KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

PROGRAMŲ INŽINERIJOS KATEDRA

DAINIUS VAIKŠNYS

ELGESIU PAREMTOS ROBOTIKOS SIMULIAVIMO APLINKŲ, SKIRTŲ PROGRAMAVIMO MOKYMUI, TYRIMAS

Magistro darbas

Vadovas

doc. dr. T. Blažauskas

Kaunas, 2014

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

PROGRAMŲ INŽINERIJOS KATEDRA

DAINIUS VAIKŠNYS

ELGESIU PAREMTOS ROBOTIKOS SIMULIAVIMO APLINKŲ, SKIRTŲ PROGRAMAVIMO MOKYMUI, TYRIMAS

Magistro darbas

Vadovas

doc. dr. T. Blažauskas

Recenzentas

prof. dr. R. Damaševičius

Studentas

D. Vaikšnys

Kaunas, 2014

Summary

The process of robot creation is extremely expensive. It requires a lot of expenses to build a prototype and the mistakes in design can lead to several rebuilds. No less expensive is a challenge of designing control program for the robot, as testing it with a prototype can cause harm to the construction or even destroy it.

This document focuses on the simulation of robot behavior in virtual environment. Egzisting simulation environmets, that allow programming and simulating of control scripts, are being explored here. A lot of attention is paid to simplicity of the software interface and possible usage for children to learn and enjoy programming. The author makes a proposal for new simulation environment and implements the software.

The research is made on the implementation of control programs for robots. The way how the control program is implemented is compared to an egzisting comercial simulation environment – Webots. The two specifications for robot design and behavior is provided and the solutions for control programs are designed and compared. The modification in robot specifications are brought up and those solutions are modified to comply with the change.

Finally the designed software is presented and explained. The chapter contains user guide, a specification for graphical user interface and an explanation of it‘s usage.

TURINYS

1. ĮVADAS

1.1. Darbo tikslai ir rašymo aplinkybės

Informatikos specialistų paklausa šiandien yra viena didžiausių iš visų specialybių. Lietuvoje yra jaučiamas programuotojų trūkumas. Informatikos studijas baigia nepakankamas kiekis studentų. Reikalingi pokyčiai švietimo sistemoje kad paskatinti moksleivių susidomėjimą programavimu. Galimas sprendimas yra informatikos pamokose naudoti roboto simuliavimo aplinką. Tokiu būdu moksleiviai mokomi programuoti rašant roboto valdančios programos kodą. Tai skatina vaikų susidomėjimą tiek pačia robotikos sritimi, tiek ir programavimu. Toks programavimas yra patrauklesnis, nes yra aiškus atgalinis ryšys ir skatinantis rezultatas.

Yra atlikta mokslinių tyrimų analizuojančių galimybes panaudojant robotus kaip mokymo objektus. Straipsnyje [1] iš „X World Conference on Computers in Education“ yra apibendrinama kad toks būdas ne tik padidina mokymo efektyvumą, bet ir suteikia moksleiviams praktinių įgūdžių labai reikalingų šiandieninėje rinkoje.

Siūlomas sprendimas yra kaip palankesnė alternatyva realiai robotikos įrangai, nes biudžetinės įstaiga negalėtų sau leisti įsigyti realios roboto platformos, nekalbant apie visos klasės paruošimą. Virtuali simuliavimo aplinka išsprendžia ir problemą dėl įrangos sugadinimo, nes mokyklinio amžiaus vaikai ne visada elgiasi atsargiai su svetimu turtu.

Šio darbo tyrimo sritis yra roboto valdančios programos kūrimas, siekiant simuliuoti norimą roboto elgesį realiu laiku. Tyrimo objektas – elgesiu paremtos robotikos simuliavimo aplinkos.

Darbo metu yra siekiama:

- išsiaiškinti kokios roboto simuliavimo aplinkos egzistuoja rinkoje
- išnagrinėti problemas kylančias kuriant tokią simuliavimo aplinką
- pasiūlyti projektą ir sukurti naują programinę įrangą
- ištirti sistemą roboto valdančios programos kūrimo aspektu
- paaiškinti kaip dirbti su naujai sukurta sistema

1.2. Elgesiu paremtos robotikos simuliavimo aplinka

Kurti robotus valdančias programas galima naudojant realią roboto platformą. Tačiau tai yra brangus procesas ir reikalauja nemažai techninių žinių. Visa tai palengvinti yra naudojama programinė įranga simuliuojanti robotą virtualioje erdvėje. Toks simuliatorius leidžia sukonstruoti virtualų prototipą, medžiagoms neskiriant lėšų. Jis taip pat leidžia apsirašyti valdymo algoritmus ir vykdant simuliaciją stebėti roboto elgseną virtualioje erdvėje. Tokiu būdu nerizikuojama padaryti žalos fiziniam turtui, išvengiama kai kurių sudėtingų roboto kūrimo proceso etapų.

