D) ter podpira pisanje programske kode v jeziku C#.

Programsko rešitev v večini primerov sestavlja več datotek z izvorno kodo. Zbirko datotek in nastavitev, ki rešujejo nek problem, imenujemo projekt. Microsoft Visual Studio ima vnaprej pripravljenih več različnih tipov projektov. Pri ustvarjanju XNA projekta se ustvarita dva razreda in sicer »Program« in »Game«. Razred »Program« vsebuje glavno funkcijo »Main« medtem ko razred »Game« vsebuje štiri pomembne funkcije za izvajanje simulacije. Ogrodje XNA poskrbi za klic vsake od posameznih funkcij v določenem zaporedju v "neskončni" simulacijski zanki (slika 35). Neskončna zanka vključuje funkciji »Draw« in »Update«. Funkciji »Initialize« in »LoadContent« se izvršita le enkrat v celotnem simulacijskem teku na začetku simulacije, kjer nastavita potrebne začetne parametre za izvajanje simulacije.



Slika 35: diagram poteka simulacijskega teka.

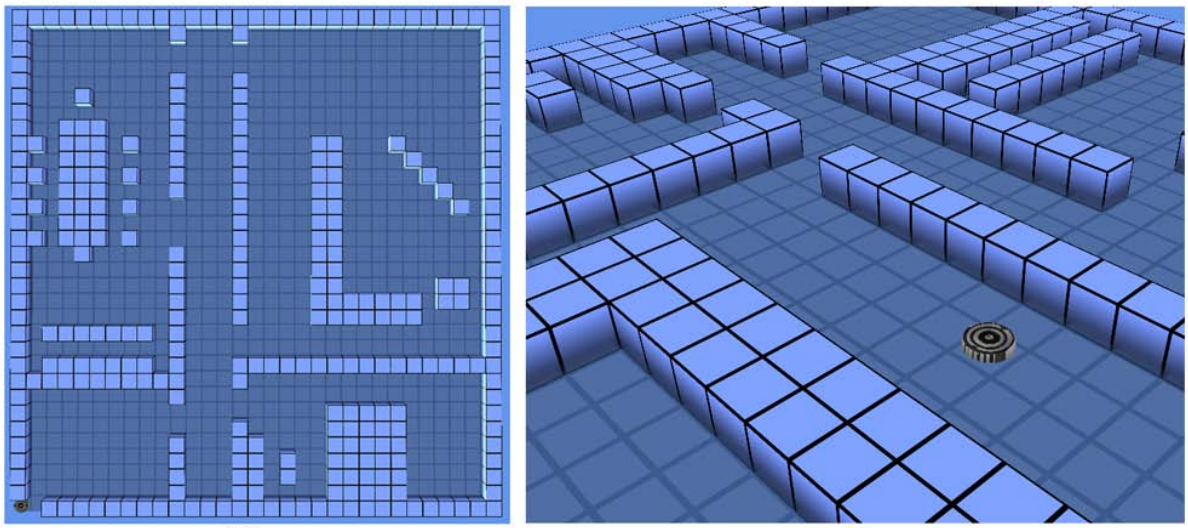
Opis posameznih funkcij v razredu »Game«:

- **»Initialize«:** Naloga funkcije je, da nastavi začetne vrednosti spremenljivk in nastavitve simulacije. Klic funkcije se opravi enkrat in preden se simulacijska zanka začne izvajati.
- **»LoadContent«:** Naloga funkcije je, da v simulacijo naloži multimedjiske datoteke, kot so 3D modeli, avdio datoteke, teksture, pisave, itd. Funkcija je klicana takoj po funkciji »Initialize« pred začetkom simulacijske zanke.

- **»Update«:** Naloga funkcije je, da obdeluje podatke simulacije. Funkcija skrbi za vnos podatkov s strani uporabnika in za posodabljanje stanja navideznega okolja ter objektov v njem. Klic se izvršuje v intervalih. V privzetih nastavitvah je čas klica vsake 1/60 sekunde oziroma v časovnih intervalih 0,167 ms.
- **»Draw«:** Funkcija skrbi za izrisovanje grafike na zaslon in je klicana po vsakem uspešnem klicu funkcije »Update«.

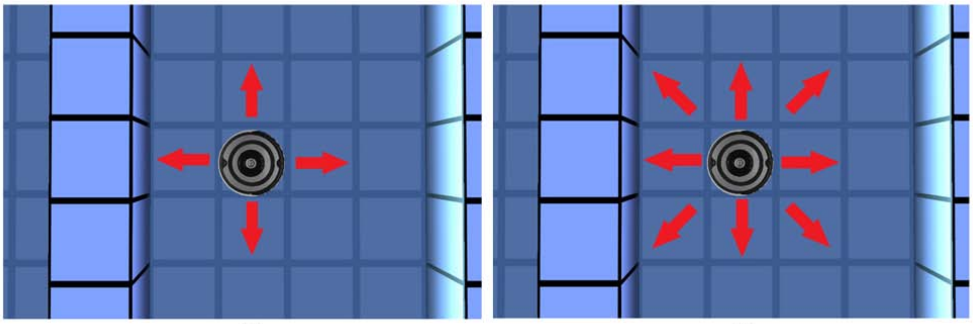
5.2 Analiza algoritma A*

Okolje smo razdelili na enako velike, dotikajoče se mnogokotnike. Ob delitvi so označeni mnogokotniki, ki predstavljajo oviro (slika 36a). Končni izdelek diskretizacije okolja je dvodimenzionalen zemljevid zasedenosti (ang. *occupancy map*). V navideznem okolju smo za oviro uporabili kocko (slika 36b).



Slika 36: a) zemljevid zasedenosti in b) prikaz ovir v navideznem okolju.

Vsakemu mnogokotniku, ki ne predstavlja ovire dodelimo vozlišče. Vozlišča imajo različno število povezav s sosednjimi vozlišči. Več povezav ima vozlišče, bolj natančno pot bo algoritem za iskanje poti izračunal. Pri delitvi okolja z mnokogotniško mrežo je najbolj pogosto število povezav vozlišč s štirimi (slika 37a) oziroma osmimi povezavami (slika 37b). Za izračun natančnejše poti v algoritmu uporabljamo osem povezav med sosednjimi vozlišči.



Slika 37: a) vozlišče s štirimi povezavami in b) z osmimi.

Za iskanje in načrtovanje poti uporabljamo algoritem A*. Hevristiko *h(n)* ocenimo na različne načine z Evklidovo, Manhattan in Čebiševo razdaljo. Učinkovitost določene hevristike je odvisna od števila povezav med sosednjimi vozlišči. Za ceno poti pri premiku v horizontalni ali vertikalni smeri uporabimo D_1=10 in po diagonali $D_2 \doteq 14{,}14$. V simulaciji uporabljamo štiri najpogostejše hevristike:

- **hevristika Manhattan**: Enačba za *h(n)* med točkama $X = \{x_1, y_1\}$ in $Y = \{x_2, y_2\}$ je:

$$h(n) = D_1(|x_1 - x_2| + |y_1 - y_2|) \quad (5.1)$$

- **Evklidova hevristika:** Enačba za *h(n)* med točkama $X = \{x_1, y_1\}$ in $Y = \{x_2, y_2\}$ je:

$$h(n) = D_1\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (5.2)$$

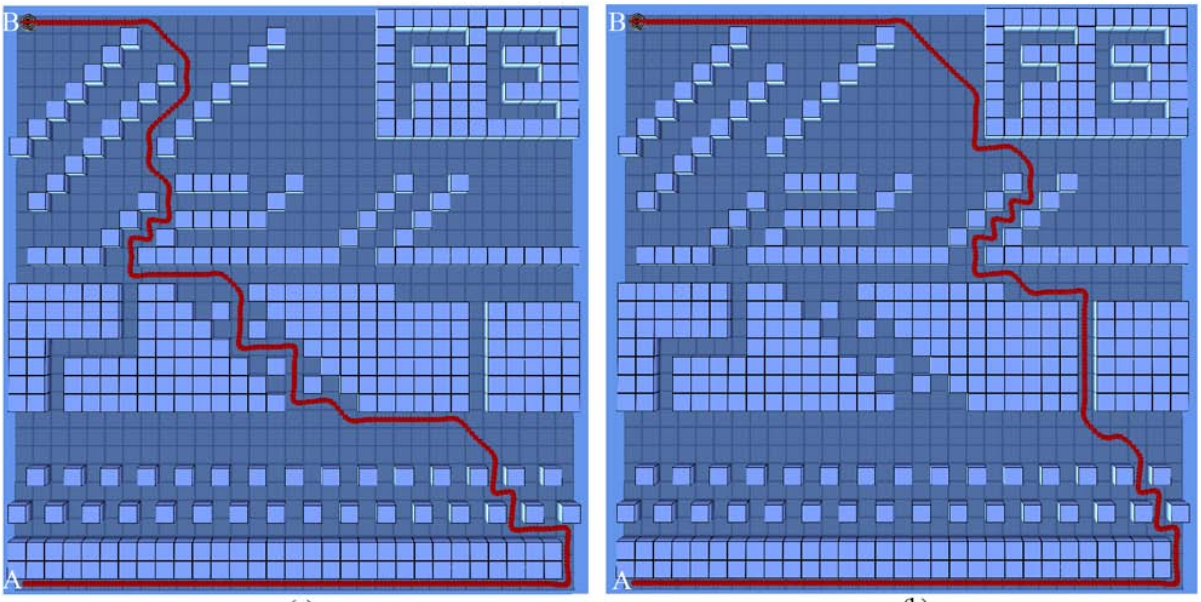
- **Čebiševa hevrstika**: Enačba za *h(n)* med točkama $X = \{x_1, y_1\}$ in $Y = \{x_2, y_2\}$:

