UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Rok Kek

Kvalitativno robotsko planiranje potiskanja predmetov

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: akad. prof. dr. Ivan Bratko

Ljubljana, 2016

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Rok Kek,

z vpisno številko 63050142,

sem avtor diplomskega dela z naslovom:

Kvalitativno robotsko planiranje potiskanja predmetov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom akad. prof. dr. Ivana Bratka;
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela;
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 30.08.2016 Podpis avtorja:

Zahvala

Na tem mestu se zahvaljujem mentorju profesorju Ivanu Bratku za njegovo vodenje pri izdelavi diplomskega dela.

Zahvala gre tudi Domnu Šoberlu za napotke pri uporabi njegovega simulatorja.

Zahvaljujem se partnerki Niki in hčerkici Zarji, ki sta mi ves čas stali ob strani.

Zahvalil bi se tudi svojim staršem, očetu Tonetu in mami Alenki, in Nikinim staršem Borisu in Kseniji, za vso njihovo podporo; vse vas imam rad.

Zahvaljujem se tudi vsem ostalim, ki so kakorkoli pripomogli k uspešni izvedbi diplomskega dela.

Kazalo

Seznam uporabljenih kratic in simbolov

planirano stanje - je stanje v planu, ki ga planer napove, da se bo zgodil, v primeru da sledimo zaporedju proženja akcij

regulator PID - kontrolni mehanizem, ki temelji na povratni zanki (angl. Proportional-Integral-Derivative controller)

QDE - kvalitativna diferencialna enačba (angl. Qualitative Differential Equation)

QMD - razdalja kvalitativne veličine (angl. Qualitative Magnitude Distance)

QMDpD - razdalja kvalitativne veličine z upoštevanjem smeri (angl. Qualitative Magnitude Distance plus Direction)

QSIM - kvalitativni simulator (angl. Qualitative SIMulation algorithm)

QUIN - algoritem za učenje kvalitativnih modelov (angl. QUalitative INduction)

lažno obnašanje - generirano kvalitativno obnašanje, ki v resnici ne obstaja (angl. spurious behaviour)

Povzetek

Diplomska naloga sodi na področje umetne inteligence, kvalitativnega sklepanja in robotike. Namen dela je uporabiti kvalitativni simulator pri planiranju kvalitativnih akcij robota. Modifikacija algoritma QSIM generira prostor stanj, katerega preiskujemo z algoritmom za hevristično preiskovanje A*. Implementacija vseh algoritmov je napisana v programskem jeziku prolog. Nekateri algoritmi učenja gradijo kvalitativne modele, katere bi želeli uporabiti za potrebe planiranja, vendar ti modeli vsebujejo omejitve QDE, ki niso definirane v orginalnem algoritmu QSIM. Ena od takih omejitev je monotona odvisnost od več spremenljivk. Take monotonostne omejitve smo implementirali in testirali na umetni domeni. Plani so bili preizkušeni na simulatorju potiskanja predmeta, ki temelji na fizikalnem pogonu Box2D. Implementiran je bil izvajalec plana v prologu, ki plan, generiran s planiranjem, tudi izvede. Razvit je bil vmesnik, ki planerju in izvajalcu plana omogoča komunikacijo s simulatorjem. Vmesnik je zadolžen za konverzijo numeričnih podatkov v kvalitativne in za implementacijo kvalitativnih akcij oziroma njihovega izvajanja na simulatorju. Delovanje algoritma je bilo preizkušeno na dveh domenah potiskanja predmetov: na primeru potiskanja pokončnega valja in na primeru potiskanja kocke. Za vsako domeno je bil ročno zgrajen kvalitativni model. Na koncu diplomskega dela predstavimo rezultate preizkusov. Diplomsko nalogo zaključimo s pregledom doseženih ciljev, pregledom morebitnih izzivov pri implementaciji algoritmov in pregledom idej za nadaljnje raziskave na obdelanem področju.

Ključne besede:

umetna inteligenca, kvalitativna simulacija, QSIM, QDE, kvalitativni model, planiranje, robotika, potiskanje predmetov.

Abstract

The thesis belongs to the field of Artificial Intelligence, robotics and qualitative reasoning. The purpose of the work is to use a qualitative simulator for planning qualitative actions of a robot. Our modification of the known QSIM algorithm generates state space, which we search with the heuristic search algorithm A*. Implementations of all algorithms are written in the programming language Prolog. Some machine learning algorithms induce qualitative models using QDE constraints that are not defined in the original QSIM algorithm. One of these QDE constraints is the monotonicity in multiple variables. This QDE constraint was implemented and tested on an artificial domain. Generated robot plans have been tested on an object pushing simulator, which is based on the Box2D engine. For this purpose, an algorithm for plan execution was developed. This plan execution algorithm communicates through an interface, which was also developed as part of the thesis. The interface is responsible for a conversion of numerical data into qualitative states. The interface also implements execution of qualitative actions on the simulator. Plans developed by the proposed algorithm have been tested in two object pushing domains: the case of pushing a vertical cylinder and the case of pushing a block. For this purpose, there was a hand-built qualitative model for each domain. The thesis is concluded with an examination of achieved objectives, a review of potential challenges in the implementation of algorithms and a review of ideas for further research.

Key words:

artificial intelligence, qualitative simulation, QSIM, QDE, qualitative model, planning, robotics, object pushing.

Poglavje 1

Uvod

Začetek uporabe kvalitativnih modelov sega v osemdeseta leta, ko je bil predlagan algoritem planiranja zaporedja akcij s pomočjo kvalitativnega modela [8]. Predstavljeni algoritem je uspešno sestavil plan, kako odpirati ventil, da se z vodo napolni posoda do določenega nivoja. Kasneje je drugi algoritem z uporabo kvalitativnega modela in podobno definiranih akcij sestavil plan, ki iz začetnega stanja praznega lonca je privedel do končnega stanja vretja vode [6]. V domeni potiskanja predmetov je bilo izvedeno učenje kvalitativnega modela z algoritmom QUIN [12]. Naučeni kvalitativni model je bil uporabljen za sestavo plana, s katerim so upravljali robota tako, da je potisnil kocko na ciljno pozicijo [12]. S podobnim naučenim modelom je bil predlagan pristop potiskanja predmeta, čigar tloris je poljuben konveksen poligon [18]. Planiranje s pomočjo kvalitativnega modela je bilo uporabljeno tudi v [15], kjer so s pomočjo kvalitativnega modela izdelali plan, s katerim je robot preplezal stopnico.

1.1 Cilji diplomske naloge

- Razširiti algoritem QSIM za potrebe planiranja;
- implementirati dodatne omejitve, ki jih uporablja algoritem QUIN in niso del osnovne implementacije QSIM;
- implementirati izvajanje plana na simulatorju;
- testirati algoritem na dveh primerih potiskaja predmetov, na primeru potiskanja pokončnega valja in na primeru potiskanja kocke.

1.2 Gradnja kvalitativnega modela

Nekaj razlogov, zakaj gradimo kvalitativne modele je:

- ker so bližji človekovem razmišljanju;
- ker morda točne relacije niso znane (npr. v psihologiji);
- ker morda numerične parametre težko izmerimo;
- ker bi bil ustrezni numerični model računsko prezahteven.

Večino kvalitativnih modelov ljudje zgradimo na podlagi izkušenj oziroma predhodnega znanja. Ob razmahu računalniške tehnologije in velikih količin podatkov, ki jih z njeno pomočjo zbiramo, pa je smiselno razmišljati tudi o avtomatskem učenju kvalitativnih modelov iz podatkov [16]. Eno od uspešnih metod kvalitativnega učenja predstavlja algoritem QUIN, ki je bil predstavljen leta 2001 [19]. Algoritem je bil uspešno uporabljen v več primerih učenja modelov za potrebe planiranja [12, 18]. Kasneje je bila predstavljena druga metoda kvalitativnega učenja, algoritem Padé [17]. Algoritem temelji na iskanju parcialnih odvodov. Prednost tega algoritma je, da gradnjo kvalitativnega modela pohitri in da iz določenih podatkov bolje izlušči znanje [16].

1.3 Kvalitativni modeli in uporabljeni zapis

Implementacija vseh algoritmov je v programskem jeziku prolog [3, 13]. Zapis, ki je uporabljen v diplomski nalogi, je temu prilagojen. Stanje kvalitativnega sistema lahko opišemo s seznamom kvalitativnih spremenljivk, kot prikazuje enačba 1.1. Vsaka spremenljivka ima obliko prikazano v enačbi 1.2.

$$[QVar1, QVar2, ..., QVarN] \tag{1.1}$$

$$QVar = Dom : QMag/Dir \tag{1.2}$$

$QMag$ je kvalitativna veličina spremenlijivke. Vrednost, ki jo lahko zasede, je odvisna od domene Dom. Če je domena spremenljivke $QVar$ enaka Dom in so $l_0, ..., l_m$ odlikovane vrednosti domene, prikazane v enačbi 1.3, lahko $QMag$ zavzame vrednosti, prikazane v enačbi 1.5.

$$dom := [l_0, .., l_n, .., l_m] \tag{1.3}$$

$$l_0 = minf, l_n = zero, l_m = inf \tag{1.4}$$

$$QMag \in \{l_i | 0 \leq i \leq m\} \cup \{l_i..l_{i+1} | 0 \leq i < m\} \tag{1.5}$$

Dir je smer spreminjanja kvalitativne spremenljivke. Tabela 1.1 prikazuje vrednosti, ki jih lahko zavzame:

Dir	**pomen**
inc	vrednost spremenljivke se povečuje
std	vrednost spremenljivke se ne spreminja
dec	vrednost spremenljivke se zmanjšuje

Tabela 1.1: Vrednosti določila Dir kvalitativne spremenljivke

Kvalitativni model je skupek kvalitativnih spremenljivk in omejitev, ki opisujejo obnašanje sistema. Kvalitativne omejitve so predstavljene kot QDE (angl. Qualitative Differential Equation). Tabela 1.2 prikazuje nekaj običajnih omejitev QDE.

QDE	**opis**
$M^+(X,Y)$	Y monotono narašča z X
$M^-(X,Y)$	Y monotono pada X
$sum(X,Y,Z)$	$Z = X + Y$
$mult(X,Y,Z)$	$Z = X \times Y$
$deriv(X,Y)$	Y je odvod X-a po času
$const(X,k)$	$X = x : k/std$

Tabela 1.2: Običajne omejitve QDE

Omejitve QDE navadno veljajo le za del prosotora opisanega s kvalitativnimi spremenljivkami. Tako na podlagi vrednosti kvalitativnih spremenljivk prostor razdelimo na takoimenovana operativna območja [4]. Ta se lahko med seboj prekrivajo. Kvalitativni model je tako predstavljen kot seznam operativnih območij s pripadajočimi omejitvami QDE.

Poglavje 2

Implementacija planerja

Algoritem planiranja je implementiran v programskem jeziku prolog. Temelji na algoritmu za hevristično preiskovanje prostora A* [5] in algoritmu za kvalitativno simulacijo QSIM [10]. Implementacijo obeh algoritmov lahko najdemo v knjigi [3]. Definicije predikatov, ki predstavlajo dodatke ali spremembe algoritma QSIM, so napisane v tem poglavju. Manjkajoči predikati pa so definirani v dodatku A.

Algoritmu QSIM so bili dodani novi prehodi med kvalitativnimi stanji sistema preko kvalitativnih akcij. Tako razširjeni algoritem generira prostor stanj, katerega preiskujemo z algoritmom A*. Za hevristično iskanje je bila izbrana primerna hevristična funkcija.

2.1 Kvalitativna simulacija

Algoritem za kvalitativno simulacijo, QSIM, v zanki pobira stanja z seznama aktivnih stanj in generira vsa možna stanja, ki ga nasledjo. Potencialne naslednike generira s pomočjo tabele 2.1. QSIM filtrira generirana stanja z omejitvami v kvalitativnem modelu. Tista stanja, ki so v skadu z omejitvami QDE, doda v seznam aktivnih. Ker se lahko zgodi, da je možnih več naslednikov aktivnega stanja, QSIM gradi drevo vseh možnih obnašanj sistema. Algoritmu postavimo naše začetno stanje v seznam aktivnih stanj in poženemo algoritem. Slednji bo tekel, dokler se seznam ne bo izpraznil, ozioma dokler ne preseže omejenosti resursov.