Šiame dokumente nagrinėjama roboto modeliavimo aplinka yra priskiriama elgesiu paremtos robotikos (behavior-based robotics) kategorijai [2]. Šios kategorijos robotai yra valdomi vykdant paprastus lygiagrečius procesus, nusakančius elgesį. Klasikinis roboto valdymas skiriasi tuo, kad viena komanda aktyvuoja variklių judesį fiksuotam laiko intervalui, griežtai apibrėžtais laiko momentais nuo veiksmo pradžios iki pabaigos. Tuo tarpu elgesiu paremti robotai nesivadovauja griežtu veiksmų grafiku, bet artimai susieja jutiklių rodmenis su variklių judesiais.

1.3. Santrauka

Roboto kūrimas yra nepaprastai brangus procesas. Sukonstruoti prototipą sunaudojama daug lėšų, o paaiškėjus trūkumams gali tekti jį perdaryti ne vieną kartą. Nemažiau sudėtingas yra algoritmų kūrimas roboto valdymui. Išbandyti valdančią programą su realiu prototipu sudėtinga, nes klaidos gali sukelti tiek smulkių gedimų tiek stipriai apgadinti įrangą.

Šiame dokumente yra gilinamasi į roboto elgesio simuliavimą virtualioje erdvėje. Nagrinėjamos roboto simuliavimo aplinkos, teikiančios galimybę suprogramuoti ir išbandyti robotą valdančius algoritmus. Kreipiamas didelis dėmesys į tokios sistemos naudojimo paprastumą ir galimybę ją

naudoti moksleivių skatinimui domėtis programavimu. Autorius pateikia projektą naujos simuliavimo aplinkos kūrimui ir realizuoja šią sistemą.

Atliekamas tyrimas siekiant nustatyti kaip yra kuriamos roboto valdančios programos. Yra palyginama naujai sukurtos sistemos galimybė realizuoti valdymo algoritmus su rinkoje esančiu komerciniu sprendimu – Webots. Yra pateikiamos dvi roboto konstrukcijos su apibrėžtu roboto elgesiu ir suprogramuojamas valdymo kodas abejose sistemose. Roboto elgesio reikalavimuose atliekamas pakeitimas ir tiriama kokią įtaką tai turi robotą valdančios programos kodui.

Galiausiai yra pristatoma naujai sukurta roboto simuliavimo aplinka. Aprašomas sistemos vartotojo vadovas, paaiškinama grafinė vartotojo sąsaja.

2. ANALITINĖ DALIS

2.1. Tikslas

Projekto tikslas yra sukurti įrankių komplektą, leidžiantį sukonstruoti virtualų roboto prototipą, pateikti valdantį algoritmą (arba panaudoti pridėtą AI algoritmą), apmokyti robotą bei stebėti jo autonomšką elgesį. Įrankiai turi būti intuityvūs ir lengvai naudojami neprofesionalios auditorijos.

2.2. Egzistuojantys sprendimai

Šis simuliatorius yra priskiriamas elgesiu paremtos robotikos (behavior-based robotics) kategorijai. Šios kategorijos simuliatorių yra sukurta tiek pramoniniam lygiui, tiek ir mėgėjiškam vartotojui.