$$h(n) = D_1(\max(|x_1 - x_2|, |y_1 - y_2|)) \quad (5.3)$$

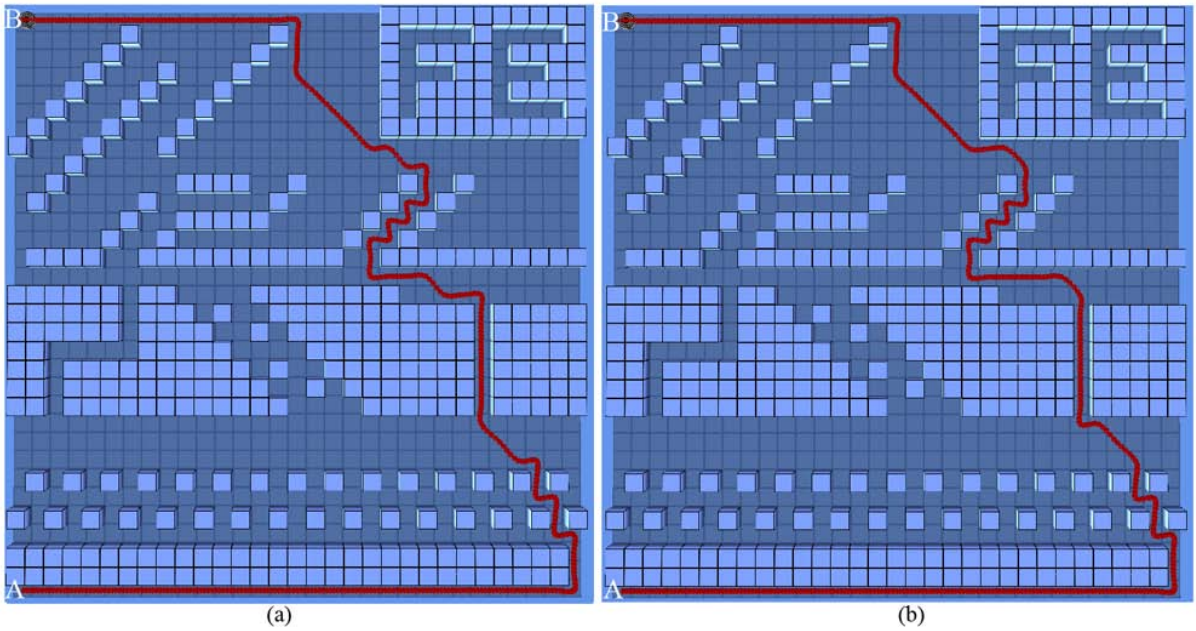
- **Diagonalna hevristika**: je kombinacija razdalje Manhattan in Čebiševe. Enačba za *h(n)* med točkama $X = \{x_1, y_1\}$ in $Y = \{x_2, y_2\}$.

$$h_1(n) = min(|x_1 - x_2|, |y_1 - y_2|) \quad (5.4)$$
$$h_2 = |x_1 - x_2| + |y_1 - y_2| \quad (5.5)$$
$$h(n) = D_2 * h_1(n) + D_1 * (h_2(n) - 2 * h_1(n)) \quad (5.6)$$

Iskanje poti izvedemo na dvodimenzionalnem zemljevidu dimenzij 31 × 32 oziroma 992 celic s približno 40 % zasedenostjo z ovirami. Začetna celica je v spodnjem levem kotu v točki A in končna v zgornjem levem kotu v točki B. Na slikah 38 in 39 so prikazane izračunane poti algoritma A* pri različnih hevristikah. V tabeli 1 podajamo čas izračuna, ceno poti ter število vseh vozlišč, ki jih je algoritem med izračunavanjem obiskal pri posamezni hevristiki.



Slika 38: a) hevristika Manhattan in b) Evklidova hevristika.



Slika 39: a) Čebiševa in b) diagonalna hevristika.

Hevristika	Čas izračunavanja [s]	Cena iskane poti	Št. obiskanih vozlišč
Manhattan	0,00175	937,3	371
Evklidova	0,00279	923,1	482
Diagonalna	0,00265	923,1	579
Čebiševa	0,00268	923,1	580

Tabela 1: primerjava časa izračunavanj, cene iskanih poti ter število obiskanih vozlišč.

Za primerjavo smo izvedli iskanje poti na zemljevidu dimenzij 93 × 32 oziroma 2976 celic s približno 39 % zasedenostjo ovir ter podali rezultate v tabeli 2.

Hevristika	Čas izračunavnja [s]	Cena iskane poti	Št. obiskanih vozlišč
Manhattan	0,00667	1683,5	1182
Evklid	0,01012	1683,5	1566
Diagonalna	0,00941	1689,4	1812
Čebiš	0,00968	1683,5	1815

Tabela 2: primerjava časa izračunavanj, cene iskanih poti ter število obiskanih vozlišč za zemljevid dimenzij 93×32.

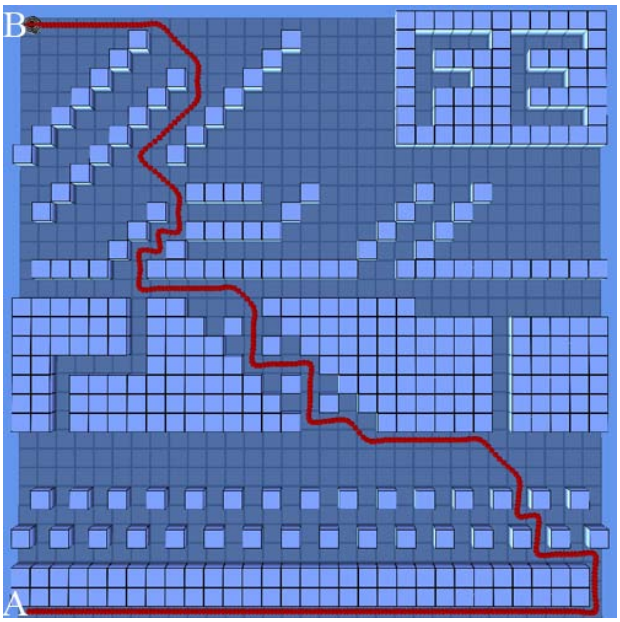
Rezultati kažejo, da je hevristika Manhattan najhitrejša v času izračunavanja, vendar ima v nekaterih primerih izračunana pot višjo ceno v primerjavi z ostalimi izbranimi hevristikami. Razlika cene poti je v večini primerov zanemarljiva. Pri hevristiki Manhattan se v nekaterih primerih pojavi višja cena poti, kar je posledica uporabe osmih povezav med vozlišči. Ta hevristika podaja najboljše rezultate pri uporabi štirih povezav (vertikalne in horizontalne). Ostale tri hevristike imajo približno enake rezultate. Evklidova hevristika se izkaže za najpočasnejšo v času izračunavanja. V nekaterih primerih (predvsem v računalniških igrah) se za hevristiko uporablja kvadrirana Evklidova razdalja, da se izognemo časovno potratni matematični operaciji korenjenja.

$$h(n) = D_1((x_1 - x_2)^2 + (y_1 - y_2)^2) \qquad (5.7)$$

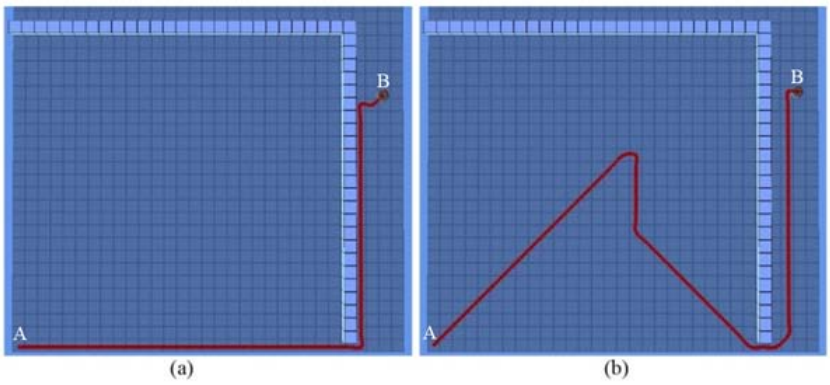
Slednja preceni hevristiko pri izračunavanju funkcije *f(n) = g(n) + h(n),* zato ni priporočljiva. Pri velikih razdaljah funkcija *g(n)* ne bo imela nobenega vpliva na *f(n)* zaradi prevelike ocene funkcije *h(n)* [17]. Problem, ki nastane v tem primeru, je pot z višjo ceno zaradi majhnega števila obsikanih vozlišč. Na obeh zemljevidih zasedenosti smo opravili meritve s kvadrirano Evklidovo hevristiko in prikazali rezultate v tabeli 3 in na sliki 40 za zemljevid iz prvega primera. Hevristika se je izkazala za hitro, vendar je njena učinkovitost v celoti odvisna od ovir na zemljevidu. V vseh primerih so cene iskanih poti višje od ostalih hevristik. V primeru ovir kot, so prikazane na sliki 41, algoritem izračuna znatno višjo ceno poti. Prikazana je primerjava s hevristiko Manhattan. Časovna zahtevnost algoritma A* je odvisna od hevristike, kar je razvidno iz dobljenih rezultatov.

Primer	Čas izračunavnja [s]	Cena iskane poti	Št. obiskanih vozlišč
Primer 1 (31×32)	0,00056	945,6	104
Primer 2 (73×32)	0,00120	1877,3	206

Tabela 3: primerjava časa izračunavanj, cene iskanih poti ter število obiskanih vozlišč.



Slika 40: iskanje poti s kvadrirano Evklidovo hevristiko.



Slika 41: a) hevristika Manhattan in b) Evklidova kvadrirana hevristika.

5.3 Odločanje inteligentnega agenta z mehko logiko

Tipično področje uporabe mehke logike so zelo zapleteni procesi, predvsem v primerih, ko model procesa ni natančno znan. Nadaljna prednost pri uporabi avtomatiziranih procesov se kaže v primerih izkustvenega znanja pri ročnem vodenju procesa, ki ga je pridobilo osebje po dolgoletnem delu z njim. Če obstaja zadostna količina izkustevenga znanja, se izkaže mehka logika za primerno obliko načina vodenje sistema.