Obstajata dva tipa prehodov med stanji:

- P-prehodi, ki premaknejo veličino spremenljivke s točkovne vrednosti na interval;

- I-prehodi, ki premaknejo veličino spremenljivke z intervala na točkovno vrednost.

prehod-P	**Aktivno stanje**	**Novo stanje**
P1	L_i/std	L_i/std
P2	L_i/std	$L_i..L_{i+1}/inc$
P3	L_i/std	$L_{i-1}..L_i/dec$
P4	L_i/inc	$L_i..L_{i+1}/inc$
P5	$L_i..L_{i+1}/inc$	$L_i..L_{i+1}/inc$
P6	L_i/dec	$L_{i-1}..L_i/dec$
P6	$L_i..L_{i+1}/dec$	$L_i..L_{i+1}dec$
prehod-I		
I1	L_i/std	L_i/std
I2	$L_i..L_{i+1}/inc$	L_{i+1}/std
I3	$L_i..L_{i+1}/inc$	L_{i+1}/inc
I4	$L_i..L_{i+1}/inc$	$L_i..L_{i+1}/inc$
I5	$L_i..L_{i+1}/dec$	L_i/dec
I6	$L_i..L_{i+1}/dec$	L_i/inc
I7	$L_i..L_{i+1}/dec$	$L_i..L_{i+1}/std$

Tabela 2.1: Tabela možnih prehodov med sosednjimi stanji

2.1.1 Matematične lastnosti QSIM

- Z danimi QDE in začetnim stanjem, QSIM generira vsa možna obnašanja sistema.
- Nekatera obnašanja, ki jih generira QSIM, se nikoli ne pojavijo na resničnem dinamičnem sistemu in celo kršijo kvalitativne omejitve v danem modelu. Takim obnašanjem pravimo lažna obnašanja. To so dejansko napačni rezultati, ki jih algoritem QSIM generira zaradi svojih znanih pomanjkljivosti.

2.2 Razširitev zapisa

Akcije lahko spreminjajo stanje hipno, torej nezvezno, za razliko od običajne QSIM simulacije, ki predpostavlja, da so vse spremembe vrednosti zvezne in

gladke. Na določenih spremenljivkah $Dom : Mag/Dir$ kvalitativnega modela, smer spreminjanja Dir nima pomena oziroma je lahko celo zavajujoča. Uvedemo nov tip spremenljivke $Dom : Mag$, ki se razlikuje od drugih spremenljivk po tem, da nima določila spreminjanja smeri. Take spremenljivke imenujemo kontrolirane spremenljivke. Njihovo stanje ni simulirano, vendar ga spreminjajo le akcije. Predikat $controlled/1$ definira prepozanvanje kontroliranih spremenljivk.

```
controlled(_Dom:QM):-
    QM \= _Qm/_Dir,!.
```

2.3 Akcije

Akcije definiramo kot predikat $action/3$. Ta ima tri argumente:

- pogoj - lastnosti stanja, na katerem se akcija lahko proži;
- učinek - spremembe stanja;
- klic - parametrizirano sporočilo, ki sproži izvajanje.

Ker sta pogoj in učinek definirana na kvalitativnem stanju, jih bomo imenovali kvalitativne akcije. Klic akcije je sporočilo, ki ga pošljemo kontrolnemu sistemu, da se akcija izvrši. Klic kvalitativne akcije je lahko brez parametra, lahko pa ima numeričen in/ali kvalitatven parameter.

Učinek akcije je definiran kot sprememba podmnožice spremenljivk stanja. Pri spremenljivkah, katerih sprememba ni definirana v učinku akcije, kvalitativna veličina ostane nespremenjena. Lahko pa se spremeni smer spreminjanja, ki je definirana na simuliranih spremenljivkah, tako da je novo stanje v skladu z modelom. Ostala stanja spremenljivk ostanejo nespremenjena. To lastnost definira predikat $action\_for\_effect/3$, ki je uporabljen pri planiranju in izvajanju plana.

```
action_for_effect(Precondition, Effect, Call) :-
        action(Precondition, Effect, Call),
        fill(Precondition, Effect).

% na podlagi State1 stanja v celoti dolosi State2
% tako da je State2 v skladu z modelom
fill(State1, State2) :-
```

```
        fill_controlled(State1, State2),
        legalstate(State2).


% predikat na podlagi prejsnjega stanja v celoti
   doloci naslednje stanje
fill_controlled([], []).
fill_controlled([Var1|Vars1], [Var2|Vars2]) :-
        controlled(Var1),
        fill_var(Var1, Var2),
        fill_controlled(Vars1, Vars2).
fill_controlled([Dom:Mag/Dir1|Vars1], [Dom:Mag/Dir2|
   Vars2]) :-
        legal_dir(Dir1, Dir2),
        fill_controlled(Vars1, Vars2).

% smer spreminjanja le nespremenjena
legal_dir(ODir, Dir):-
    ODir == Dir,!.
% smer spreminjanja v primeru spremembe, kakrsna koli
legal_dir(_, Dir):-
    qdir(Dir,_).
```

2.4 Prehodi med kvalitativnimi stanji

Imamo dva tipa prehodov med kvalitativnimi stanji. Prvi tip prehoda se izvaja s pomočjo simulacije algoritma QSIM in ga bomo imenovali simulacijski prehod. Drugi tip prehoda pa staro stanje spremeni na podlagi učinka akcije, definirane v poglavju 2.3. Tak prehod imenujemo akcijski prehod. Ta dva tipa prehodov se med seboj razlikujeta v tem, da se sprememba v novem stanju, pri simuliranem prehodu, odraža le na spremenljivkah, ki niso kontrolirane. Stanje kontroliranih spremenljivk pa pri tem tipu prehoda ostane povsem nespremenjeno. Pri akcijskem prehodu se sprememba odraža na kontroliranih spremenljivkah. V tem primeru kvalitativna veličina simuliranih spremenljivk ostane nespremenjena, spremeni pa se lahko smer spreminjanja vrednosti simuliranih sremenljivk. Druga razlika pa je, da se robot odloči, kdaj akcijo izvesti. Posledica tega je, da se bo akcijski prehod, ki je v planu, zares zgodil. Pri simuliranem prehodu pa je točno določen prehod le eden od možnih

prehodov, zaradi lastnosti algoritma QSIM, ki lahko generira lažna obnašanja. Implementacija preiskovalnega algoritma v prologu predvideva definicijo predikata $s/3$ in definira prehod iz stanja *State* v stanje *NewState* s ceno prehoda *Cost*.

```
% s(State, NewState, Cost)
s(State, NewState, 1) :-
        (   action_for_effect(State, NewState, _)
        ;   legal_trans(State, NewState)
        ).
```

2.5 Preiskovanje prostora stanj

Rezultat algoritma QSIM je simulacijski graf, ki opisuje vse možne prehode med stanji v času od t_0 do t_n. Za potrebe planiranja mora algoritem najti pot v grafu, ki ga generira QSIM, oziroma razširjeni QSIM. Za preiskovanje prostora je bil uporabljen algoritem za usmerjeno hevrsitično iskanje A*, ki je bil predstavljen v [7], natančneje uporabljena je implementacija algoritma, ki je opisan v dvanajstem poglavju knjige [3]. Algoritem uporablja tipično cenovno funkcijo, ki je definirana za vsako stanje s kot $f(s) = g(s) + h(x)$. Funkcijo $g(x)$ imenujemo trenutna cena in je definirana kot cena od začetnega stanja do trenutnega stanja. Ker je cena vsakega prehoda enaka eni enoti, je trenutna cena enaka številu stanj, to je dolžini poti. Funkcijo $h(x)$ definiramo kot oceno za preostalo ceno za pot iz trenutnega stanja do cilja. Hevristična funkcija, ki smo jo uporabili je zelo podobna TotalQMD, ki je bila predlagana v [14]. TotalQMD temelji na QMD (angl. Qualitative Magnitude Distance). QMD ocenjuje minimalno število kvalitativnih stanj, ki jih mora kvalitativna spremenljivka zavzeti, da preide iz enega kvalitativnega stanja $S_1 = Mag_1/Dir_1$ v drugo kvalitativno staje $S_2 = Mag_2/Dir_2$. QMD je definirana z enačbo:

$$\begin{aligned}
\mathrm{QMD}(L_i, L_j) &= 2 + 2 * lands(L_i, L_j) \\
\mathrm{QMD}(L_i..Li+1, Lj) &= 1 + 2 * lands(L_i, L_j) \\
\mathrm{QMD}(L_i, L_j..L_j+1) &= 1 + 2 * lands(L_i, Lj+1) \\
\mathrm{QMD}(L_i..L_i+1, L_j..L_j+1) &= 2 * lands(L_i, L_j+1)
\end{aligned}$$

pri čemer je $lands(L_i, L_j)$ enako številu odlikovanih vrednosti med odlikovanima vrednostima L_i in L_j. S QMD_i označimo QMD med trenutno vrednostjo spremenljivke S_i in vrednostjo te spemenljivke v ciljnem stanju. Tako

je TotalQMD definirana kot vsota QMD_i vseh simuliranih spremenljivk sistema. Našo hevristično funkcijo bomo imenovali TotalQMDpD, ker temelji na funkciji QMDpD (angl. QMD plus Direction). QMDpD z razliko od QMD upošteva tudi smer spreminjanja spremenljivke v trenutnem stanju. QMDpD je definirana z enačbo:

$$\mathrm{QMDpD}(S_1, S_2) = \begin{cases} 0 & ;\mathrm{QMD}(S_1, S_2) = 0 \text{ or } Dir1 = std \\ \mathrm{QMD}(S_1, S_2) + 1 & ; Dir1 = Dir2 \\ \mathrm{QMD}(S_1, S_2) - 1 & ; Dir1 \neq Dir2 \end{cases}$$

pri čemer je $Dir2 = relative\_qmag(S1, S2)$ potrebna smer spremembe, da pridemo iz stanja S_1 v stanje S_2. Tako na podlagi QMDpD definiramo TotalQMDpD kot vsoto QMDpD_i vseh simuliranih spremenljivk. Koda 2.1 prikazuje implemntacijo TotalQMDpD v programskem jeziku prolog.

Koda 2.1: Implementacija TotalQMDpD

```
cost_total_qmdpd([], [], 0).
cost_total_qmdpd([Var|Vars], [VarNew|VarsNew],
   TotalCost) :-
        cost_qmdpd(Var, VarNew, QMDpD),
        cost_total_qmdpd(Vars, VarsNew, Cost),
        TotalCost is Cost+QMDpD, !.

cost_qmdpd(_Dom:_Var/_Dir, V, 0) :-
        var(V), !.
cost_qmdpd(Dom:V/Dir, Dom:Vn/_, C) :- !,
        landmarks(Dom, Lands),
        cost_var(V, Vn, Lands, Cv),
        cost_pd(Dir, Cv, Cd), !,
        C is abs(Cv)+Cd, !.
cost_qmdpd(Dom:_V, Dom:_V1, 0) :- !.

cost_pd(Dir, SQMD, PD) :-
        SQMD == 0 -> PD is 0;
  Dir == std -> PD is 0;
  SQMD > 0 -> (Dir == inc -> PD is -1; PD is 1);
  SQMD < 0 -> (Dir == dec -> PD is -1; PD is 1);
  PD is 0,!.
```

```
%negtive when L2 is before L1
cost_qmd(_Var, VarN, _Lands, 0):-
  var(VarN),!.
cost_qmd(L1, L2, _Lands, 0):-
  L1 == L2,!.

cost_qmd(L1.._L2, VarN, Lands, C):-
  cost_qmd(L1, VarN, Lands, C1),
  C is C1 - 1,!.

cost_qmd(Var, L1.._L2, Lands, C):-
  cost_qmd(Var, L1, Lands, C1),
  C is C1 + 1,!.

cost_qmd(L1, L2, [L1|Ls], C):-
  conc(L, [L2|_], Ls),
  length(L, C1),
  C is (C1+1)*2,!.

cost_qmd(L1, L2, [L2|Ls], C):-
  conc(L, [L1|_], Ls),
  length(L, C1),
  C is (-C1-1)*2,!.

cost_qmd(L1, L2, [_|Ls], C):-
  cost_qmd(L1, L2, Ls, C1),
  C is C1 + 2.
```

2.5.1 Uspešnost TotalQMDpD

Primerjava hevrističnih funkcij je bila izvedena na primerih, opisanih v poglavju 4. Gre za domeno potiskanja valja in domeno potiskanja kocke. Konkretnje, v primeru valja je bil izmerjen čas, ki ga planer potrebuje, da z izbrano hevristično funkcijo sestavi plan, ki ga prikazuje tabela 4.3. V primeru potiskanja kocke pa je bil izmerjen čas, ki ga planer potrebuje za izgradnjo dveh planov, prikazanih v tabelah 4.6 in 4.7. Pravzaprav so bili izmerjeni časi, ki jih planer potrebuje za izgradnjo planov z istim začetnim in končnim stanjem kot omenjeni plani.