2.2.1. Atviro kodo sprendimai

- Gazebo. Šis sprendimas yra daugelio robotų modeliavimo aplinka lauko sąlygomis. Jis geba modeliuoti robotų populiaciją, jutiklius ir objektus trimačiame pasaulyje. Modeliavimo aplinka generuoja tiek realistiškus jutiklių parodymus, tiek adekvačias sąveikas tarp objektų (standaus kūno fizikos modeliavimas).
- LpzRobots. Tai yra Leipcigo universiteto mokslinių tyrimų projektas. Jį sudaro daug subprojektų, iš kurių svarbiausi yra selforg – karkasas robotų valdymo kūrimui, ir ode_robots – trimatė, fiziškai realistiška roboto modeliavimo aplinka.
- ORCA Simulator for Robotics. ORCA modeliavimo aplinka yra galinga platforma robotų modeliavimui [6], kurią naudoja ne vienas pasaulyje žinomas projektas (GNOSYS, INDIGO, XENIOS, FIRST MM). Sistema palaiko individualias roboto bei aplinkos konfigūracijas. Sistemos principas yra centrinis serveris, prie kurio jungiasi visi moduliai per TCP-IP protokolą.
- Blender for Robotics. Blender yra trimatės grafikos kūrimo sistema, kuri turi daugybę modulių ir yra pilnai plečiama. Vienas pagrindinių robotikos komponentų yra Blender Game Engine [7], skirtas žaidimų kūrimui. Jo pagalba yra vykdomas roboto modeliavimas. Įvairias robotikos puses įgyvendina tam skirti robotikos moduliai.

2.2.2. Komerciniai sprendimai

- Microsoft Robotics Developer Studio. Tai yra vienas didžiausių robotikos projektų. Jis apima tiek programinės įrangos kūrimą robotams, tiek ir fizinį roboto konstravimą. Pagrindinis privalumas yra tas kad sukurta valdymo programa yra lengvai perkeliama į fizinį robotą. Be to modeliuojamus jutiklius galima įsigyti kaip ir kitas robotų dalis. Sistema taip pat stipriai supaprastina programos kūrimą pateikdama tokius įrankius kaip kodo generavimas naudojant grafinę sąsają.
- Webots. Tai yra kūrimo aplinka [8] skirta programuoti ir modeliuoti mobilius robotus. Vartotojas gali sukurti sudėtingas robotikos konfigūracijas su vienu ar keliais vienodais ar skirtingais robotais bendroje aplinkoje. Vartotojas gali parinkti objektų savybes, tokias kaip forma, spalva, tekstūra, masė, trintis ir panašiai.
- V-REP. VREP (virtuali robotų eksperimentavimo platforma) yra naudojama sparčiam algoritmų kūrimui, gamyklos automatizavimo modeliavimui, sparčiam prototipavimui ir verifikacijai, su robotika susijusiam švietimui, saugumo užtikrinimui ir tt. [9]. Kiekvienas modelis sistemoje gali būti atskirai valdomas įterptu skriptu, įskiepiu, nuotoliniu API klientu ar kitais būdais.
- RoboLogix. Šis sprendimas yra skirtas pramoninių robotų programų testavimui [10]. Su šia sistema galima apmokyti, testuoti, vykdyti, derinti programas parašytas penkių-ašių pramoniniam robotui. Taikymo sritys apima rūšiavimą, produktų sukrovimą, virinimą, dažymą ir tt., leidžiant sugalvoti savo aplinkas ar naujas taikymo sritis.

2.3. Programų sistemų savybių kiekybinis ir kokybinis palyginimas

2.3.1. Pagrindinių funkcijų palyginimas

Užsienio rinkoje yra tikrai nemažai sukurtų produktų. Verta palyginti esminius jų funkcionalumo kriterijus kad galėtume bendrai pažvelgti į situaciją rinkoje. Lentelėje 2.1 pateikiamas kiekybinis funkcijų palyginimas. Vertinamos funkcijos reikalingos simuliacijos aplinkoms, skirtoms programavimo mokymui.

2.1 lentelė. Kiekybinis funkcijų palyginimas

	Gazebo	ORCA	Blender	Webots	V-REP	Microsoft Robotics
Grafinė konstravimo sąsaja	-	-	+	+	+	+
Įterptiniai valdymo skriptai	-	-	+	+	+	-
Realistinis fizikos simuliatorius	+	+-	+	+	+	+
Daugelio robotų galimybė	+	-	+	+	+	+
Simuliacijos valdymas	-	-	-	+	+	-
Grafinis algoritmo apmokymas	-	-	-	-	-	-
Realistiška simuliacijos grafika	+	-	+	+	+	+
Standartinių robotų biblioteka	+	-	-	+	+	+

2.3.2. Naujas sprendimas

Visi minėti sprendimai yra solidžios sistemos, tačiau nei viena iš jų nėra specializuota mokyklinio amžiaus vaikams. Pagrindinės kliūtys trukdančios įsisavinti šiuos sprendimus švietimo sistemoje:

- Roboto detalės yra analogai realių robotikoje naudojamų komponentų. Detalių yra daug ir jos sudėtingos, turi daug parametrų. Neturint patirties robotų kūrime yra sunku išmokti dirbti su tokia sistema.
- Nėra grafinės vartotojo sąsajos arba sistema sudaryta iš kelių dalių, turinčių skirtingas vartotojo sąsajas.