5.3.1. Definicija problema in potek modeliranja sistema mehkega odločanja

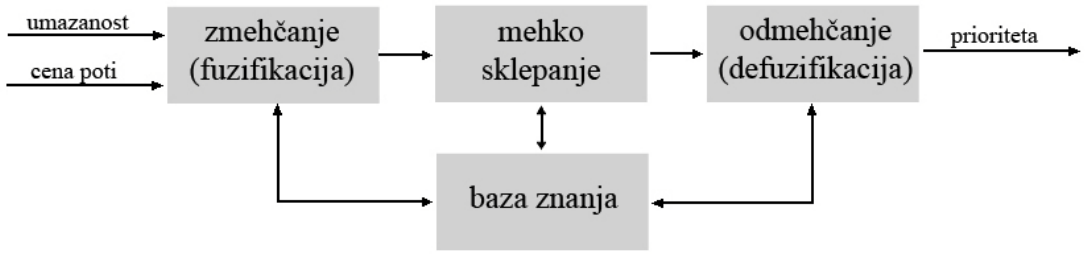
V našem primeru je naloga inteligentnega agenta čiščenje tal. Inteligentno okolje generira seznam smeti, o katerih hranimo naslednje podatke:

- globalno lokacijo v diskretiziranem okolju,
- umazanost v odstotkih,
- ceno poti, ki jo agent porabi za premik do smeti iz trenutne lokacije,
- prioriteto, na podlagi katere se inteligentni agent odloči za akcijo.

Podatek, ki podaja umazanost, lahko v splošnem podaja tudi drug kriterij na osnovi česar se agent odloči, npr.:

- če imamo v okolju kosovne smeti, je lahko to podatek o velikosti,
- v industrijskem okolju lahko to informacijo podaja podatek o stopnji nevarnosti objekta, ki ga je potrebno umakniti (razlita tekočina ipd.)
- analogno problem predstavimo kot medicinskega inteligentnega agenta, ki v bolnišnici (ali v domu za ostarele) skrbi za distribucijo zdravil, pri čemer za kriterij uporabimo čas ob katerem pacient zaužije zdravilo [21] itd.

Skupaj s podatkom cene poti se agent s postopki mehke logike (slika 42) odloči za smet, ki je najbolj primerna, da jo najprej odstrani. Smet, ki je najbolj primerna ima najvišjo prioriteto na seznamu vseh smeti v okolju.



Slika 42: postopek fuzifikacije in defuzifikacije posamezne smeti na seznamu vseh smeti.

V naslednjem odstavku je prikazana psevdo koda algoritma, ki definira delovanje inteligentnega agenta:

Postopek

dokler »seznam smeti ni prazen« **naredi**

»Pridobi trenutno lokacijo (START) inteligentnega agenta.«

dokler »ne prideš do konca seznama smeti« **povečaj števec**

»Za vsako *smet[i]* izračunaj z algoritmom A* ceno poti iz START v *smet[i].lokacija*. Opravi fuzifikacijo numeričnih vhodov umazanosti in cene poti ter opravi mehko sklepanje. Opravi defuzifikacijo – tvori numerični izhod - prioriteto«

konec povečaj števec

»Sortiraj seznam smeti glede na prioriteto. Nastavi za točko END lokacijo najbolj primerne smeti iz seznama in z algoritmom A* poišči pot za premik agenta iz točke START v END. Opravi premik agenta do točke END. Ko agent počisti smet na lokaciji END jo izbriši iz seznama smeti.«

konec naredi.

Po uspešno končanem postopku poišči pot za premik agenta iz END k bazni (polnilni) postaji.

Konec postopka

Pri modeliranju sledimo naslednjim postopkom:

- **Določanje baze pravil.** Pri tem izhajamo iz osnovne ideje posnemanje človeka oziroma operatorja, ki vodi proces.
- Postopek fuzifikacije ali mehčanja zahteva, da za numerične vhode, ki jih ponavadi dobimo iz senzorjev določimo **vhodne lingivstične spremenljivke** s pripadajočimi opisnimi razredi.
- Postopek defuzifikacije ali odmehčanja zahteva, da določimo **izhodne lingivstične spremenllјivke** s pripadajočimi opisnimi razredi iz katerih dobime ostre oziroma numerične vrednosti.

Če izvedeni postopek snovanja mehanizma mehkega odločanja ne daje zadovoljivih rezultatov, sistem izboljšamo s ponovnim opazovanjem procesa in določanjem novih mehkih pravil. Rezultate izboljšamo tudi z eksperimentalno spremembo pripadnostnih funkcij lingvističnih spremenljivk, že majhne spremembe le-teh lahko dajo želene rezultate.

5.3.2 Sestava baze pravil mehkega odločanja

Cilj jezikovnega opisa sistema s pravili "IF – THEN" je realizacija postopkov, ki mehke in ostre informacije na določen način predelajo v oblike, ki jih človek označuje za "verjetne". Večje število mehkih pravil povečuje točnost mehkega sistema.

V našem primeru inteligentni agent za odločanje uporablja mehko logiko in ima naslednje numerične vhode:

- **umazanost** celice v diskretiziranem okolju in
- **ceno poti**, ki jo potrebuje robot za premik.

Na osnovi vhodnih podatkov definiramo mehke spremenljivke *umazanost* in *cena poti* s pripadajočimi mehkimi vrednostmi. V določenem stanju reševanja problema se agent odloči, da počisti tisto smet, ki ima najvišjo prioriteto izmed vseh možnih smeti na seznamu. Pri tem je izhodna vrednost mehkega sklepanja *prioriteta* smeti.

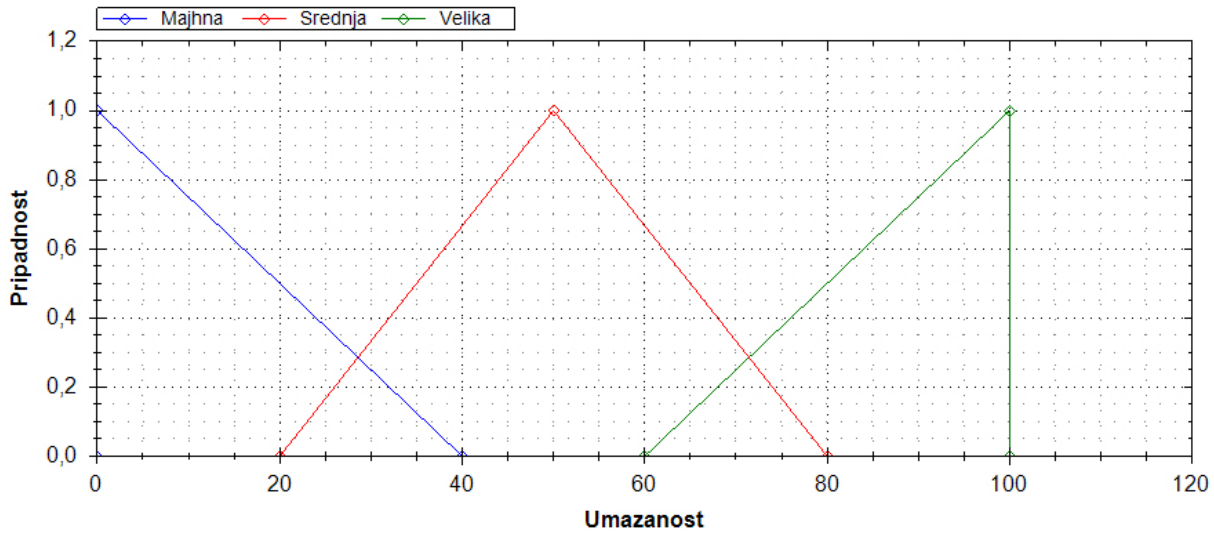
Pri sestavljanju pravil smo uporabili izkustevno znanje. Ceno poti v našem primeru dojemamo kot oddaljenost. Če je smet v neposredni bližini agenta, jo ta ne glede na stopnjo umazanosti počisti, pri "srednje" in "daleč" oddaljenih smeteh pa agent upošteva tudi stopnjo umazanosti. Dojemanje razdalj "blizu", "srednje blizu" in "daleč" je odvisno od dojemanja posameznika in okolja, za katere načrtujemo algoritem. V industrijskem okolju, ki ima nekajkrat večjo površino kot stanovanje, ima razdalja "daleč" večje razsežnosti. Pri načrtovanju mehanizma odločanja smo se omejili na stanovanjske površine in z eksperimentalnim spreminjanjem pripadnostnih funkcij določili posamezne mehke spremenljivke.

Iz zgoraj opisanih zahtev sestavimo za odločanje agenta naslednji nabor pravil:

- $\boldsymbol{IF}\ cena\ poti = nizka\ \ \boldsymbol{THEN}\ prioriteta\ = visoka$
- $\boldsymbol{IF}\ cena\ poti = srednja\ \wedge\ umazanost = velika\ \boldsymbol{THEN}\ prioriteta = srednja$ 3
- $\boldsymbol{IF}\ cena\ poti = srednja\ \wedge\ umazanost = srednja\ \boldsymbol{THEN}\ prioriteta = srednja$ 2
- $\boldsymbol{IF}\ cena\ poti = srednja\ \wedge\ umazanost = majhna\ \boldsymbol{THEN}\ prioriteta = srednja$ 1
- $\boldsymbol{IF}\ cena\ poti = visoka\ \wedge\ umazanost = velika\ \boldsymbol{THEN}\ prioriteta = nizka$ 3
- $\boldsymbol{IF}\ cena\ poti = visoka\ \wedge\ umazanost = srednja\ \boldsymbol{THEN}\ prioriteta = nizka$ 2
- $\boldsymbol{IF}\ cena\ poti = visoka\ \wedge\ umazanost = majhna\ \boldsymbol{THEN}\ prioriteta = nizka$ 1

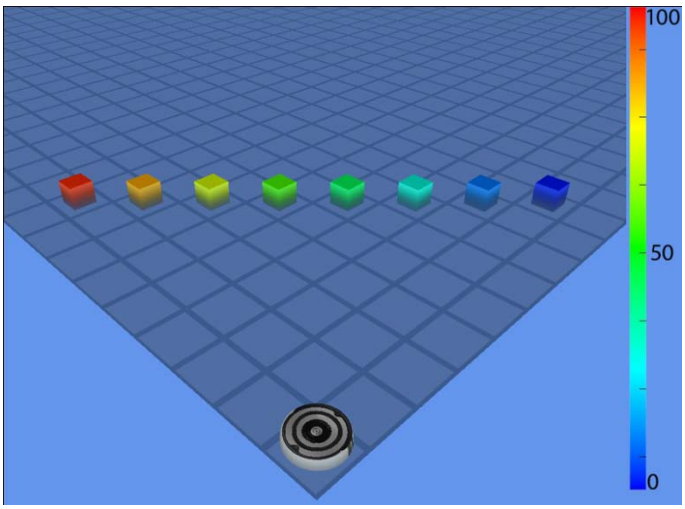
5.3.3 Določanje vhodnih in izhodnih lingvističnih spremenljivk

Za prvo vhodno lingvistično spremenljivko 'onesnaženost' uvedemo tri opisne razrede – majhna, srednja in velika. Vsakemu razredu pripada ustrezna mehka množica M_{majhna}, $M_{srednja}$ in M_{velika}. Definicijsko območje za umazanost je podano v odstotkih v intervalu od 0 % do 100 %. Grafi pripadnostnih funkcij za vse tri mehke množice so na sliki 43.