Definicije hevrističnih funkcij, ki smo jih primerjali, prikazuje tabela 2.3.

hevristka	**definicija**
brez	$h(x) = 0$
TotalQMDpD	$\sum_{i=0}^{n} \mathrm{QMDpD}_i$
TotalQMD	$\sum_{i=0}^{n} \mathrm{QMD}_i$
MaxQMDpD	$\max_{i=0}^{n} \mathrm{QMDpD}_i$
MaxQMD	$\max_{i=0}^{n} \mathrm{QMD}_i$

Tabela 2.2: Definicije testiranih hevristik

Pri definicijah v tabeli je privzeto, da so $\{S_i \mid 0 \leq i \leq n\}$ simulirane spremenljivke stanja. TotalQMDpD smo primerjali z že omenjeno podobno hevristiko TotalQMD. V primerjavo smo dodali tudi MaxQMD, ki je bila prav tako predlagana v [14]. MaxQMD temelji na QMD in je definirana kot maksimalni QMD_i vseh simuliranih spremenljivk. MaxQMDpD temelji na QMDpD in je definiran kot maksimalni QMDpD_i vseh simuliranih spremenljivk. Za primerjavo pa smo izvedli planiranje tudi brez hevristike, v tem primeru je $h(x) = 0$.

hevristika	**plan valj**	**kocka plan1**	**kocka plan2**
brez	0,14s	49,10s	32,62s
TotalQMDpD	**0,01s**	**0,30**	**1,65**
TotalQMD	0,03s	9,89s	2,21s
MaxQMDpD	0,03s	2,18s	2,59s
MaxQMD	0,06s	9,90s	2,37s

Tabela 2.3: Izmerjeni časi gradnje planov

Izmerjeni časi se lahko razlikujejo glede na strojno opremo, na kateri teče planer, zato primerjamo le razmerje med časi iskanja planov. Pri preprostem primeru potiskanja valja so časi izredno krati, tako da razlike niso tako očitne, vendar so opazne. Večje pa so razlike med časi pri kompleksnejšem primeru potiskanja kocke. Pri iskanju prvega plana nam hevristika TotalQMD iskanje plana pohitri za 5×. Enako pohitritev beležimo z uporabo hevristike MaxQMD. Naša izbrana hevristika TotalQMDpD, ki je po definiciji zelo podobna TotalQMD, pa nam iskanje prvega plana pohitri kar za 163×. Iskanje drugega plana brez hevristike je krajše od iskanja prvega plana. Tudi pohitritev hevrtistike TotalQMD je za iskanje drugega plana bolj izrazita in to za 12×. TotalQMDpD pa iskanje plana 2 pohitri za 20×, kar je še vedno bistveno

več kot TotalQMD.

2.6 Dodatna kvalitativna omejitev

Modeli v obliki kvalitativnega drevesa, ki jih sestavi algoritem QUIN [20], navadno vsebujejo omejitev QDE monotone odvisnosti od več spremenljivk [12, 18], kot je $Z = M^{+,-}(X, Y)$ ali $A = M^{+,-,+}(X, Y, Z)$.

Definicijo monotone odvisnosti v matematičnem smislu lahko pokažemo na primeru. Naj bo $Z = M^{+,-}(X, Y)$. Potem za vse vrednosti x_0, x_1, y_0, y_1 velja: če $x_0 < x_1$ in $y_0 < y_1$, potem $Z(x_0, y_0) < Z(x_1, y_1)$.

Omejitev monotone odvisnosti od več spremenljivk ni ena od kvalitativnih omejitev predstavljenih v originalnem algoritmu QSIM. Tako tudi implementacija v programskem jeziku prolog [3] ne vsebuje te omejitve. Ena od razširitev algoritma je implementacija predikata *mono*/3.

Predikat *mono*/3 ima tri argumente:

- operativno območje, v kateri je omejitev veljavna;
- seznam neodvisnih spremenljivk s predpisanim določilom + ali −;
- odvisna spremenljivka.

Ime operativnega območja potrebujemo, da določimo, katere korespondenčne vrednosti lahko uporabimo. Vsako operativno območje ima svoje korespondenčne vrednosti. Te definiramo s predikatom *correspond*/3, ki ima prav tako tri argumente:

- operativno območje, kateremu korespondenčna vrednost pripada;
- seznam kvalitativnih veličin neodvisnih spremenljivk;
- kvalitativna veličina odvisne spremenljivke.

Spodnja koda prikazuje primer uporabe *mono*/3 in *correspond*/3. V tem primeru imamo le eno operativno območje imenovano *test*.

```
landmarks(x, [minf, xn, zero, inf]).
landmarks(y, [minf, zero, yp, inf]).
landmarks(z, [minf, zero, zg, inf]).

legalstate([X,Y,Z]):-
    mono(test, [-X,+Y],Z).
```

```
correspond(test, [x:zero, y:zero], z:zero).
correspond(test, [x:xn, y:yp], z:zg).
```

2.6.1 Implementacija

```
mono(Region, FVars, Dom:QM/Dir) :-
    qmag(Dom:QM),
    maplist(extract_dom_val, FVars, Corr), % nastavek
        za correspond
    \+ (
      correspond(Region, Corr, Dom:V),
      relative_qmag(Dom:QM,Dom:V, Sign1),
      relative_qmag_list(FVars, Corr, Sign2),
      Sign1 \== Sign2
    ),!,
    mono_dir(FVars, Dir).

mono_dir(Vars, Dir1) :-
  mono_dir_na(Vars, Dir2),!,
  Dir1 = Dir2.
mono_dir(_, Dir) :-
  qdir(Dir,_),!.

mono_dir_na([],std).
mono_dir_na([+(_Dom:_QM/Dirv)| Dirs], Dir):-
  mono_dir(Dirs, Dir1),
  merge_dir(Dirv, Dir1, Dir).
mono_dir_na([-(_Dom:_QM/Dirv)| Dirs], Dir):-
  mono_dir(Dirs, Dir1),
  opdir(Dirv, Diro),
  merge_dir(Diro, Dir1, Dir).

merge_dir(Dir, Dir, Dir):-!.
merge_dir(Dir, std, Dir):-!.
merge_dir(std, Dir, Dir):-!.

relative_qmag_list([],[], zero).
```

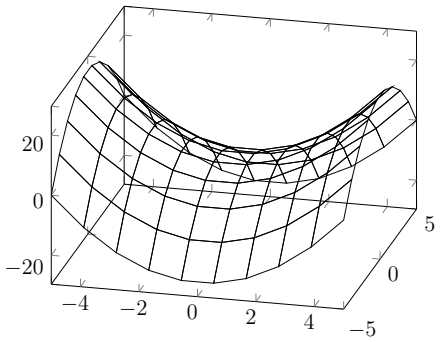
```
relative_qmag_list([+(Dom:QM/_Dir)|Vs1], [V2|Vs2],
   Sign):-
  relative_qmag_list(Vs1,Vs2, Sign1),
  relative_qmag(Dom:QM,V2, Sign2),
  merge_sign(Sign1, Sign2, Sign).
relative_qmag_list([-(Dom:QM/_Dir)|Vs1], [V2|Vs2],
   Sign):-
  relative_qmag_list(Vs1,Vs2, Sign1),
  relative_qmag(Dom:QM, V2, Sign2),
  opsign(Sign2, Sign3),
  merge_sign(Sign1, Sign3, Sign).

merge_sign(Sign, Sign, Sign):-!.
merge_sign(Sign, zero, Sign):-!.
merge_sign(zero, Sign, Sign):-!.

extract_dom_val(+(Dom:_QM/_Dir), Dom:_).
extract_dom_val(-(Dom:_QM/_Dir), Dom:_).
```

2.6.2 Testiranje

Predikat je bil testiran na funkciji $f(x, y) = X^2 - Y^2$, ki je bila uporabljena za testiranje algoritma QUIN [20]. Gre za tako imenovano funkcijo sedla, ki jo prikazuje slika 2.1. Testiranje je potekalo na kvalitativnem drevesu, ki je



Slika 2.1: Funkcija $f(x, y) = X^2 - Y^2$

definirano kot ciljno drevo. Uporabljen je bil za primerjavo rezultatov učenja v [20]. Model v programskem jeziku prolog prikazuje koda spodaj.

```
legalstate([X,Y,Z]):-
    region([X,Y,Z], R),
    (
     R = aa -> mono(R, [-X,+Y], Z);
     R = ab -> mono(R, [-X,-Y], Z);
     R = ba -> mono(R, [+X,+Y], Z);
     R = bb -> mono(R, [+X,-Y], Z);
     fail
    ).

region([X,Y,_], R) :-
    X = Dx:QMx/_Dirx,
    Y = Dy:QMy/_Diry,
    relative_qmag(Dx:QMx, Dx:zero, Sx),
    relative_qmag(Dy:QMy, Dy:zero, Sy),
    tag(Sx, Sy, R).

tag(A,B, T) :-
    tag(A, T1),
    tag(B, T2),
    atom_concat(T1,T2,T).
tag(neg, 'a').
tag(pos, 'b').
tag(zero, 'b').
```

Za nekaj naključnih numeričnih vrednosti X in Y je bila izračunana vrednost Z. Za vsako numerično vrednost so bile dodane odlikovane vrednosti v ustrezno domeno. Za vsak trojček je bil dodan predikat *correspond*/3. Nadalje so bili naključno generirani numerični primeri. Primeri so bili spremenjeni v kvalitativne in preverjeni z modelom. Preverjene so bile ocene kvalitativne veličine ter smer spreminjanja. Na ta način je bilo ugotovljeno, da implementacija omejitve *mono*/3 predstavlja skupino kvalitativnih omejitev QDE, ki jih algoritem QUIN uporablja za gradnjo kvalitativnih dreves. To so kvalitativne omejitve monotone odvisnosti od več spremenljivk.

Poglavje 3

Proces vodenja robota

3.1 Komuniciranje z numeričnim sistemom

Izvajanje plana temelji na interakciji z zunanjim sistemom. Tako že v fazi planiranja pridobimo začetno stanje sistema s predikatom *state*/1. Ta predikat je namreč eden od predikatov, ki komunicirajo z vmesnikom numeričnega sistema. Tabela 3.1 prikazuje predikate, ki komunicirajo z vmesnikom. Tako je

predikat	**opis**
state/1	vrne kvalitativno stanje
state_fresh/1	blokira, dokler ni na voljo novo kvalitativno stanje
set_wheels/1	spremeni navor na kolesih
set_angle/1	zapelje robota na stranico predmeta
set_angle/2	zapelje robota na del stranice predmeta

Tabela 3.1: Predikati, s katerimi komuniciramo z vmesnikom

state/1 v uporabi v začetni fazi planiranja, ko smo že prejeli stanje, v katerem je trenutno sistem, vendar ga potrebujemo ponovno, da sestavimo plan. State *fresh*/1 pa je v uporabi pri samem izvajanju, ko čakamo na novo stanje.

3.2 Izvajalna zanka

Proces vodenja robota je celoten proces, ki z akcijami vodi robota, da doseže nek cilj. Plan, generiran s planerjem opisanim v poglavju 2, je le možna sosledica stanj, skozi katere naj bi šel sistem v primeru točno določenega zaporedja

proženja akcij. To pomeni, da med izvajanjem plana lahko pride do določenih odstopanj od plana in je morda potrebno ponovno planiranje. Tako bi lahko proces potiskanja predmeta v grobem opisali kot zanko, v kateri poizkušamo ustvariti plan in ga pričnemo izvajati. Ta proces se lahko zaključi neuspešno, če na podlagi modela iz trenutnega stanja ni mogoče zgraditi plana. Če pa plan lahko zgradimo, lahko pričnemo s samim izvajanjem slednjega. Predikat *plan_push*/0 prikazuje izvajanje plana.

```
plan_push :-
    repeat,
    interface_stop,
    plan_bestfirst(Plan),
    (Plan == [],!,fail
    ;
      execute(Plan),
      interface_stop,!
    ).
```

S predikatom *interface_stop*/0 pred vsakim novim poizkusom ustavimo vse aktivnosti, ker se planiranje izvede v trenutku in bi bilo aktualno že neko novo začetno stanje. V tem primeru bi bilo potrebno planiranje ponoviti. Ustavljanje nam ne zagotavlja, da se taki primeri ne bi dogajali, vendar zmanjša njihovo pojavnost.