Naujai kuriamai sistemai yra keliami tokie tikslai:

- teikti bazines robotų prototipų kūrimo funkcijas
- būti prieinama ir moksleiviams – nesunku naudotis sistema
- tapti pagrindiniu įrankiu robotikos mėgėjų veikloje

Sukurta sistema yra įrankių rinkinys turintis bendrą vartotojo sąsają, leidžiančią atlikti visus roboto prototipo kūrimo etapus. Modeliavimo aplinka pasižymi tokiomis savybėmis:

- Kiekvienas prototipo komponentas (konstrukcija, valdymo programa, modeliavimo savybės) gali būti įkeliamos atskirai. Tai reiškia kad norint sukurti valdymo algoritmą nereikia konstruoti savo roboto, galima įsikelti kitų sukonstruotus variantus.
- Modeliavimui naudojamas fizikos variklis Open Dynamics Engine. Tai plačiai naudojamas [4] variklis, užtikrinantis tikslų realaus laiko modeliavimą.
- Valdančiam skriptui aprašyti naudojama JavaScript scenarijų kalba. Ši kalba šiuo metu yra populiariausia [5] programavimo kalba GitHub bendruomenėje. Be to programos kodo nereikia kompiliuoti, todėl jis gali būti keičiamas nenutraukiant modeliavimo.
- Trimatės grafikos redaktorius yra atskiras programos komponentas, todėl redaktorių galima atnaujinti neperkompiliuojant pačios programos. Šis redaktorius yra naudojamas tiek konstruojant robotą, tiek ir peržiūrint modeliavimą.

Didelis siūlomos sistemos privalumas yra daugelio platformų palaikymas. Programa veikia Windows, Linux/X11 ir Mac OS X operacinėse sistemose. Todėl įrengti tinkamas darbo vietas galima ženkliai pigiau, kompiuteriuose diegiant nemokamas operacines sistemas.

2.4. Įgyvendinimo problemos

2.4.1. Konstravimo sąsajos problemos

2.4.1.1. Trimatė grafika

Egzistuoja daug trimatės grafikos redaktorių skirtų architektūrai, kompiuterinei animacijai, paveikslams kurti. Visi jie yra dideli projektai, vystyti eilę metų. Problema yra sukurti minimalistinį trimatės grafikos redaktorių, turintį tik nedidelį apibrėžtą funkcionalumo poaibį, tačiau neprarandant valdymo patogumo. Trimatės grafikos vaizdavimui reikalingas spartintuvas. Šiuo metu egzistuoja dviejų tipų konkurencingos spartintuvų tvarkyklės su programavimo sąsajom:

- DirectX
- OpenGL

Kadangi kuriama sistema yra daugiaplatforminė, bus pasirinktas OpenGL variantas, nes DirectX yra Microsoft produktas ir tinka tik jų operacinėm sistemom. OpenGL yra žemo lygio sąsaja, trurinti tik bazines funkcijas, kuriomis galima valdyti vaizdo adapterį. Kuriant konstravimo grafinę sąsaja reikalinga aukšto lygio biblioteka, apgaubianti OpenGL sąsają. Bus reikalinga pasirinkti labiausiai tinkamą šiam tikslui.

2.4.1.2. Roboto modelis

Kuriant virtualų roboto modelį iškyla kelios problemos – detalių formos aproksimacija ir aktyvių (judančių) dalių parametrų atitikimas. Pasyvių detalių forma yra svarbi kontaktui su kitais objektais. Pagrindinis uždavinys yra pateikti tokį konstravimo būdą, kad vartotojas galėtų ir sparčiai sumodeliuoti sugalvotą apytikslį robotą, ir atkartoti griežtai specifikuotas formas iš brėžinių.