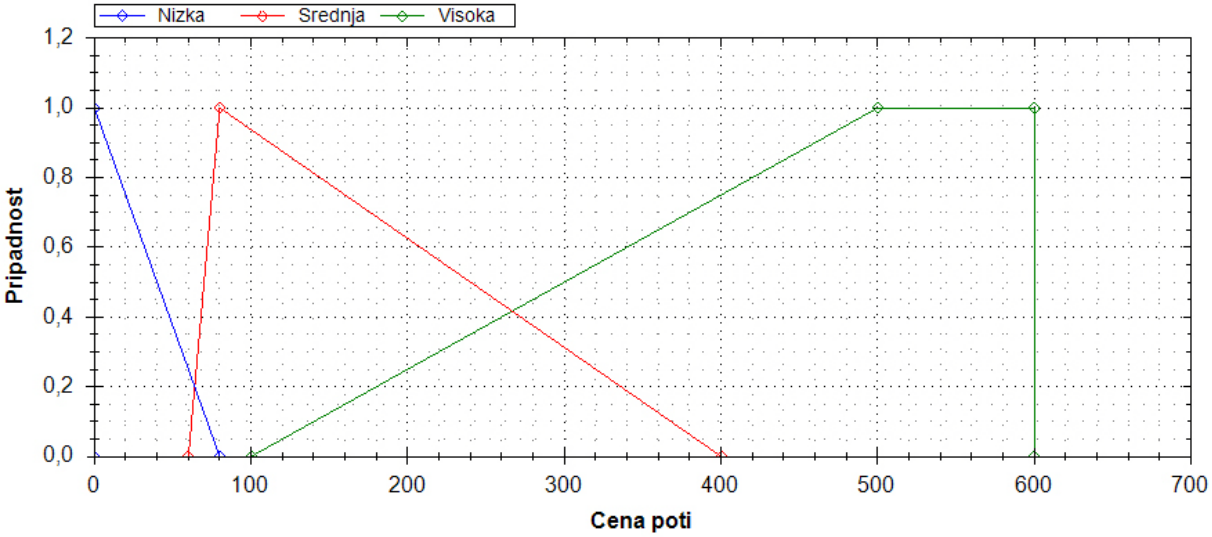
Slika 43: vhodna mehka spremenljivka 'umazanost'.

V navideznem okolju umazanost določene diskretizirane celice oziroma smeti prikazujemo s kocko ustrezne barve. Z rdečo barvo označujemo 100 % umazanost in z modro 0 % umazanost, ostale vrednosti zavzemajo vmesne barve, ki so prikazane na barvni skali slike 44.



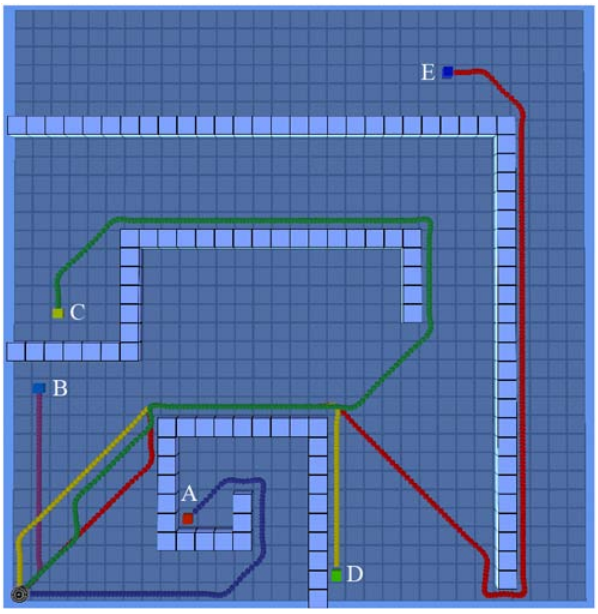
Slika 44: prikaz smeti z barvami v navideznem okolju.

Za drugo vhodno spremenljivko, ceno poti, uvedemo prav tako tri opisne razrede – nizka, srednja in visoka. Vsakemu razredu pripada ustrezna mehka množica M_{nizka}, $M_{srednja}$ in M_{visoka}. Grafi pripadnostnih funkcij za vse tri mehke množice so na sliki 45.



Slika 45: vhodna mehka spremenljivka 'cena poti'.

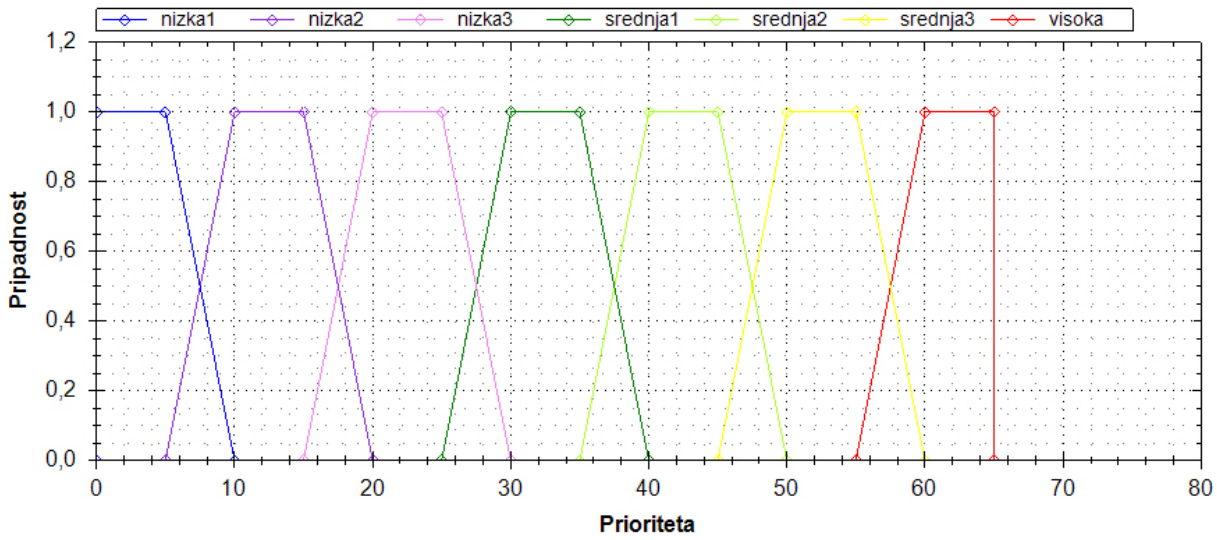
V navideznem okolju agent v eni iteraciji s klicem algoritma A* izračuna cene poti do vseh smeti, ki so na seznamu (slika 46).



Slika 46: izračun cene poti do vseh smeti v eni iteraciji.

Za izhod uvedemo lingvistično spremenljivko 'prioriteta'. Ta spremenljivka označuje prioriteto smeti, s katero sklepamo nujnost čiščenja. Celotno definicijo spremenljivke

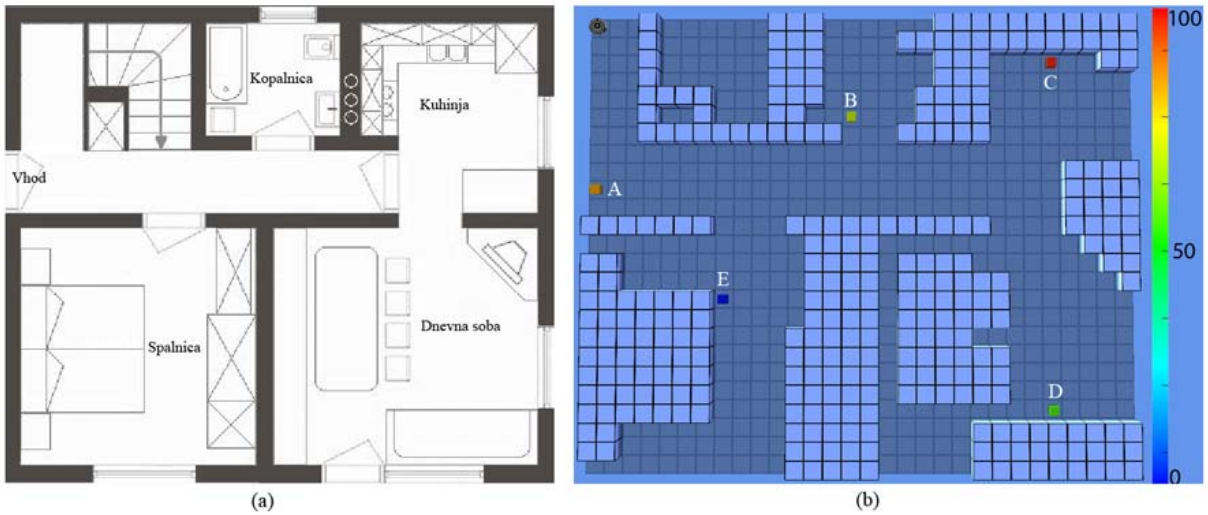
'prioriteta' s sedmimi mehkimi vrednostmi prikazuje slika 47, kjer smo kot razpon numeričnih ocen prioritet izbrali interval [0,70].