3.3 Izvajanje plana

Ciljno stanje definiramo pred samim procesom vodenja robota. Definiran je kot predikat *goal*/1. Preberemo trenutno stanje sistema in sestavimo plan. Generirani plan je možna sosledica stanj v primeru točno določenega zaporedja proženja akcij. Izvajanje je sprehod skozi seznam stanj $[..., S_i, S_{i+1}, ...]$, pri čemer je S_i stanje, ki je bilo ravno doseženo, in S_{i+1} stanje, ki pričakujemo, da bo dosežno. Da bo stanje S_{i+1} doseženo, obstajajo naslednje možnosti:

- sprememba spremenljivk sistema je posledica akcije; v tem primeru izvedemo kodo akcije in pričnemo s čakanjem na učinek akcije;
- iz stanja S_i je možno z legalnim prehodom QSIM preiti v stanje S_{i+1}; v tem primeru pričnemo s čakanjem na novo stanje;
- za prehod ni nobene akcije, stanje je stabilno in QSIM ne ugotovi prehoda; v tem primeru prekinemo izvajanje.

Novo stanje se lahko ujema s pričakovanim stanjem; v tem primeru se pomaknemo po planu naprej. Če se novo stanje ne ujema, pa prekinemo izvajanje plana in se vrnemo v izvajalno zanko.

Kot lahko vidimo, je algoritem QSIM uporabljen tudi med samim izvajanjem plana. Predikat *execute*/2 definira izvajanje plana v programskem jeziku prolog.

```
execute([Goal|Rest]):-
  goal(Goal),
  goal(G),
  execute(Rest, G),!.

execute([], State):-
  state(State1), % pogledamo stanje sistema
  State1 = State,!.

execute([Current|Rest], State) :-
   % izvedemo plan do trenutnega stanja
   execute(Rest, Current),!,
   (
      action_for_effect(Current, State, Action),
      Action % izvedemo akcijo
      ;
      legal_trans(Current, State) % QSIM prehod
   ),!,
   repeat,
   state_fresh(NewState), % cakaj na sveze stanje
   ( NewState = State,!  % novo stanje se ujema z
      pricakovanim
      ;
        % ne obstaja QSIM prehod
        \+ legal_trans(NewState, State),
        !,fail % zakljuci
      ;
        fail % ponovi
  ).
```

Poglavje 4

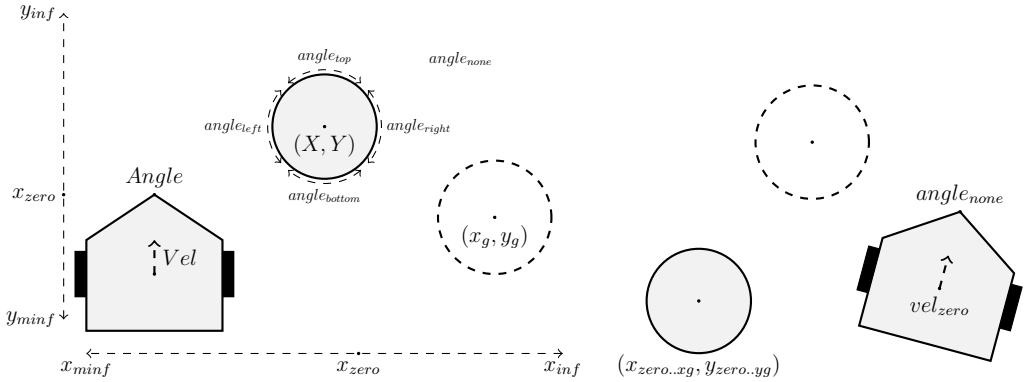
Primeri

Testiranje planerja in izvajalca plana je potekalo na dveh različnih primerih, in sicer na primeru potiskanja pokončenga valja in na primeru potiskanja kocke. Oba primera sta bila testirana na simulatorju potiskanja predmetov, ki ga je razvil Domen Šoberl iz Laboratorija za umetno inteligenco FRI za lastno uporabo in temelji na fizikalnem pogonu Box2D [1]. Simulator nam omogoča, da beremo pozicije in rotacijo simuliranih predmetov. Eden od simuliranih predmetov je tudi mobilen robot z dvema kolesoma (Slika 4.1), kateremu lahko spreminjamo navor levega in/ali desnega kolesa. Ker planer in izvajalec plana pričakujeta kvalitativno stanje, je med planerjem in simulatorjem vgrajen vmesnik. Ta bere simulirano stanje in pošilja simulatorju zahteve po stanjih objektov. Vmesnik, na podlagi dobljenih pozicij, izračuna hitrost gibanja objektov. Na podlagi dane konfiguracije, spremeni vmesnik numerično stanje v kvalitativno. Vmesnik je zadolžen za implementacijo akcij, ki so na voljo planerju. Akcije so lahko preproste, kot na primer, akcija za spreminjanje navora na kolesih. Ta akcija spremeni navor na kolesih tako, da je na obeh kolesih enak delež zahtevanega navora. Ko pa se robot dotika stranice predmeta, takrat vmesnik uravnava razmerje med navoroma levega in desnega kolesa s pomočjo regulatorja PID [11]. Robot na ta način ves čas pritiska pravokotno na stranico predmeta. Akcije so pa lahko tudi kompleksnejše, na primer akcija, ki robota pripelje točno na določeno stranico predmeta ali celo na del stranice predmeta.

4.1 Primer potiskanja okroglega predmeta

Zaradi enostavnosti in boljšega pregleda nad delovanjem planerja najprej predstavimo primer potiskanja pokončnega valja. Ne glede na rotacijo valja je tlorisen pogled vedno enak, tako nam rotacije pri potiskanju ni potrebno upoštevati.

Robot lahko valj potiska pod katerimkoli kotom. Kot potiskanja definiramo kot kot med Y osjo in pravokotnico na tangento v točki dotikanja odbijača robota in predmeta, vendar zaradi enostavnosti kot potiska diskretiziramo le na štiri skupine kotov: zgoraj, spodaj, levo in desno.



(a) Spremenljivke sistema (b) Začetno stanje robota

Slika 4.1: Primer potiskanja valja

Kvalitativno stanje sistema opišemo s štirimi kvalitativnimi spremenljivkami. Slika 4.1a prikazuje kvalitativne spremenljivke in njihove domene. Spremenljivka X je pozicija valja na x osi, ima odlikovano vrednost (angl. landmark) xg, ki leži med $minf$ in inf in predstavlja x koordinato označbe cilja. Podobno spremenljivka Y opisuje pozicijo valja, ima odlikovano vrednost yg, ki leži med $minf$ in inf in predstavlja y koordinato označbe cilja. Spremenljivka Vel opisuje hitrost robota. Značilna točka $zero$ leži med $minf$ in inf in predstavlja mirovanje robota. Spremenljivka $Angle$ opisuje kot potiskanja predmeta. Odlikovane vrednosti domene so top, $bottom$, $right$ in $left$, ki predstavljajo diskretizirane točke dotikanja predmeta in stanje $none$, ki predstavlja stanje, v katerem se robot ne dotika predmeta. Spremenljivki Vel in $Angle$ sta kontrolirani spremenljivki, oziroma njuno stanje spreminjajo akcije. $Angle$ je primer nezvezne spremenljivke. Domene spremenljivk prikazuje slika 4.2.

4.1.1 Model valja

Na podlagi kota potiskanja se tako razdeli omejitve sistema v pet operativnih območij: območje potiskanja zgoraj, spodaj, levo, desno in območje, kjer se robot ne dotika predmeta. Če robot potiska predmet z leve strani, se pozicija robota po x koordinati povečuje, oziroma je odvod pozicije v x smeri enak hitrosti robota. Če robot potiska predmet z desne strani, se pozicija robota

$$X \in [minf, zero, xg, inf] \tag{4.1}$$
$$Y \in [minf, zero, yg, inf] \tag{4.2}$$
$$Vel \in [minf, zero, inf] \tag{4.3}$$
$$Angle \in [top, bottom, left, right, none] \tag{4.4}$$

Slika 4.2: Spremenljivke sistema potiskanja valja in njihove domene

po x koordinati zmanjšuje, oziroma je odvod pozicije v x smeri enak minus hitrosti robota. Če robot potiska predmet z zgornje strani, se pozicija robota po y koordinati zmanjšuje, oziroma je odvod pozicije v y smeri enak hitrosti robota. Če robot potiska predmet s spodnje strani, se pozicija robota po y koordinati povečuje, oziroma je odvod pozicije v y smeri enak minus hitrosti robota. Če se robot ne dotika predmeta, se pozicija predmeta ne spremeni. Na podlagi hitrosti gibanja robota imamo območje, kjer se robot premika vzvratno. Tudi v tem primeru predmet miruje. To območje se prekriva z območji, ki določujejo kot potiskanja. V tabeli 4.1 je prikazana delitev na območja. V prvem stolpcu je oznaka območja. Drugi stolpec prikazuje pogoj, ki je resničen znotraj označenega območja. V tretjem stolpcu pa so naštete kvalitativne omejitve. Omejitev $stdy(X)$ zahteva, da spremenljivka X miruje. V dodatku B se nahaja definicija modela valja v programskem jeziku prolog.

op. območja	**pogoj**	**omejitve**
top	$Angle = top$	$mderiv(Y, Vel), stdy(X)$
$bottom$	$Angle = bottom$	$deriv(Y, Vel), stdy(X)$
$left$	$Angle = left$	$deriv(X, Vel), stdy(Y)$
$right$	$Angle = right$	$mderiv(X, Vel), stdy(Y)$
$none$	$Angle = none$	$stdy(X), stdy(Y)$
$backing$	$Vel = minf..zero$	$stdy(X), stdy(Y)$

Tabela 4.1: Razdelitev na operativna območja in pripadajoče omejitve (primer valja)

Kot že omenjeno kontrolirani spremenljivki $Angle$ in Vel spreminjamo z akcijami, ki so opisane v podpoglavju 2.3. Imamo akcijo za spreminjanje hitrosti robota in izbiranje kota potiska. Akcija za spreminjanje hitrosti robota ima predpogoj, da se robot dotika predmeta, torej, da je $Angle \neq none$. Hitrost

akcija	predpogoj	učinek
$set\_angle(EffectAngle)$ $EffectAngle \in [top, bottom, left, right]$	$Angle = none$ $Vel = zero$	$Angle = EffectAngle$
$wheels\_power(zero)$	$Angle = none$	$Vel = zero$
$wheels\_power(minf..zero)$	$Angle \neq none$	$Angle = none$ $Vel = minf..zero$
$wheels\_power(EffectVel)$ $EffectVel \in [zero, zero..inf]$	$Angle \neq none$	$Vel = EffectVel$

Tabela 4.2: Akcije za primer valja

lahko spremeni v $minf...zero$ ali $zero$ ali $zero...inf$. Imamo kompleksnejšo akcijo za izbiranje kota, ki ima predpogoj, da je hitrost robota enaka nič, torej, da je $Vel = zero$. Ta akcija sporoči kontrolerju robota, da spremeni kot potiskanja iz kateregakoli stanja v katerokoli drugo stanje. Pri obravnavi primera ni pomembno, na kakšen način kontroler izvrši akcijo, predpostavimo le, da izvrševanje akcije vedno uspe. Tabela 4.2 prikazuje omenjene akcije, ki so na voljo.

4.1.2 Plan za potiskanje valja

Začetno stanje, ki ga prikazuje slika 4.1b. Robot je postavljen tako, da se ne dotika predmeta. Valj je postavljen tako, da se njegova pozicija razlikuje od ciljne pozicije po obeh koordinatah. Na poligonu ni nobenih drugih ovir.