Dažniausiai naudojamos robotų aktyvios detalės yra:

- Servo mechanizmai – tai įrenginiai kurie gali sukimo ašį atsukti į tiksliai nurodytą kampą leistiname diapazone.
- Varikliai – jų yra keletas tipų, jie gali sukti ašį į vieną arba abi puses, pastoviu greičiu arba pastovia jėga.
- Stūmokliai – atlieka postūmį vienoje ašyje į vieną ar kitą pusę.

Sistema turi turėti galimybę pridėti naujų tipų aktyvių detalių modelius bei vartotojas turi galėti keisti kiekvienos aktyvios detalės tipą ir parametrus. Be to kiekviena aktyvi detalė turi turėti valdymo sąsają kad vartotojo pateiktas skriptas galėtų ją valdyti.

2.4.1.3. Objektų manipuliavimas

Grafinė sąsaja turi turėti būdą pažymėti norimą objektą trimatėje aplinkoje. Tą reikia padaryti naudojant dvimatį įrenginį – pelę. Sprendimas turi būti intuityvus vartotojui. Taip pat reikalinga įrankių sąsaja manipuliuoti objektais. Tai yra kiekvienas konstravimo įrankis privalo turėti savo grafinę sąsają kuri vartotojui parodytų kaip atlikti manipuliaciją ir kokią įtaką jo veiksmai turi roboto konstrukcijai. Beto grafinė sąsaja turi būti papildyta parametrų redaktoriumi, kuris būtų grafinės sąsajos alternatyva įrankio manipuliacijai.

2.4.1.4. Jutiklių pridėjimas

Nei vienas robotas neapsieina be jutiklių. Jutikliai perteikia robotui informaciją apie jo aplinką. Yra labai platus spektras robotikoje naudojamų jutiklių. Reikia atrinkti iš visų jutiklių labiausiai reikalingus ir palikti sistemoje galimybę pridėti naujų jutiklių modelių. Jutiklio modelis turi generuoti tam tikro standartizuoto formato duomenis realiai interpretuodamas aplinką. Šiuos duomenis galės pasiekti vartotojo pateiktas valdymo skriptas.

Kitas uždavinys yra sugalvoti mechanizmą jutiklių pridėjimui prie roboto konstrukcijos. Vartotojas turi galėti paprastai nurodyti jutiklio poziciją bei stebėjimo kampą. Galbūt reikės sugalvoti jutiklių generuojamų duomenų atvaizdavimo būdą, kad vartotojas galėtų kalibruoti jutiklius, bei susipažinti su jų veikimo principu.

2.4.2. Valdančių programų kūrimas

2.4.2.1. Pateikimo būdas

Bet kurio roboto pagrindas yra valdanti programa. Jos esmė yra valdyti robotą siekiant kažkokio tikslo ar vykdant apibrėžtą funkciją, panaudojant jutiklių generuojamus duomenis. Virtualiam robotui valdyti taip pat reikalinga valdanti programa. Pagrindinis skirtumas yra tas, kad realaus roboto programą riboja techninė skaičiavimo įranga (mikroprocesoriaus architektūra, registrai ir tt.). Yra labai daug skirtingų techninių įrangų su skirtingais kodo kompiliatoriais, kurių programų kodai tarpusavyje nesuderinami. Tuo tarpu simuliuojamas robotas tokių apribojimų neturi ir valdančią programą riboja tik kuriama programinė įranga. Yra galimybė pasirinkti du variantus:

- Simuliuoti apribojimus pagal populiarius sprendimus. Tokiu atveju vartotojas rašytų programą tokiu formatu kurio reikalauja jo pasirinkta techninė įranga. Privalumas yra tas kad parašyta valdanti programa gali būti tiesiogiai perkelta į fizinį robotą neperrašant kodo. Tai naudinga didelėms programos turinčioms daug specializuoto konkrečiam sprendimui kodo. Pagrindinis trūkumas yra labai sudėtinga realizacija.
- Valdančiai programai naudoti populiarias kompiuterio programavimo/skripto kalbas, netaikant konkrečiai techninei platformai. Privalumai yra ženkliai lengvesnis valdančios programos kūrimas (aukštesnio lygio kalba), vartotojas gali kurti programą dar negalvodamas kokią techninę įrangą naudos, paprasta realizacija.