Slika 47: vhodna mehka spremenljivka 'cena poti'.

Primer mehkega odločanja inteligentnega agenta v simulaciji

V naslednjem primeru prikazujemo odločanje inteligentnega agenta. Za primer navideznega okolja smo diskretizirali stanovanjsko enoto (slika 48a). Pri generiranju seznama smeti iz izkustvenih znanj z lingvistični spremenljivkami sklepamo, da bo umazanost tal v kuhinji 'visoka'. Manj umazana bodo tla v kopalnici in pri vhodnih vratih. V dnevni sobi predpostavljamo, da bo onesnaženost tal 'srednja' in v spalnici 'nizka'. Temu primerno porazdelimo smeti po prostoru (slika 48b).

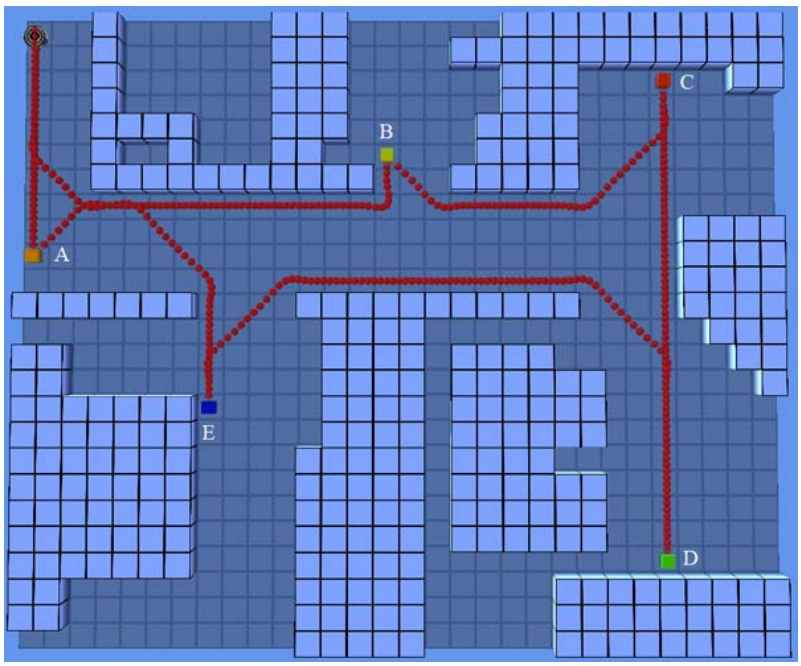


Slika 48: a) tloris stanovanjske enote in b) diskretizacija.

V prvem koraku se inteligentni agent odloči za smet A, ki ima najvišjo prioriteto. Agent se premakne na lokacijo smeti A in jo počisti. V drugem koraku s trenutne lokacije izračuna nove prioritete vseh smeti in se odloči za smet B, ki je v kopalnici. Po vsaki počiščeni smeti agent znova ovrednoti prioritete smeti na seznamu. Smeti B sledi smet C v tretjem koraku, smet D v četrtem koraku in smet E v petem koraku. Ko je seznam smeti prazen, se agent premakne nazaj k bazni postaji. Trajektorija celotne poti je prikazana na sliki 49. Pri izračunavanju poti algoritem A* uporablja hevristiko Manhtattan. Defuzifikacijo opravimo po enačbi 4.21 oziroma po težiščni metodi. Tabela 4 prikazuje numerične vrednosti vhodnih in izhodnih mehkih spremenljivk posamezne smeti v posameznem koraku. Vhod 1 predstavlja umazanost, vhod 2 predstavlja ceno poti in izhod 1 predstavlja prioriteto.

	mehka spr.	**1.korak**	**2.korak**	**3.korak**	**4.korak**	**5.korak**
A	vhod 1	80				
	vhod 2	90,0				
	izhod 1	52,50				
B	vhod 1	72	72			
	vhod 2	218	168			
	izhod 1	33,06	34,62			
C	vhod 1	90	90	90		
	vhod 2	340	291	151		
	izhod 1	29,29	33,76	46,30		
D	vhod 1	50	50	50	50	
	vhod 2	413	346	223	190	
	izhod 1	12,63	18,98	29,52	32,91	
E	vhod 1	10	10	10	10	10
	vhod 2	191	124	135	257	305
	izhod 1	26,16	30,33	29,87	22,46	19,29

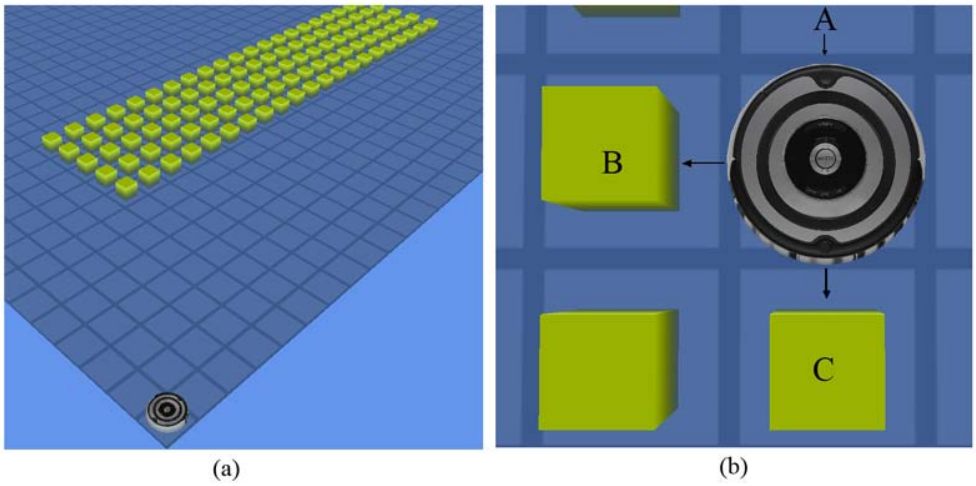
Tabela 4: posamezni koraki odločanja inteligentnega agenta.



Slika 49: trajektorija poti inteligentnega agenta.

5.3.4 Dodatne modifikacije algoritma inteligentnega agenta ter nastavitve simulacije

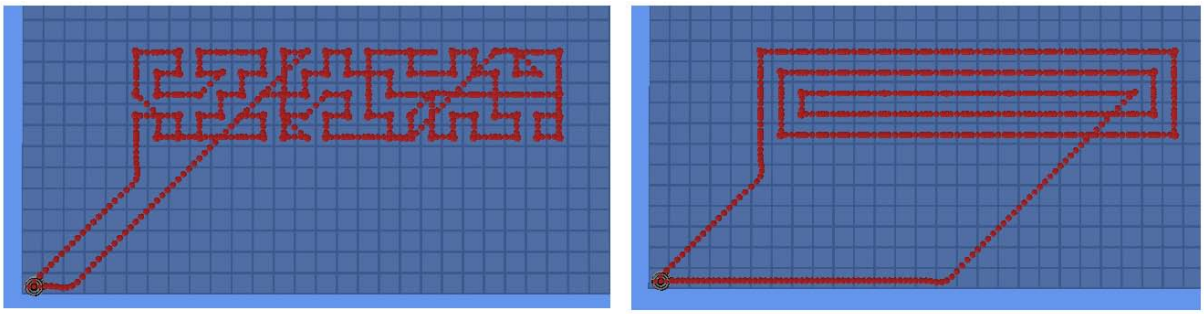
Pri realizaciji algoritma smo uvedli pravilo, ki v primeru več smeti z enako in najvišjo prioriteto na vrhu seznama smeti določi tisto, pri kateri agent ne spremeni smer svoje trajektorije. Omenjen problem se pojavlja v raporeditvi umazanih celic, kot je prikazano na sliki 50a. Glede na pravila mehke logike smo določili, da je umazanost smeti v neposredni bližini agenta nepomebna in štejemo le ceno poti. V bližini agenta tako ima lahko več smeti najvišjo in enako prioriteto.



Slika 50: a) prikaz gruče smeti in b) agent v postopku odločanja.

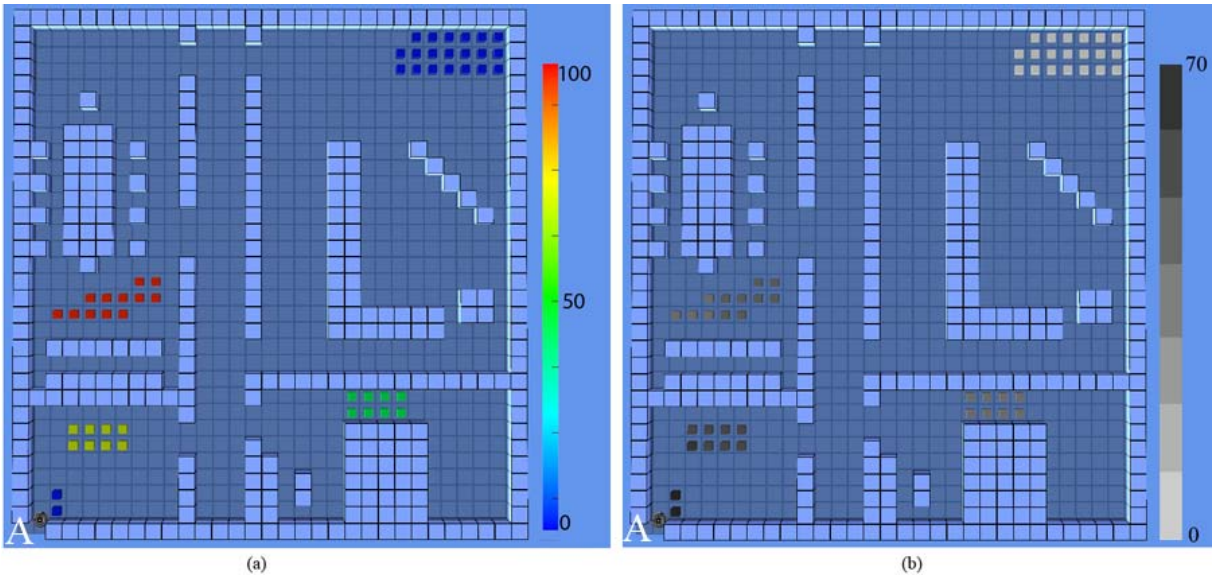
Na sliki 50 prikazujemo agenta, ki pride iz smeri A. V tem primeru imata smeti v smeri B in C enako najvišjo prioriteto, zato se bo agent odločil naključno. Ko se pojavijo dve ali več

smeti z najvišjo prioriteto na seznamu smeti, preverimo trenutno in predhodno lokacijo agenta ter lokacijo smeti. Tako dobimo tri točke iz katerih lahko preverimo ali bo agent spremenil smer. Na vrh seznama prestavimo tisto smet, pri kateri agent ne spremeni smeri. Slika 51a prikazuje trajektorijo, ko ne zaznavamo spremembe, medtem ko rezultat zaznavanja spremembe smeri prikazuje slika 51b.