Enačba 4.3 prikazuje kvalitativno vrednost spremenljivk sistema v začetnem stanju. Za ciljno pozicijo predmeta določimo ciljno pozicijo označbe $X = xg$ in $Y = yg$. Ker hitrost robota in kot potiskanja pri definiranju cilja nista pomembna, ju pri zahtevi cilja planerju ne podamo. Planer tako poišče plan, ki ga prikazuje tabela 4.3. V prvem stolpcu je zaporedna številka planiranega stanja, v drugem stolpcu je planirano stanje in v tretjem stolpcu je akcija, ki je pripomogla do planiranega stanja. Akcija se je lahko prožila v prejšnjem stanju ali tik ob prehodu iz prejšnjega v neko drugo stanje. Primer akcije, ki je bila prožena v prejšnjem stanju, je akcija pri stanju številka dva v tabeli, ki je bila prožena v stanju številka ena v tabeli. Primer za akcijo, ki je planirana, da se proži tik ob prehodu stanj, je akcija pri stanju številka štiri v tabeli, ki je

bila prožena ob prehodu iz stanja številka tri v tabeli. Izvajalec plana razbere, za kateri primer akcije gre, na podlagi spremenljivk, ki niso kontrolirane. Če se je kvalitativna vrednost teh spremenljivk spremenila, pomeni, da je akcija prožena ob prehodu. V nasprotnem primeru pa je prožena v prejšnjem stanju.

$$X = x : zero..xg/std \tag{4.5}$$
$$Y = y : zero..yg/std \tag{4.6}$$
$$Vel = vel : zero \tag{4.7}$$
$$Angle = angle : none \tag{4.8}$$

Slika 4.3: Začetno stanje sistema

Prvo stanje v tabeli 4.3 prikazuje začetno stanje robota. Akcija *set_ angle(bottom)* postavi robota v potiskanje od spodaj in iz začetnega kvalitativnega stanja postavi robota v stanje številka dva. Nato z akcijo *wheels_ power(zero..inf)* poveča hitrost robota, tako da robot prične potiskati valj proti ciljni koordinati $Y = yg$. Stanje potiskanja ponazarja stanje številka tri. Takoj ko robot doseže ciljno koordinato, robot izvede akcijo *wheels_ power(minf..zero)* ki ga postavi v vzvratno vožnjo. Pri tem se gibanje valja ustavi na ciljni y koordinati in robot izgubi stik z valjem. To stanje ponazarja stanje številka štiri. Ko se robot valja več ne dotika, se robot z akcijo *wheels_ power(zero)* zaustavi, da izpolni pogoje za spreminjanje kota potiskanja. Nato se robot z akcijo *set_ angle(left)* postavi na levo stran valja. To stanje ponazarja stanje številka šest. Robot z akcijo *wheels_ power(zero..inf)* prične s potiskanjem. Ko pa je ciljna koordinata poravnana, se robot z akcijo *wheels_ power(zero..inf)* ponovno postavi v vzvratno vožnjo, s čimer doseže, da se valj ustavi na ciljnih koordinatah, kar ustreza našemu cilju.

Plan, ki je bil generiran, je ena od možnih zaporedij stanj, ki se je v našem primeru s proženjem akcij v celoti uresničil. Implementacija akcije, ki pripelje robota pod določen kot na objekt je taka, da pripelje robota v sredino tega območja. Tako se pozicija predmeta spreminja le po eni koordinati. Obnašanje, ki ga je napovedal planer in implementacija akcije se le naključno ujemata. Kar se tiče same definicije kvalitativne akcije, bi lahko robota pripeljala kamorkoli znotraj zahtevanega območja. Tako bi se lahko v primeru potiskanja, predmet premikal po več koordinatah. To stanje ne bi bilo planirano, zato bi se planiranje ponovilo z novim začetnim stanjem, kjer se robot že premika po

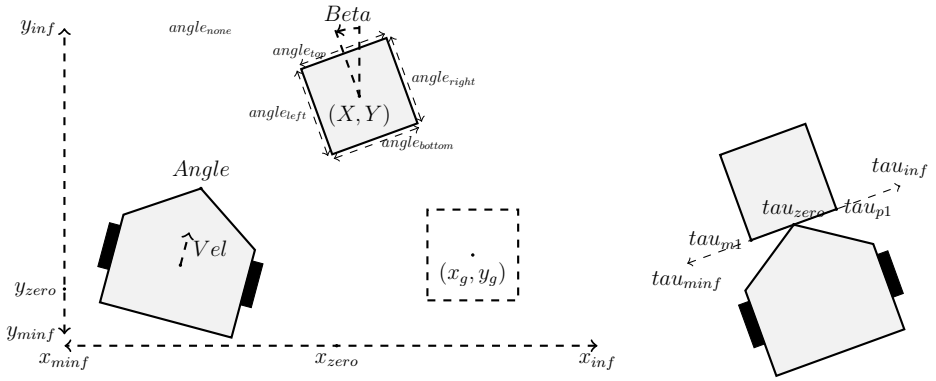
dveh koordinatah.

	akcija **stanje** $[X, Y, Vel, Angle]$
1	$[zero..xg/std, zero..yg/std, zero, none]$ $set\_angle(bottom)$
2	$[zero..xg/std, zero..yg/std, zero, bottom]$ $wheels\_power(zero..inf)$
n 3	$[zero..xg/std, zero..yg/inc, zero..inf, bottom]$ $wheels\_power(minf..zero)$
4	$[zero..xg/std, yg/std, zero..zero, none]$ $wheels\_power(zero)$
5	$[zero..xg/std, yg/std, zero, none]$ $set\_angle(left)$
6	$[zero..xg/std, yg/std, zero, left]$ $wheels\_power(zero..inf)$
7	$[zero..xg/inc, yg/std, zero..inf, left]$ $wheels\_power(minf..zero)$
8	$[xg/std, yg/std, minf..zero, none]$

Tabela 4.3: Izdelani plan za potiskanje valja

4.2 Primer potiskanja kocke

Zaradi rotacije je primer potiskanja kocke (Slika 4.4) kompleksnejši od primera potiskanja valja. Rotacija je odklon vzdolžne osi kocke od koordinatne osi Y. Medtem ko smo imeli v primeru potiskanja valja kot potiskanja, imamo v primeru potiskanja kocke stranico na katero pritiska robot. Pomembno je tudi upoštevati, kje na stranici se robot dotika predmeta, saj to vpliva na samo rotacijo. Kota med vzdolžno osjo robota in robom stranice predmeta ne upoštevamo, ker smo v kontroler robota vgradili mehanizem, ki robota drži ves čas pod pravim kotom na stranico.



(a) Spremenljivke sistema (b) Spremenljivka Tau

Slika 4.4: Primer potiskanja kocke

Kvalitativno stanje sistema opišemo s šestimi kvalitativnimi spremenljivkami. Slika 4.5 prikazuje kvalitativne spremenljivke in njihove domene. Spremenljivke X, Y in Vel pri primeru potiskanja kocke so enake kot že opisane pri primeru potiskanja valja. Prav tako imamo spremenljivko $Angle$, ki predstavlja smer potiskanja predmeta in zaseda enake odlikovane vrednosti kot pri primeru potiskanja valja, vendar imajo odlikovane vrednosti *top*, *bottom*, $left$ in *right* v primeru potiskanja kocke drugačno interpretacijo, saj te določajo stranico predmeta na katero robot pritiska. Odlikovana vrednost *none* pa ima enak pomen kot pri primeru potiskanja valja, oziroma, da se robot ne dotika predmeta. Spremenljivka $Beta$ nam pove, koliko je kocka rotirana od pokončnega položaja. Odlikovane vrednosti spremenljivke $Beta$ prikazuje tabela 4.5. Slika 4.4.a prikazuje spremenljivke sistema. Pri primeru potiskanja kocke imamo še dodatno spremenljivko Tau, ki nam pove, na katerem delu stranice predmeta se robot dotika. Slika 4.4.b prikazuje spremenljivko Tau. Odliko-

vana vrednost $zero$ nam pove, da se robot dotika na sredini stranice predmeta. Vrednosti $m1$ in $p1$ pa predstavljata skrajna robova stranice predmeta. Kadar se robot ne dotika predmeta, je Tau nedefiniran. V tem primeru mu dodelimo vrednost $zero$.

$$X \in [minf, zero, xg, inf] \tag{4.9}$$
$$Y \in [minf, zero, yg, inf] \tag{4.10}$$
$$Beta \in [minf, mPi, m3Pi4, mPi2, mPi4, zero, pPi4, pPi2, p3Pi4, pPi, inf] \tag{4.11}$$
$$Vel \in [minf, zero, inf] \tag{4.12}$$
$$Tau \in [minf, m1, zero, p1, inf] \tag{4.13}$$
$$Angle \in [top, bottom, left, right, none] \tag{4.14}$$

Slika 4.5: Spremenljivke sistema potiskanja kocke in njihove domene

Operativno območje all nam definira omejitve, katere je potrebno vedno upoštevati, v smislu, da spremenljivka $Beta$ ne more zavzeti manjše vrednosti od mPi in da ne more nikoli doseči vrednosti pPi. Spremenljika Tau lahko zavzema vrednosti le med $m1$ in $p1$. Operativno območje $stdy$ nam definira območje, kjer kocka miruje. To je takrat, ko se robot ne premika naprej in/ali takrat, ko se robot ne dotika kocke. Območje, kjer robot potiska kocko, se je dotika in se premika naprej, imenujemo $pushing$. V tem operativnem območju velja, da je vrednost spremenljivke Tau enaka odvodu spremenljivke $Beta$. Skupina operativnih območij A/B definira spreminjanje koordinat kocke. Območja so razdeljena na podlagi absolutnega kota potiskanja, ki ga označimo z $e\beta$.

$$Region(e\beta) = \begin{cases} s/p & ; e\beta = mPi \\ m/p & ; mPi < e\beta < mPi2 \\ m/s & ; e\beta = mPi2 \\ m/m & ; mPi2 < e\beta < zero \\ s/m & ; e\beta = zero \\ p/m & ; zero < e\beta < pPi2 \\ p/s & ; e\beta = pPi2 \\ p/p & ; pPi2 < e\beta < pPi \end{cases} \tag{4.15}$$

4.2.1 Model potiskanja kocke

Tabela 4.4 prikazuje operativna območja, pogoj, pri katerem so veljavna, in pripadajoče omejitve. V dodatku C pa je definicija modela v programskem jeziku prolog.

op. območja	**pogoj**	**omejitve**
A/B $A, B \in [m, s, p]$	$e\beta$ $=$ $qsum(Beta, \delta(Angle))$ A/B $=$ $Region(e\beta)$	$\begin{cases} mderiv(X, Vel) & ; A = m \\ stdy(X) & ; A = s \\ deriv(X, Vel) & ; A = p \end{cases}$ $\begin{cases} mderiv(Y, Vel) & ; B = m \\ stdy(Y) & ; B = s \\ deriv(Y, Vel) & ; B = p \end{cases}$
$pushing$	$Angle \neq none$ $Vel = zero..minf$	$deriv(Beta, Tau)$
any		$mPi \leq Beta < pPi$ $m1 \leq Tau \leq p1$
$stdy$	$Angle = none$ **or** $Vel \neq zero..minf$	$stdy(X), stdy(Y), stdy(Beta)$

Tabela 4.4: Razdelitev na operativna območja in pripadajoče omejitve

Enačba 4.15 prikazuje katero območje je veljavno za določen kot. Kot $e\beta$ izračunamo kot kvalitativno vsoto spremenljivke $Beta$ in relativnega kota na stranico kocke, na katero robot trenutno potiska. Kot stranice $delta(Angle)$ prikazuje enačba 4.16 in je v primeru kocke splošno znan.

$$\delta(Angle) = \begin{cases} zero & ; Angle = top \\ pPi2 & ; Angle = left \\ pPi & ; Angle = bottom \\ mPi2 & ; Angle = right \end{cases} \tag{4.16}$$

Nabor akcij je v primeru potiskanja kocke zelo podoben kot v primeru potiskanja valja: akcija *wheels_power(zero)* z enakim predpogojem in enakim učinkom; prav tako akcjia *wheels_power(EffectVel)*. Akcija *wheels_power(minf..zero)* ima dodaten učinek, da se spremenljivka Tau spremeni v *zero*, ker se robot več ne dotika predmeta. Največja sprememba je dodaten parameter