Slika 51: trajektorija pri a) ne-upoštevanju spremembe in b) upoštevanju spremebe smeri.

V simulaciji lahko vključimo funkcijo, ki poleg numeričnih vrednosti prikazuje izhodno mehko vrednost "prioriteto" s sivinskimi barvami. Vsako smet pobarvamo ustrezno prioritetnemu razredu. Na sliki 52a prikazujemo razpored smeti ter na sliki 52b prioriteto z barvami. Prioriteto je določil agent v diskretni celici A. Takoj ko počisti prvo smet, bo znova izračunal prioriteto vseh smeti.

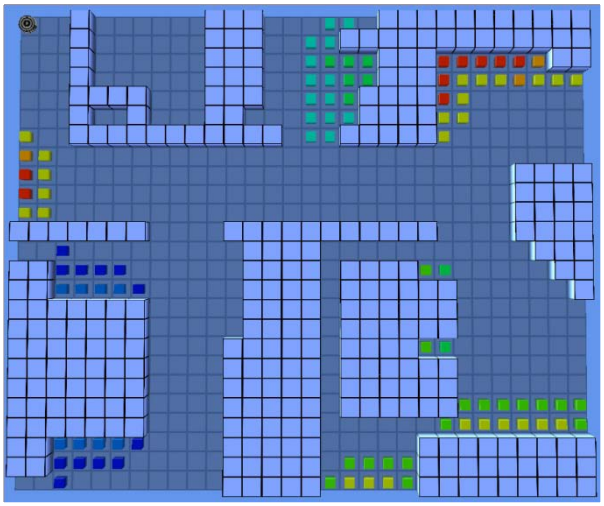


Slika 52: a) prikaz umazanega okolja in b) prikaz prioritete v sivinskih barvah.

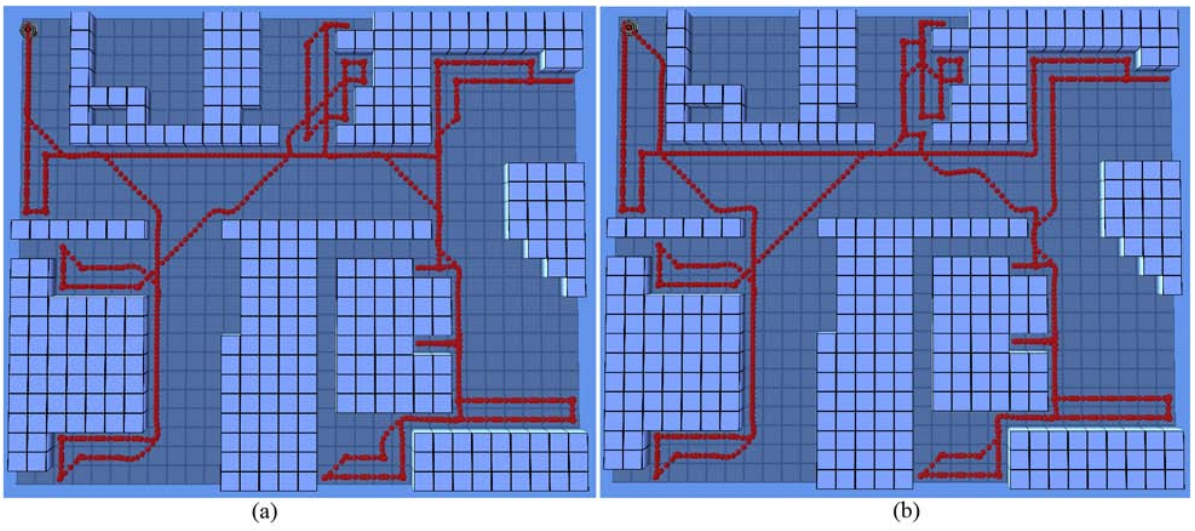
5.3.5 Analiza uporabe mehke logike kot sistema odločanja

Strategijo čiščenja okolja, realizirano po psevdo kodi iz poglavja 5.3.1, preizkusimo v treh različnih diskretiziranih okoljih. Umazanost okolja določimo naključno glede na predpostavke. Delovanje agenta opazujemo pri različnih hevristikah cen poti.

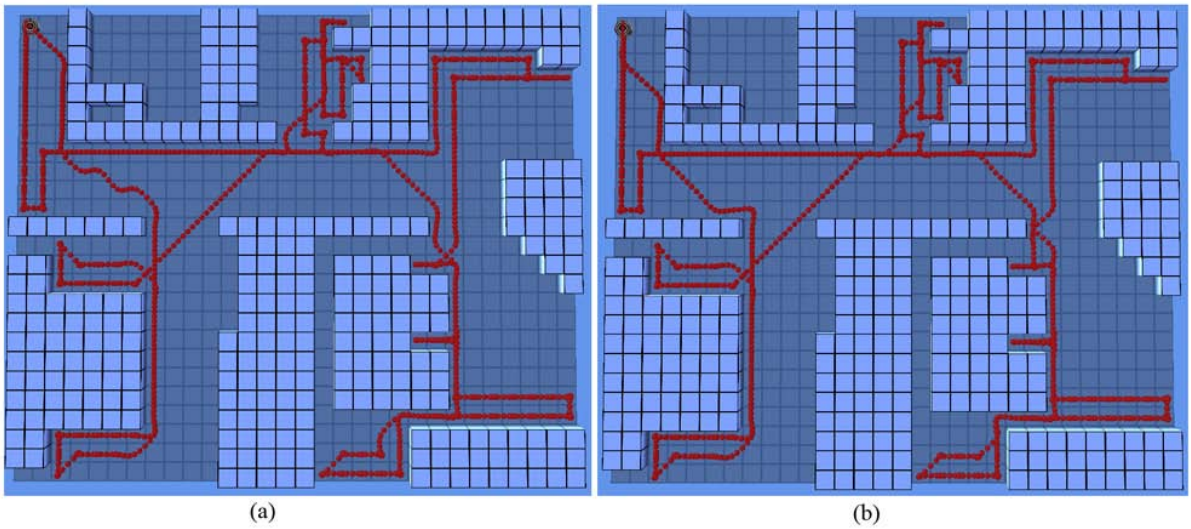
Primer 1: okolje smo diskretizirali glede na tloris iz slike 48a z mnogokotniško mrežo dimenzij 25×30 oziroma 750 celic (slika 53). 40 % celic na zemljevidu zasedenosti predstavlja ovire. Seznam smeti vsebuje 96 predmetov. Na slikah 54 in 55 so prikazane trajektorije poti, ki jih je naredil agent pri različnih hevristikah. V tabeli 5 so podane cene celotnih poti pri posameznih hevristikah.



Slika 53: diskretizirano in 'umazano' okolje.



Slika 54: trajektorija agenta s a) hevristiko Manhattan in b) Evklidovo hevristiko.

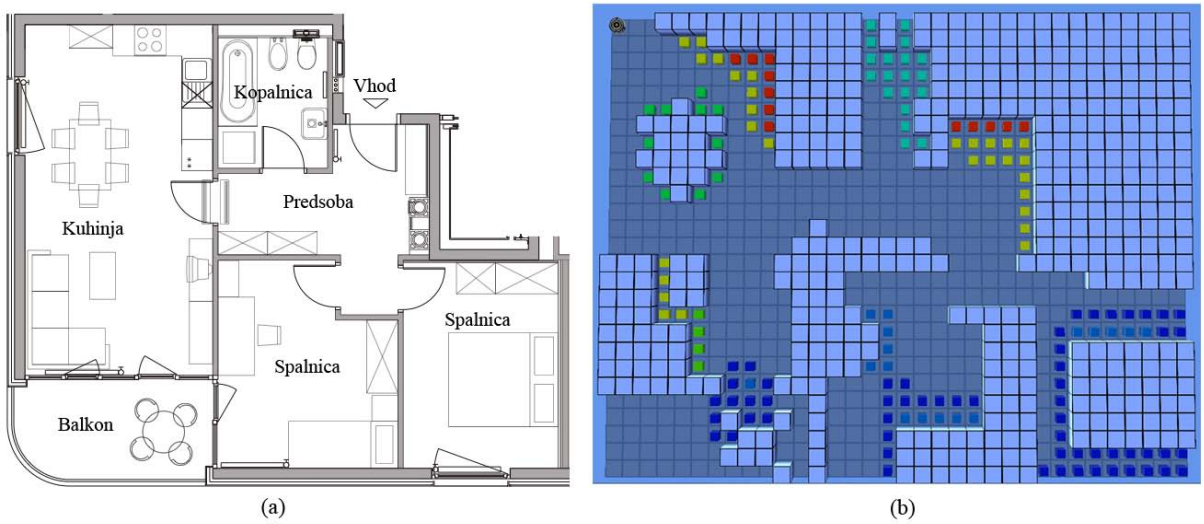


Slika 55: trajektorija agenta z a) diagonalno in b) Čebiševo hevristiko.

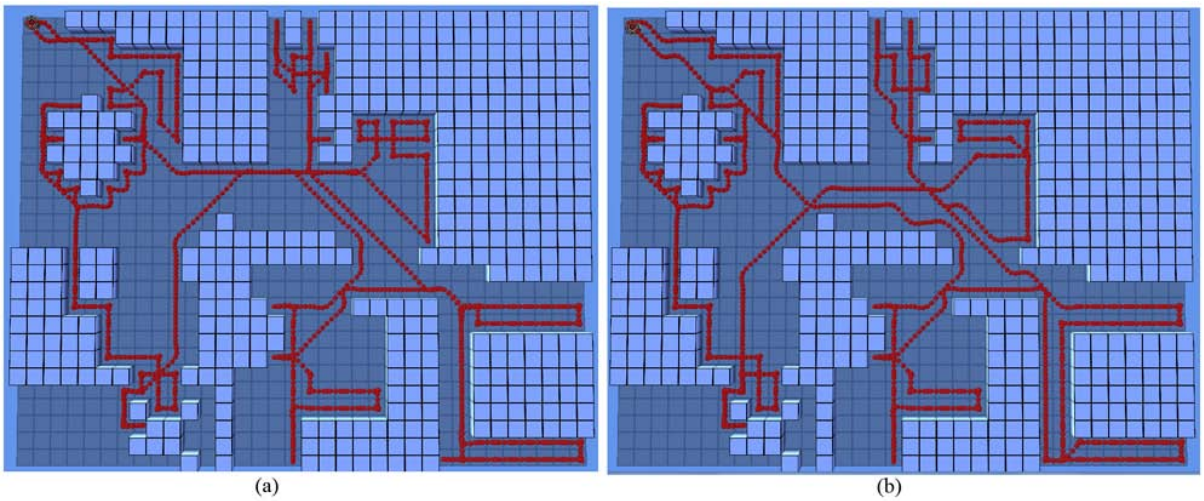
Hevristika	Manhtattan	Evklidova	Diagonalna	Čebiševa
Cena poti	2275,98	2307,70	2307,70	2307,70

Tabela 5: cene poti pri posameznih hevristikah.

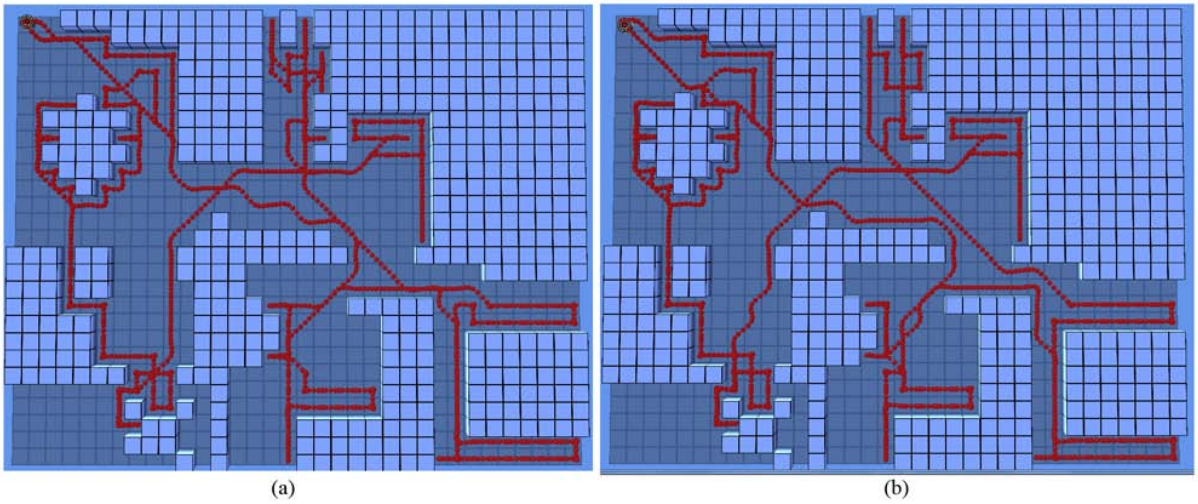
Primer 2: okolje smo diskretizirali glede na tloris na sliki 56a z mnogokotniško mrežo 27×34 oziroma 918 celic (slika 56b). 50 % celic na zemljevidu zasedenosti predstavlja ovire. Seznam smeti vsebuje 150 predmetov. Na slikah 57 in 58 so prikazane trajetkorije poti, ki jih je naredil agent pri različnih hevristikah. V tabeli 6 so podane cene celotnih poti posameznih hevristikah.



Slika 56: a) tloris stanovanjske enote in b) diskretizacija.



Slika 57: trajektorija agenta s a) hevristiko Manhattan in b) Evklidovo hevristiko.

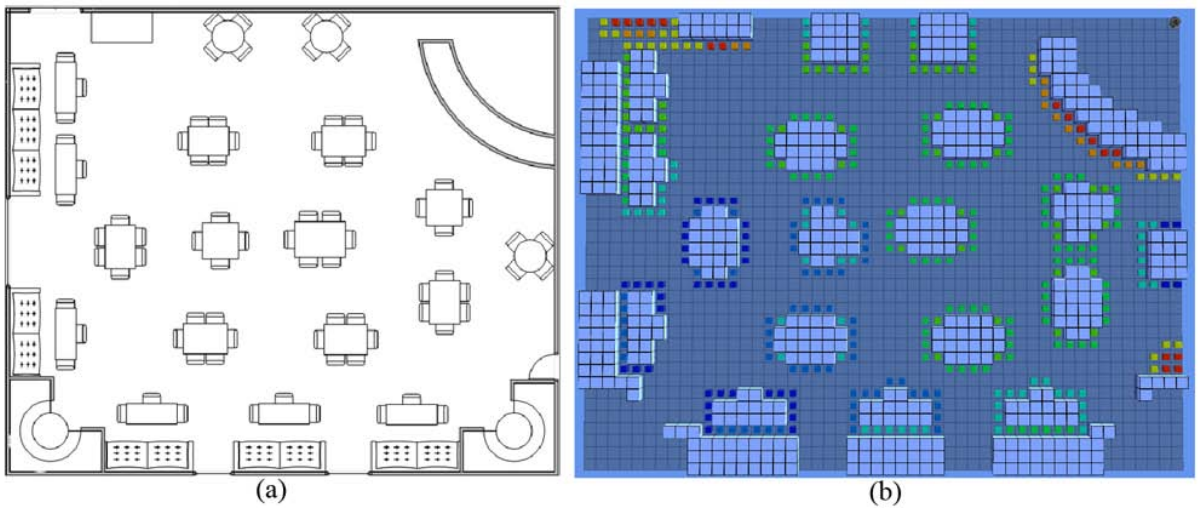


Slika 58: trajektorija agenta z a) diagonalno in b) Čebiševo hevristiko.

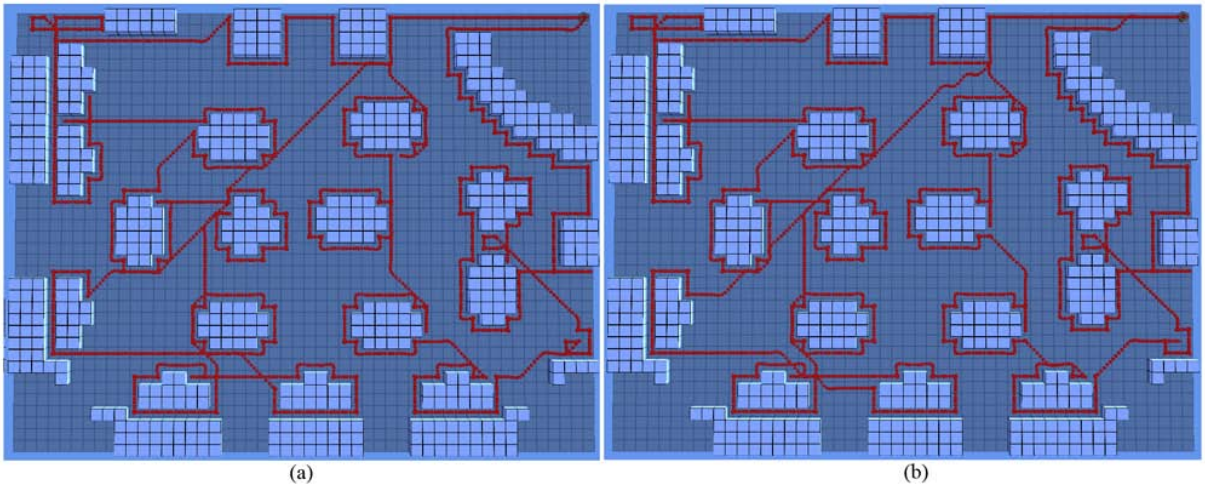
Hevristika	Manhtattan	Evklidova	Diagonalna	Čebiševa
Cena poti	3156,40	3096,40	3208,11	3202,05

Tabela 6: cene poti pri posameznih hevristikah.