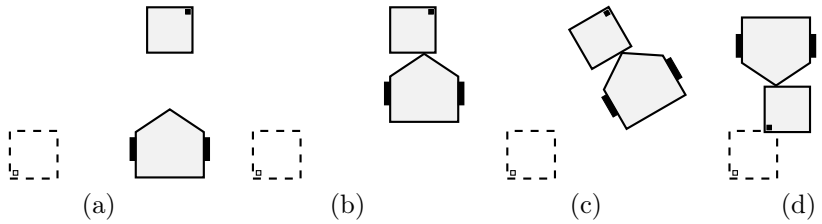
akcije *set_ angle(EffectAngle,EffectTau)*. Dodatni parameter akcije *EffectTau* omogoča izbiro lokacije na stranici *EffectAngle*. Predpogoj za akcijo pa ostaja nespremenjen. Definicijo akcij opisuje Tabela 4.5.

akcija	**predpogoj**	**učinek**
set_ angle(EffectAngle,EffectTau) $EffectAngle \in [top, bottom, left, right]$ $Tau \in [m1..zero, zero, zero..p1]$	$Angle = none$ $Vel = zero$	$Angle = EffectAngle$ $Tau = EffectTau$
wheels_ power(zero)	$Angle = none$	$Vel = zero$
wheels_ power(minf..zero)	$Angle \neq none$	$Angle = none$ $Vel = minf..zero$ $Tau = zero$
wheels_ power(EffectVel) $EffectVel \in [zero, zero..inf]$	$Angle \neq none$	$Vel = EffectVel$

Tabela 4.5: Akcije za primer kocke

4.2.2 Plan za potiskanje kocke

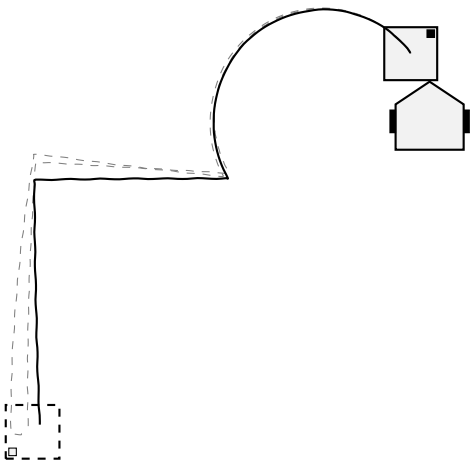
Ker je primer potiskanja kocke kompleksnejši, je bil prvi plan, ki ga je generiral planer preveč optimističen. Planer je moral ponoviti planiranje, da je kocko uspešno potisnil na ciljno pozicijo.



Slika 4.6: Izvajanje prvega plana potiskanja kocke

Začetno stanje prikazuje slika 4.6a. Stanje spremenljivk sistema prikazuje prva vrstica tabele 4.6. Z akcijo *set_ angle(top,zero..p1)* planer zapelje robota na zgornji rob kocke. Ker je kocko potrebno zarotirati, je planer izbral točko potiskanja proti robu, kjer je spremenljivka $Tau = zero..p1$. Slika 4.6b prikazuje položaj robota. Stanje kvalitativnih spremenljivk pa opisuje druga vrstica

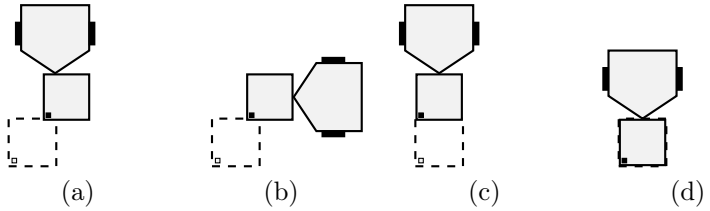
v tabeli 4.6. Z akcijo *wheels_power(zero..inf)* prične robot potiskati kocko in jo rotirati proti pravilni usmerjenosti, kar prikazuje slika 4.6c. Kocko v rotaciji opisujejo stanja med tretjo in deseto vrstico v tabeli 4.6.



Slika 4.7: Kvantitativna trajektorija kocke

Iz stanja desete vrstice v stanje enajste vrstice tabele 4.6 je planer napovedal, da bo kocka istočasno dosegla ciljni kot $Beta = zero$ in pravi vrednosti koordinat $X = xg$ in $Y = yg$. Koordinati se zmanjšujeta, v stanju desete vrstice v tabeli, vendar ne dovolj, da bi dosegli ciljni položaj. Potrebno je ponovno planiranje. Novo začetno stanje prikazuje slika 4.8a. Robot se dotika kocke na skrajnem robu. Vrednost spremenljivk opisuje prva vrstica v tabeli 4.7. Rotacija kocke je dosežena. Z akcijo *wheels_power(minf..zero)* planer robota zapelje vzvratno ter ga z akcijo *wheels_power(zero)* ustavi. Tako so zadoščeni pogoji za akcijo *set_angle(right,zero)*, ki robota zapelje na desno stranico kocke. Dodatna rotacija ni zaželjena. Planer zapelje robota na sredino stranice kocke, kjer je $Tau{+}zero$. Stanje prikazuje slika 4.8b in je opisano v četrti vrstici tabele 4.6.

Iz stanja štiri v tabeli 4.6 z akcijo *wheels_power(zero..inf)* robot prične potiskati kocko vzdolž koordinate X. Ko robot doseže vrednost koordinate $X = xg$, planer robotu z akcijo *wheels_power(minf..zero)* naroči, naj gre vzvratno in s tem robot preneha potiskati kocko. Planer robota ustavi in ga ponovno premakne na zgornjo stranico kocke, kar prikazuje slika 4.8c. Z akcijo *wheels_power(zero..inf)* planer robotu naroči, da potisne kocko po Y koordinati v končni položaj, ki ga opisuje enajsta vrstica v tabeli 4.6 in je prikazan na sliki 4.8d. Izvajanje obeh planov, vključno s planiranjem, je tra-



Slika 4.8: Izvajanje drugega plana potiskanja kocke

jalo 91 sekund. Kvantitativna trajektorija, po kateri se giblje kocka, v primeru opisanega potiskanja, je prikazana z odebeljeno krivuljo na sliki 4.7. Video celotnega opisanega izvajanja po planih 1 in 2 pa je dostopen na sledeči povezavi `https://youtu.be/ftC_IyMTDvc`.

Kot že omenjeno v primeru potiskanja valja, je plan, ki ga sestavi planer, le možno zaporedje stanj. V primeru potiskanja kocke je to še bolj razvidno, ker imamo dva generirana plana. Na sliki 4.7 so s črtkano krivujlo prikazani kavantitativni trajektoriji dveh drugih potiskanj. Prvi plan je za vsa izvajanja enak, saj imajo vsi primeri isto začetno stanje in isto končno stanje. Planer je v prvem planu napovedal, da bosta koordinati predmeta in njegova orientacija hkrati dosegli ciljno stanje. To se ni zgodilo v nobenem od prikazanih primerov. Planerjeva izbira takega obnašanja je razumljiva, saj je plan ob taki izbiri krajši. Tudi drugi plan je bil pri vseh primerih enak. Ob pravilnem proženju akcij, naj bi kocka dosegla ciljno pozicijo. V enem od primerov se je ena od koordinat tako oddaljila od ciljne koordinate, da je bilo potrebno ponovno planiranje, ki je popravilo položaj kocke po tej koordinati. Do tega je prišlo zaradi napak pri merjenju in netočnosti robota.

	akcija **stanje** $[X, Y, Beta, Vel, Tau, Angle]$
1	$[xg..inf/std, yg..inf/std, mPi/std, zero, zero, none]$
	set_ angle(top,zero..p1)
2	$[xg..inf/std, yg..inf/std, mPi/std, zero, zero..p1, top]$
	wheels_ power(zero..inf)
3	$[xg..inf/std, yg..inf/inc, mPi/inc, zero..inf, zero..p1, top]$
4	$[xg..inf/dec, yg..inf/inc, mPi..m3Pi4/inc, zero..inf, zero..p1, top]$
5	$[xg..inf/dec, yg..inf/inc, m3Pi4/inc, zero..inf, zero..p1, top]$
6	$[xg..inf/dec, yg..inf/inc, m3Pi4..mPi2/inc, zero..inf, zero..p1, top]$
7	$[xg..inf/dec, yg..inf/std, mPi2/inc, zero..inf, zero..p1, top]$
8	$[xg..inf/dec, yg..inf/dec, mPi2..mPi4/inc, zero..inf, zero..p1, top]$
9	$[xg..inf/dec, yg..inf/dec, mPi4/inc, zero..inf, zero..p1, top]$
10	$[xg..inf/dec, yg..inf/dec, mPi4..zero/inc, zero..inf, zero..p1, top]$
11	$[xg/std, yg/dec, zero/inc, zero..inf, zero..p1, top]$
	wheels_ power(minf..zero)
12	$[xg/std, yg/std, zero/std, minf..zero, zero, none]$

Tabela 4.6: Prvi izdelani plan za potiskanje kocke

	akcija **stanje** $[X, Y, Beta, Vel, Tau, Angle]$
1	$[xg..inf/std, yg..inf/std, zero/std, zero, zero..p1, top]$ *wheels_power(minf..zero)*
2	$[xg..inf/std, yg..inf/std, zero/std, minf..zero, zero, none]$ *wheels_power(zero)*
3	$[xg..inf/std, yg..inf/std, zero/std, zero, zero, none]$ *set_angle(right,zero)*
4	$[xg..inf/std, yg..inf/std, zero/std, zero, zero, right]$ *wheels_power(zero..inf)*
5	$[xg..inf/dec, yg..inf/std, zero/std, zero..inf, zero, right]$
6	$[xg/dec, yg..inf/std, zero/std, zero..inf, zero, right]$ *wheels_power(minf..zero)*
7	$[xg/std, yg..inf/std, zero/std, minf..zero, zero, none]$ *wheels_power(zero)*
8	$[xg/std, yg..inf/std, zero/std, zero, zero, none]$ *set_angle(top,zero)*
9	$[xg/std, yg..inf/std, zero/std, zero, zero, top]$ *wheels_power(zero..inf)*
10	$[xg/std, yg..inf/dec, zero/std, zero..inf, zero, top]$
11	$[xg/std, yg/dec, zero/std, zero..inf, zero, top]$

Tabela 4.7: Drugi izdelani plan za potiskanje kocke

Poglavje 5

Zaključek

5.1 Doseženi cilji

V diplomski nalogi smo pokazali, da je z modificiranim QSIM algoritmom možno sestaviti plan za vodeneje robota, pri katerem se jasno vidi, koliko znanja je bilo uporabljeno iz samega modela in koliko znanja dejansko vsebujejo akcije, s katerimi upravljamo robota. Implementirana je bila tudi dodatna omejitev monotone odvisnosti od več spremenljivk. Odvisnost je bila uspešno testirana na kvalitativnem modelu umetne domene. Tako je mogoče program za kvalitativno planiranje izvajati tudi na modelih, zgrajenih z algoritmom QUIN. Implementiran je bil vmesnik, ki je uspešno omogočil izvajanje generiranega plana na simulatorju. Tako nam je uspelo v domeni potiskanja pokončnega valja kot tudi kocke, s ponavljajočim se planiranjem in izvajanjem plana, potisniti predmet iz začetnega v končni položaj.

5.2 Izzivi in pridobljene izkušnje

Pri eksperimentiranju dodeljevanja cene akcijam se je izkazalo, da cena akcije ni dražja od običajnega prehoda kvalitativnega stanja. To ima preprosto razlago, saj kot smo omenili, imamo lahko pri simuliranih prehodih lažno obnašanje, pri akcijskem prehodu pa imamo željeno spremembo zagotovljeno.

Kot pomemben dosežek bi tudi izpostavili izbiro hevristične ocene, ki temelji na QMDpD, kjer z razliko od QMD, upoštevamo tudi smer spreminjanja spremenljivke. To se izkaže za pomembno, saj s tem zgodaj usmerimo iskanje v pravo smer z izbiro primernejše akcije. Učinek akcij je navadno spreminjanje smernega določila simuliranih spremenljivk. V poglavju 2.5.1 je bila predsta-

vljena primerjava, kjer se je hevristika TotalQMDpD, na konkretnem primeru, pokazala kot izrazito boljša od hevristike TotalQMD. Zanimiva se nam zdi tudi kompaktna definicija plana. Plan je definiran kot seznam kvalitativnih stanj. Tak plan je človeku blizu. Izvajalec plana pa zlahka ugotovi, katero akcijo je potrebno sprožiti in kakšen je njen klic.