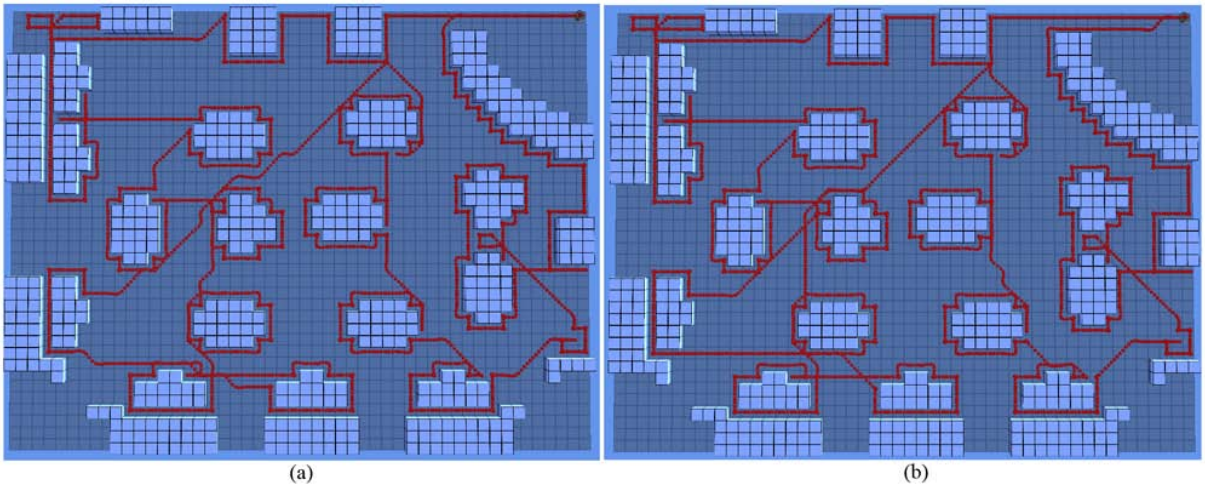
Primer 3: okolje smo diskretizirali glede na tloris na sliki 59a z mnogokotniško mrežo 50×38 oziroma 1900 celic (slika 59b). 25 % celic na zemljevidu zasedenosti predstavlja ovire. Seznam smeti vsebuje 381 predmetov. Na slikah 60 in 61 so prikazane trajetkorije poti, ki jih je naredil inteligentni agent pri različnih hevristikah. V tabeli 7 so podane cene celotnih poti posameznih hevristik.



Slika 59: a) tloris gostinske enote in b) diskretizacija.



Slika 60: trajektorija agenta s a) hevristiko Manhattan in b) Čebiševo hevristiko.



Slika 61: trajektorija agenta z a) Evklidovo in b) diagonalno hevristiko

Hevristika	Manhtattan	Evklidova	Diagonalna	Čebiševa
Cena poti	6169,53	6163,62	6183,67	6163,67

Tabela 7: cene poti pri posameznih hevristikah.

V vseh simulacijskih tekih je algoritem uspešno odstranil vse smeti na seznamu. To pomeni, da je s procesom mehkega odločanja opravil odločanje za vsako smet in z algoritmom za iskanje poti načrtoval pot do njih. Ker želimo, da algoritem izračuna najkrajšo možno pot pri izbrani odločitvi, primerjamo med seboj vpliv različnih hevristik na ceno poti. Rezultati kažejo, da je cena poti uporabljene hevristike odvisna od ovir v okolju. V prvem primeru izračuna najnižjo celotno ceno hevristika Manhtattan, v preostalih dveh primerih pa Evklidova hevristika. Če upoštevamo še rezultate časa računanj iz poglavja 5.2, sklepamo, da je hevristika Manhtattan najbolj primerna za uporabo. To še posebej velja v okoljih, v katerih je seznam smeti obsežen, saj v posamezni iteraciji algoritem opravlja klic algoritma A* za vsako smet posebej. Kvadrirane Evklidove hevristike nismo uporabili zaradi omenjenga problema v poglavju 5.2.

Z opisanim algoritmom smo dosegli učinkovito odločanje inteligentnega agenta in načrtovanja poti zanj. Sistem mehke logike bi lahko parametrizirali z več vhodnimi spremenljivkamim, povečali bazo pravil in tako povečali natančnost sistema. Z več vhodnimi spremenljivkami bi upoštevali več kriterijev, ki se lahko pojavijo pri umazanem okolju in čiščenju le-tega. Strategijo čiščenja bi tako lahko izpopolnili z nadaljnim raziskovalnim delom.

6. Zaključek

Cilj diplomske naloge je bila realizacija inteligentnega agenta, sposobnega odločanja in iskanja poti v diskretiziranem okolju. Pri tem smo predstavili uporabo mehke logiko v sistemih odločanja in algoritem A* pri načrtovanju poti. Primer smo analogno predstavili kot avtonomnega robotskega sesalnika v inteligentnem okolju, ki izvaja napredno strategijo čiščenja okolja. Izvedli smo programsko rešitev simulacije za analizo strategije.

Prednost uporabe mehke logike je, da za načrtovanje ne potrebujemo natančnega matematičnega modela sistema. Slabost pristopa je pisanje pravil, ki zahteva izkustveno znanje. Kot kriterij v mehanizmu sklepanja mehke logike smo uporabili rezultate algoritma A*. Izkazalo se je, da izbira hevristike vpliva tudi na potek odločanja. Analizirali smo rezultate simulacije in določili hevristiko Manhtattan za najbolj primerno, saj z njo algoritem izračuna pot opazno hitreje od ostalih uporabljenih hevristik, pri tem pa je cena poti primerljiva. S kombinacijo izkustvenega znanja in verbalne komunikacije smo uspešno realizirali sistem, ki posnema človeško strategijo vodenja. V nalogi smo uporabili dva vhoda in en izhod. Izbira mehke logike se je izkazala za primeren mehanizem odločanja.

V prvem delu naloge smo predstavili obstoječe projekte na področju inteligentnih okolij, ki temeljijo na omrežju robotskih sistemov, in nakazali možnost uporabe obravnavanih algoritmov v takšnih okoljih. Z nadaljnim razvojem bodo inteligentna okolja v prihodnosti cenovno dostopnejša, predvsem pa bolj funkcionalna od trenutnih avtomatiziranih gospodinjskih naprav, ki delujejo kot izolirane enote. Funkcionalnost in učinkovitost pri avtomatiziranih opravilih bo odvisna od uporabljenih programskih rešitev. Prednosti in možne izboljšave sesalnika, ki izkorišča funkcionalnosti inteligentnega okolja v primerjavi s trenutnimi komercialnimi inteligentnimi sesalniki, je:

- globalno pozicioniranje sesalnika v okolju,
- inteligentno okolje lahko samodejno določi, katerih prostorov se ne sme pospravljati, tako da zemljevid dinamično posodbi z ovirami (npr. ponoči se lahko sesalnik zadržuje le v kuhinji in dnevni sobi, medtem ko v spalnico nima vstopa),
- smet lahko počisti takoj, ko jo inteligentno okolje doda na seznam, celovito čiščenje tal pa opravlja le po potrebi ali občasno,

- med premikanjem ima lahko motorje za sesanje izključene in jih vključi le na lokacijah smeti,
- glede na umazanost regulira moč sesanja,
- sesalnik lahko pošlje zahtevo okolju, da mu pomaga premostiti ovire, npr. odpreti robotska vrata, premakniti oviro itd.,
- sesalnik se lahko izogiba prostorom, kjer se nahaja človek itd.

Obstoječe delo lahko uporabimo za prikaz in analizo odločanja inteligentnih sistemov ter iskanja in načrtovanja poti v mobilni robotiki. Nadaljni razvoj algortima za energijsko varčno strategijo čiščenja je lahko osnova za razvoj realnega sistema, kot je PEIS ekologija, vendar trenutno ne predstavlja celovite programske rešitve, predvsem iz vidika varnosti v človeškem okolju. Pri slednjem je ključnega pomena upoštevanje dinamične ovire, kot je človek, ki pa je v algoritem nismo vključili.

7. Literatura

[1] http://www.aaai.org/AITopics/pmwiki/pmwiki.php/AITopics/AIOverview, (dostopnost preverjena januarja 2010).

[2] Guid, N.: Umetna inteligenca, Univerza v Mariboru, 2007.

[3] Prassler, E., Ritter, A., Schaeffer, C., Fiorini, P.: A Short History of Cleaning Robots, Autonomous Robots, Special Issue on Cleaning and Housekeeping Robots, vol. 9, izvod 3, December 2000.

[4] Saffiotti A., Broxvall M.: PEIS Ecologies: Ambient Intelligence meets Autonomous Robotics, Proc. of the sOc-EUSAI conference on Smart Objects and Ambient Intelligence. Grenoble, Francja, Oktober 2005.

[5] http://www.ideafinder.com/history/inventions/vacleaner.htm, (dostopnost preverjena januarja 2010).

[6] http://www.irobot.com/ , (dostopnost preverjena januarja 2010).

[7] http://trilobite.electrolux.com/ , (dostopnost preverjena januarja 2010).

[8] Sanfeliu A., Hagita N., Saffiotti A.: Special issue on network robot systems, Robotics and Autonomous Systems, 2008.

[9] http://urus.upc.es/ , (dostopnost preverjena januarja 2010).

[10] http://www.scat.or.jp/nrf/English/ , (dostopnost preverjena januarja 2010).

[11] Saffiotti A., Broxvall M., Gritti M., LeBlanc K., Lundh R., Rashid J., Seo B.S., Cho Y.J.: The PEIS-Ecology Project: vision and results, Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS); Nica, Francija, September 2008.

[12] DiLello E., Saffiotti A.: The PEIS Table - An Autonomous Robotic Table for Domestic Environments. Proc. of the European Conf on Mobile Robots (ECMR). Dubrovnik, Hrvaška, 2009.

[13] Hyo-Sung A., Wonpil Y.: Wireless localization network for a ubiquitous robotic space: Background and concept, in Proc. of the 3rd International Conference on Ubiquitous Robots and Ambient Intelligence, Seul, Koreja,2006, str. 187–192.

[14] Millington, I.: Artificial Intelligence for games, Elsevier, 2006.

[15] Pavešič, N.: Razpoznavanje vzorcev: Uvod v analizo in razumevanje vidnih in slušnih signalov, 2. Izdaja, Založba FE in FRI, 2000.

[16] Jouni Smed, Harri Hakonen: Algorithms and Networking for Computer Games, John Wiley & Sons Ltd., 2006.

[17] http://theory.stanford.edu/~amitp/GameProgramming/, (dostopnost preverjena januarja 2010).

[18]http://www.cs.cmu.edu/afs/cs/academic/class/15451-s99/www/lectures/lect0223, (dostopnost preverjena januarja 2010).

[19] Dobnikar, A.,Šter, B.: Mehko računanje, Založba FE in FRI, 2008.

[20] http://msdn.microsoft.com/en-us/library/bb200104.aspx, (dostopnost preverjena januarja 2010).

[21] Hanak D., Takacs B.: A prototype home robot with an ambient facial interface to improve drug compliance, Journal of Telemedicine and Telecare, 2008.