Skupina izzivov, ki v delu ni dobila prave pozornosti, je implementacija vmesnika. Ta je implementiran v programskem jeziku Go [9].

- Včasih se pojavi težava, da senzorji napačno zaznajo numerično stanje. Tako lahko dobimo kvalitativno stanje, ki sploh ni v skladu s kvalitativnim modelom. Denimo, da se takoj po končanem potiskanju predmeta robot začne premikati vzvratno. V tem primeru preberemo stanje, da se robot pomika vzvratno, vendar naj bi se še vedno dotikal predmeta. Tako stanje moramo zavreči. Rešitev je, da preden kvalitativno stanje predamo izvajalcu plana, najprej preverimo, če je v skladu s kvalitativnim modelom. Na ta način filtriramo napake pri branju numeričnih podatkov.

- Ker pri izvajanju plana vsako nepričakovano stanje povzroči ponovno planiranje, smo pri pretvarjanju numeričnih vrednosti v kvalitativne uvedli histerezo. Za spremembo kvalitativne veličine iz intervala v odlikovano vrednost vzamemo ε okolico odlikovane vrednosti. Ko pa kvalitativna veličina zapušča ε okolico odlikovane vrednosti in se pomika proti intervalu, pa mora numerična vrednost spremenljivke zapustiti $\varepsilon \times k$, pri čemer je faktor $k > 0$. Faktor k določimo s poizkušanjem. Za nemoteno delovanje je pogoj, da so odlikovane vrednosti vsaj v razmaku $\varepsilon \times (k+1)$. Na ta način dobimo robustno spreminjanje kvalitativnih stanj.

- Zanimiva je tudi preprosta implementacija kompleksne akcije, ki robota pripelje na točno določeno stranico predmeta, oziroma na rob stranice predmeta. Pri tem lokacijo, kjer mora robot končati, ni težko izračunati. Nato z nizom preprostejših akcij, ki temeljijo na sledenju krožnici in sledenju ravni črti, sestavimo pot, ki jo naj opravi robot. Robot se najprej zapelje v krožnico okoli predmeta. V krožnici kroži toliko časa, da je njegova usmeritev paralelna na ciljno usmeritev, vendar gleda ravno v obratno smer. Iz te pozicije izračunamo krožnico do ciljnega položaja, s središčem točno med pozicijo robota in ciljno točko dotikališča na stranici predmeta.

5.3 Ideje za nadaljnje delo

- Za vsako operativno območje poizkusimo izračunati verjetnost kvalitativnih prehodov. Na ta način bi omilili vpliv lažnih obnašanj kvalitativnega modela. Verjetnosti bi lahko določili na podlagi učnih primerov. Določena območja bi se dodatno razdelila in v vsakem od njih bi verjetnosti prehodov imele različne vrednosti. V prostoru stanj, kjer je kvalitativnim prehodom dodana verjetnost, bi s pomočjo metod mehkega planiranja [2] poiskali tak plan, katerega potek bo možno najverjetneje uresničiti.

- Denimo, da imamo omejitvi $\Delta X = M^{+,+}(I, K)$ in $\Delta Y = M^{-,-}(I, K)$. Ti nam opisujeta histrost spreminjanja po koordinati X in hitrost spreminjanja po koordinati Y. Če je naše ciljno stanje neka pozicija (x_{xg}, y_{yg}), potem mora vsebovati spremenljivki X in Y in omejitvi $deriv(X, \Delta X)$ in $deriv(Y, \Delta Y)$. Vendar pa v tem primeru planer ne uporabi znanja o razmerju histrosti spreminjanja. Menimo, da bi bilo dobro raziskati, na kakšen način bi razmerje upoštevali.

- Generirani modeli ne vsebujejo korespondenčnih vrednosti za omejitve monotonosti. Smiselno se nam zdi raziskati vpliv dodajanja korespondenčnih vrednosti med izvajanjem plana. Še posebej v primerih, ko je potrebno ponovno planiranje. Če v tem stanju ne obstajajo ustrezne odlikovane vrednosti, jih dodamo v ustrezne domene.

Dodatek A

QSIM v prologu

Algoritem QSIM povzet iz knjige [3]. Program v tem dodatku je razširitev QSIM programa v prologu v knjigi [3].

```
% An interpreter for Qualitative Diferential
   Equations
:- op(100, xfx, ..).

conc([], L, L).

conc([H|T], L2, [H|L]):-
  conc(T, L2, L).


% landmarks(Domain, [Land1, Land2, ...])
% Land1, Land2 etc. are landmarks for Domain
% This part of qualitative model definition, user-
   defined

% correspond(Constraint):
% Constraint specifies corresponding values for type
   of constraint.

correspond(sum(_Dom1:zero,_Dom2:zero,_Dom3:zero)).

correspond(sum(Dom1:L, _Dom2:zero, Dom1:L)) :-
  qmag(Dom1:L), L \== zero, not(L = _..._).    % L is
     nonzero landmark in Dom1
```

```
correspond(sum(_Dom1:zero, Dom2:L, Dom2:L)) :-
  qmag(Dom2:L), L \== zero, not(L = _..._).    % L is
     nonzero landmark in Dom2


qmag(Domain:Qm) :-
  landmarks(Domain, Lands),
  qmag(Lands, Qm).

qmag(Lands, L) :-
  member(L, Lands),
  L\==minf, L\==inf. % A finite landmark
qmag(Lands, L1..L2) :- % Interval
    conc(_, [L1,L2|_], Lands).  % Two adjacent
       landmarks

% relative_qmag(Domain1:QM, Domain2:Landmark, Sign):
% Sign is the sign of the difference between QM and
   Landmark
% if QM < Landmark then Sign is neg, etc.
relative_qmag(Domain:Ma..Mb, Domain:Land, Sign) :- !
   ,
  landmarks(Domain, Lands),
  (compare_lands(Ma, Land, Lands, neg), Sign=neg,!
   ;
   Sign = pos
  ).

relative_qmag(Domain:M1, Domain:M2, Sign) :-
  landmarks(Domain, Lands),
  compare_lands(M1, M2, Lands, Sign),!.

% qdir(Qdir, Sign):
% Qdir is qualitative direction of change with Sign
qdir(dec, neg).
qdir(std, zero).
qdir(inc, pos).
```

```
% Laws of qualitative summation
% qsum(Q1, Q2, Q3):
% Q3 = Q2 + Q1, qualitative sum over domain [pos,
   zero, neg]
qsum(pos, pos, pos).
qsum(pos, zero, pos).
qsum(pos, neg, pos).
qsum(pos, neg, zero).
qsum(pos, neg, neg).

qsum(zero, pos, pos).
qsum(zero, zero, zero).
qsum(zero, neg, neg).

qsum(neg, pos, pos).
qsum(neg, pos, zero).
qsum(neg, pos, neg).
qsum(neg, zero, neg).
qsum(neg, neg, neg).

% qdirsum(D1, D2, D3)
% qualitative sum overdirections of change
qdirsum(D1, D2, D3) :-
  qdir(D1, Q1), qdir(D2, Q2), qdir(D3, Q3),
  qsum(Q1, Q2, Q3).

% sum(QV1, QV2, QV3):
% qualitative sum over qualitative values of form
   Domain:Qmag/Dir
% When called, this predicate assume that the
% domains of all arguments are instantiated

sum(D1:QM1/Dir1,D2:QM2/Dir2,D3:QM3/Dir3) :-
  qdirsum(Dir1,Dir2,Dir3),
                                        %Directions of
     change: Dir1 + Dir2 = Dir3
  qmag(D1:QM1), qmag(D2:QM2), qmag(D3:QM3),  % QM1+
     QM2=QM3 must be consistent with all corresponding
      values
```

```
  not((
    correspond(sum(D1:V1,D2:V2,D3:V3)), % V1 + V2 =
       V3
    relative_qmag(D1:QM1,D1:V1,Sign1),
    relative_qmag(D2:QM2,D2:V2,Sign2),
    relative_qmag(D3:QM3,D3:V3,Sign3),
    not(qsum(Sign1,Sign2,Sign3))
  )).

% mplus(X, Y):
%   Y is a monotonically increasing function of X.

mplus(D1:QM1/Dir,D2:QM2/Dir) :-
  qmag(D1:QM1), qmag(D2:QM2),
    % QM!, QM2 consistent with all corresponding
       values between D1, D2:
  not((
    correspond(D1:V1,D2:V2),
    relative_qmag(D1:QM1,D1:V1,Sign1),
    relative_qmag(D2:QM2,D2:V2,Sign2),
    Sign1 \== Sign2
  )).

% oposite direction
opdir(dec, inc).
opdir(std, std).
opdir(inc, dec).

% oposite direction
opsign(neg, pos).
opsign(zero, zero).
opsign(pos, neg).

mminus(D1:QM1/Dir,D2:QM2/Dir2) :-
  opdir(Dir,Dir2),
  qmag(D1:QM1), qmag(D2:QM2),
    % QM!, QM2 consistent with all corresponding
       values between D1, D2:
  not((
```

```
    correspond(D1:V1,D2:V2),
    relative_qmag(D1:QM1,D1:V1,Sign1),
    relative_qmag(D2:QM2,D2:V2,Sign2),
    Sign1 \== Sign2
  )).


% deriv(Var1, Var2):
%   time derivate of Var1 is quallitatively equal
   Var2

deriv(_Dom1:_Qmag1/Dir1,Dom2:Qmag2/_Dir2) :-
 qdir(Dir1,Sign1),
 qmag(Dom2:Qmag2),
 % Sign2=sign of QMag2
 relative_qmag(Dom2:Qmag2,Dom2:zero,Sign2),
 Sign1=Sign2.

mderiv(_Dom1:_Qmag1/Dir1,Dom2:Qmag2/_Dir2) :-
  opdir(Dir1, Dir11),
  qdir(Dir11,Sign1),
  qmag(Dom2:Qmag2),
  % Sign2=sign of QMag2
  relative_qmag(Dom2:Qmag2,Dom2:zero,Sign2),
  Sign1=Sign2.

% transition(Domain:Qmag1/Dir1,Domain:Qmag2/Dir2):
% Variable state transitions between `close` time
   points

transition(Dom:L1..L2/std,Dom:L1..L2/Dir2) :-
 qdir(Dir2, _AnySign).

transition(Dom:L1..L2/inc, Dom:L1..L2/inc).

transition(Dom:L1..L2/inc, Dom:L1..L2/std).

transition(Dom:_L1..L2/inc, Dom:L2/inc) :-
 L2 \== inf.
```

```
transition(Dom:_L1..L2/inc, Dom:L2/std) :-
 L2 \== inf.

transition(Dom:L1..L2/dec, Dom:L1..L2/dec).

transition(Dom:L1..L2/dec, Dom:L1..L2/std).

transition(Dom:L1.._L2/dec, Dom:L1/dec) :-
 L1 \== minf.

transition(Dom:L1.._L2/dec, Dom:L1/std) :-
 L1 \== minf.

transition(Dom:L1/std, Dom:L1/std) :-
 \+ L1 = _A.._B.

transition(Dom:L1/std, Dom:L1..L2/inc) :-
 qmag(Dom:L1..L2).

transition(Dom:L1/std, Dom:L0..L1/dec) :-
 qmag(Dom:L0..L1).

transition(Dom:L1/inc, Dom:L1..L2/inc) :-
 qmag(Dom:L1..L2).

transition(Dom:L1/dec, Dom:L0..L1/dec) :-
 qmag(Dom:L0..L1).

controlled(_Dom:QM):-
   QM \= _Qm/_Dir,!.

% system_trans(State1,State2):
%   System state transition;
%   system state is a list of variable values
system_trans([],[]).

system_trans([Val1|Vals1], [Val2|Vals2]) :-
  (
```

```
  controlled(Val1),
  Val1 = Val2
  ;
  transition(Val1,Val2)
  ),
  system_trans(Vals1,Vals2).

% legal_trans(State1,State2):
%    possible transition between states according to
   model
legal_trans(State1,State2) :-
  % legalstate(State2),
   system_trans(State1,State2),
  % Qualitatively different next state
   State1 \== State2,
   not((
       point_state(State1),
       point_state(State2)
      )),
   int_legalstate(State2).
                                          % Legal
      according to model

int_legalstate(State):-
  legalstate(State),!.

point_state(State) :-
    member(_:Qmag/Dir, State),
    not(Qmag=_..._),
    Dir \== std.

% simulate(SystemStates,MaxLength):
%  SystemStates is a sequence of states of simulated
   system
%  not longer than MaxLength

simulate([State], MaxLength) :-
  ( MaxLength = 1
    ;
```

```
    not(legal_trans(State,_))
  ),!.

simulate([State1,State2|Rest], MaxLength) :-
  MaxLength > 1, NewMaxL is MaxLength - 1,
  legal_trans(State1, State2),
  simulate([State2|Rest], NewMaxL).

% simulate( InitialState, QualitativeBehaviour,
   MaxLength)

simulate(InitialState, [InitialState|Rest], MaxLength
   ) :-
  legalstate(InitialState),
  simulate([InitialState|Rest], MaxLength).

% compare_lands(X1,X2,List,Sign):
%   if X1 before X2 in List then Sign = neg
%   if X2 before X1 then Sign = pos else Sign = zero

compare_lands(X1,X2,[First|Rest],Sign) :-
  X1 = X2,!, Sign=zero
  ;
  X1 = First,!,Sign=neg
  ;
  X2 = First,!,Sign=pos
  ;
  compare_lands(X1,X2,Rest,Sign).
```

Dodatek B

Model valja v prologu

```
landmarks(vel, [minf, zero, inf]).
landmarks(angle, [top, right, bottom, left, none]).
landmarks(x, [minf, zero, xg, inf]).
landmarks(y, [minf, zero, yg, inf]).

region([_, _, _, angle:A], A) :-
        member(A, [top, bottom, left, right, none]).
region([_, _, vel:minf..zero, _], backing).

legalstate(A) :-
        forall(region(A, B), legalstate(B, A)).

legalstate(all, [_, _, vel:_, angle:A]) :-
        member(A, [top, bottom, left, right, none]).
legalstate(top, [C, A, vel:B, _]) :-
        mderiv(A, vel:B/std),
        stdy(C).
legalstate(bottom, [C, A, vel:B, _]) :-
        deriv(A, vel:B/std),
        stdy(C).
legalstate(right, [A, C, vel:B, _]) :-
        mderiv(A, vel:B/std),
        stdy(C).
legalstate(left, [A, C, vel:B, _]) :-
        deriv(A, vel:B/std),
        stdy(C).
```

```
legalstate(none, [A, B, _, _]) :-
        stdy(A),
        stdy(B).
legalstate(backing, [A, B, _, _]) :-
        stdy(A),
        stdy(B).

action([_, _, vel:zero, angle:none], [_, _, vel:zero,
    angle:A], set_angle(A)) :-
        member(A, [top, bottom, left, right]).
action([_, _, vel:_, angle:none], [_, _, vel:zero,
   angle:none], wheels_power(zero)).
action([_, _, vel:_, angle:A], [_, _, vel:minf..zero,
    angle:none], wheels_power(minf..zero)) :-
        A\=none.
action([_, _, vel:_, angle:A], [_, _, vel:B, angle:A]
   , wheels_power(B)) :-
        A\=none,
        member(B, [zero, zero..inf]).

stdy(_:_/std).
```

Dodatek C

Model kocke v prologu

```
landmarks(vel, [minf, zero, inf]).
landmarks(angle, [top, right, bottom, left, none]).
landmarks(tau, [minf, m1, zero, p1, inf]).
landmarks(x, [minf, zero, xg, inf]).
landmarks(y, [minf, zero, yg, inf]).
landmarks(beta, [minf, mPi, m3Pi4, mPi2, mPi4, zero,
   pPi4, pPi2, p3Pi4, pPi, inf]).

region([_, _, _, _, _, _], any).
region([_, _, _, _, _, angle:none], stdy).
region([_, _, _, vel:A, _, _], stdy) :-
        A\==zero..inf.
region([_, _, beta:mPi/_, _, _, angle:A], B) :-
        change_region_ang(A, s/p, B).
region([_, _, beta:A/_, _, _, angle:C], D) :-
        relative_qmag(beta:A, beta:mPi, B),
        B==pos,
        relative_qmag(beta:A, beta:mPi2, neg),
        change_region_ang(C, m/p, D).
region([_, _, beta:mPi2/_, _, _, angle:A], B) :-
        change_region_ang(A, m/s, B).
region([_, _, beta:A/_, _, _, angle:C], D) :-
        relative_qmag(beta:A, beta:mPi2, B),
        B==pos,
        relative_qmag(beta:A, beta:zero, neg),
        change_region_ang(C, m/m, D).
```

```
region([_, _, beta:zero/_, _, _, angle:A], B) :-
        change_region_ang(A, s/m, B).
region([_, _, beta:A/_, _, _, angle:C], D) :-
        relative_qmag(beta:A, beta:zero, B),
        B==pos,
        relative_qmag(beta:A, beta:pPi2, neg),
        change_region_ang(C, p/m, D).
region([_, _, beta:pPi2/_, _, _, angle:A], B) :-
        change_region_ang(A, p/s, B).
region([_, _, beta:A/_, _, _, angle:C], D) :-
        relative_qmag(beta:A, beta:pPi2, B),
        B==pos,
        relative_qmag(beta:A, beta:pPi, neg),
        change_region_ang(C, p/p, D).
region([_, _, beta:pPi/_, _, _, angle:A], B) :-
        change_region_ang(A, s/p, B).
region([_, _, _, vel:zero..inf, _, angle:A], pushing)
    :-
        A\=none.

legalstate(A) :-
        forall(region(A, B), legalstate(B, A)).

legalstate(any, [_, _, beta:A/_, _, _, _]) :-
        relative_qmag(beta:A, beta:mPi, B),
        (   B==pos
        ;   B==zero
        ),
        relative_qmag(beta:A, beta:pPi, C),
        C==neg.
legalstate(A/D, [B, E, _, vel:C, _, _]) :-
        (   A==p,
            deriv(B, vel:C/std)
        ;   A==m,
            mderiv(B, vel:C/std)
        ;   A==s,
            stdy(B)
        ),
        (   D==p,
```

```
            deriv(E, vel:C/std)
        ;   D==m,
            mderiv(E, vel:C/std)
        ;   D==s,
            stdy(E)
        ).
legalstate(stdy, [A, B, C, _, _, _]) :-
        stdy(A),
        stdy(B),
        stdy(C).
legalstate(pushing, [_, _, A, _, tau:B, _]) :-
        deriv(A, tau:B/std).

action([_, _, _, vel:zero, _, angle:none], [_, _, _,
   _, tau:B, angle:A], set_angle(A, B)) :-
        member(A, [top, bottom, left, right]),
        member(B, [m1..zero, zero, zero..p1]).
action([_, _, _, vel:_, _, angle:none], [_, _, _, vel
   :zero, _, angle:none], wheels_power(zero)).
action([_, _, _, vel:_, _, A], [_, _, _, vel:minf..
   zero, tau:zero, angle:none], wheels_power(minf..
   zero)) :-
        A\=angle:none.
action([_, _, _, vel:_, _, A], [_, _, _, vel:B, _, A]
   , wheels_power(B)) :-
        A\=angle:none,
        member(B, [zero, zero..inf]).

stdy(_:_/std).

change_region_ang(top, A, A).
change_region_ang(left, A, B) :-
        change_region(A, B).
change_region_ang(bottom, A, C) :-
        change_region(A, B),
        change_region(B, C).
change_region_ang(right, A, D) :-
        change_region(A, B),
        change_region(B, C),
```

```
        change_region(C, D).

change_region(s/p, m/s).
change_region(p/p, p/m).
change_region(m/s, s/m).
change_region(p/m, m/m).
change_region(s/m, p/s).
change_region(m/m, m/p).
change_region(p/s, s/p).
change_region(m/p, p/p).
```

Slike

Tabele

Literatura

[1] Box2D | A 2D Physics Engine for Games, 2016. Dostopno na `www.box2d.org/`. 21

[2] Jim Blythe. An overview of planning under uncertainty. Objavljeno v *Artificial intelligence today*, strani 85–110. Springer, 1999. 37

[3] Ivan Bratko. *Prolog Programming for Artificial Intelligence, 4th edition.* Pearson Education, Addison-Wesley, 2012. 4, 6, 10, 14, 38

[4] Johan De Kleer in John Seely Brown. A qualitative physics based on confluences. *Artificial intelligence*, 24(1-3):7–83, 1984. 5

[5] Richard Fikes in Nils J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artif. Intell.*, 2(3/4):189–208, 1971. Dostopno na `http://dx.doi.org/10.1016/0004-3702(71)90010-5`. 6

[6] Kenneth D. Forbus. Introducing Actions into Qualitative Simulation. Objavljeno v *Proceedings of the 11th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, August 1989*, strani 1273–1278, 1989. Dostopno na `http://ijcai.org/Proceedings/89-2/Papers/068.pdf`. 3

[7] Peter E. Hart, Nils J. Nilsson, in Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Systems Science and Cybernetics*, 4(2):100–107, 1968. Dostopno na `http://dx.doi.org/10.1109/TSSC.1968.300136`. 10

[8] John C. Hogge. Compiling Plan Operators from Domains Expressed in Qualitative Process Theory. Objavljeno v *Proceedings of the 6th National Conference on Artificial Intelligence. Seattle, WA, July 1987.*, strani 229–233, 1987. Dostopno na `http://www.aaai.org/Library/AAAI/1987/aaai87-041.php`. 3

[9] Google Inc. The Go Programming Language, 2016. Dostopno na `www.golang.org/`. 36

[10] Benjamin Kuipers. Qualitative Simulation. *Artif. Intell.*, 29(3):289–338, 1986. Dostopno na `http://dx.doi.org/10.1016/0004-3702(86)90073-1`. 6

[11] Y. Li, K.H. Ang, in G.C.Y. Chong. Patents, software and hardware for PID control: an overview and analysis of the current art. *IEEE Control Systems Magazine*, 26(1):42–54, 2006. Dostopno na `http://eprints.gla.ac.uk/3816/1/IEEE2pdf.pdf`. 21

[12] Miha Troha in Ivan Bratko. Qualitative learning of object pushing by a robot. Objavljeno v *25th International Workshop on Qualitative Reasoning, Barcelona, Spain*, strani 175–180, 2011. 3, 4, 14

[13] Anja VanDerHulst. SWI-Prolog, 2016. Dostopno na `www.swi-prolog.org/`. 4

[14] Timothy Wiley, Claude Sammut, in Ivan Bratko. Planning with Qualitative Models for Robotic Domains. Objavljeno v *Advances in Cognitive Systems (Poster Collection), Second Annual Conference on Cognitive Systems*, strani 251–266, 2013. 10, 13

[15] Timothy Wiley, Claude Sammut, in Ivan Bratko. Qualitative Planning with Quantitative Constraints for Online Learning of Robotic Behaviours. Objavljeno v *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, strani 2578–2584, 2014. Dostopno na `http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8475`. 3

[16] Jure Žabkar. *Učenje kvalitativnih odvisnosti*. Doktorska disertacija, Univerza v Ljubljani, 2010. 4

[17] Jure Žabkar, Ivan Bratko, in Janez Demšar. Learning qualitative models through partial derivatives by Padé. Objavljeno v *Proceedings of the 21th International Workshop on Qualitative Reasoning, Aberystwyth, U.K*, 2007. 4

[18] Domen Šoberl, Jure Žabkar, in Ivan Bratko. Qualitative Planning of Object Pushing by a Robot. Objavljeno v *Foundations of Intelligent Systems - 22nd International Symposium, ISMIS 2015, Lyon, France,*

October 21-23, 2015, Proceedings, strani 410–419, 2015. Dostopno na http://dx.doi.org/10.1007/978-3-319-25252-0_44. 3, 4, 14

[19] Dorian Šuc. *Machine reconstruction of human control strategies*. Doktorska disertacija, Univerza v Ljubljani, 2001. 4

[20] Dorian Šuc in Ivan Bratko. Induction of Qualitative Trees. Objavljeno v *Machine Learning: EMCL 2001, 12th European Conference on Machine Learning, Freiburg, Germany, September 5-7, 2001, Proceedings*, strani 442–453, 2001. Dostopno na http://dx.doi.org/10.1007/3-540-44795-4_38. 14, 16