2.4.2.2. Valdančio kodo funkcijos

Reikia sugalvoti metodą kaip vartotojo pateikiama valdanti programa įsiterpia į roboto simuliaciją. Turi būti sukurtą sąsaja valdančiai programai gauti jutiklių duomenis. Taip pat reikalinga sąsaja valdančiai programai valdyti aktyvias roboto detales. Kadangi simuliacija susideda iš daugybės iteracijų laike, tikslinga yra valdančią programą išskaidyti į tris dalis:

- Pasiruošimas (Setup). Čia programa apsirašys reikalingas duomenų struktūras, suteiks reikalingiems kintamiesiems pradines reikšmes.

- Vykdymas (Process). Šioje dalyje aprašytas kodas bus vykdomas kiekvienos iteracijos metu. Programa turės priėjimą prie jutiklių duomenų, ir turės galimybę keisti aktyvių detalių būsenas.
- Apmokymas (Teaching). Ši dalis bus vykdoma kuomet vartotojas simuliacijos metu atliks apmokymą. Tai reiškia pateiks reikiamas roboto detalių būsenas esant dabartiniams jutiklių duomenims bei detalių būsenoms.

2.4.2.3. Kodo kontekstas

Reikia nuspręsti koks programos kontekstas bus pasiekiamas vartotojo pateiktai valdančiai programai. Įvairiems robotikos uždaviniams reikalingi įvairūs skaičiavimai. Jiems atlikti galima panaudoti egzistuojančias bibliotekas, tokias kaip:

- Matematinės funkcijos
- Skaičių matricų uždaviniai
- Geometriniai skaičiavimai
- Vaizdo apdorojimo funkcijos
- Neuroninių tinklų algoritmai
- ir kitos...

Problema yra kad jei daug įvairių funkcijų bus integruota į kontekstą, reali roboto platforma tokių funkcijų gali neturėti ir valdančios programos gali nepavykti transformuoti specifiniai techniniai įrangai. Kita vertus sudėtingų funkcijų trūkumas gali labai apsunkinti valdančios programos kūrimą, o neprofesionaliam vartotojui gali būti per sudėtinga sukurti net paprasčiausias programas.

2.4.3. Fizikos simuliacija

2.4.3.1. Alternatyvos

Realaus pasaulio fizikos simuliavimas yra labai sudėtingas uždavinys. Laimei egzistuoja jau sukurtų sprendimų, kurie yra vystyti ne vienerius metus. Populiariausi trimačio pasaulio fizikos simuliatoriai yra:

- ODE (Open Dynamics Engine). Šis sprendimas yra dažniausiai naudojamas jau sukurtuose robotikos simuliatoriuose. Taip pat šis fizikos variklis yra dažnas pasirinkimas kompiuterinių žaidimų kūrime.
- Bullet 3D. Tai yra trimatės fizikos biblioteka sukurta kompiuterinių žaidimų kūrimui. Keletas robotikos simuliatorių naudoja šį sprendimą.

Abu šie sprendimai yra atviro kodo, todėl puikiai tinka vykdomam projektui.

2.4.3.2. Integravimo problemos

Fizikos variklio integravimas nėra trivialus uždavinys. Tai atliekant reikės susidurti su tokiom problemom:

- Roboto aplinka. Kad fizikos simuliacija būtų realistinė, kiekvienas roboto aplinkoje esantis objektas, pradedant nuo paviršiaus kuriuo judės robotas, turi būti aprašytas fizikos variklio formatu. Tai reiškia kad kiekvienam objektui reikės nustatyti tokius parametrus kaip masė, trinties koeficientas, judėjimo laisvės laipsniai ir panašiai. Kadangi aplinka nėra pastovi, o ją suformuoja vartotojas, atsiranda nauja problema kaip patogiau ir tiksliau visą informaciją apie objektus turėtų pateikti vartotojas.

- Aktyviosios roboto detalės. Šių detalių veikimo principai turi sutapti su fizikos variklio palaikomais įrenginiais. Taip pat jie turi nemažai konfigūracijos parametrų, kurių vartotojas konstruodamas robotą gali nežinoti. Be to gali kilti problemų dėl šių detalių valdymo vartotojo pateiktoje valdymo programoje. Reikia taip apriboti leistinas valdymo reikšmes, kokios yra numatytos fizikos variklio specifikacijoje.
- Mastelis. Turi būti pateiktas toks konstravimo būdas kuriame aiškiai matytųsi detalių mastelis. Kitaip fizikos simuliacija gali atrodyti nerealistiškai. Be to roboto detalių masė turi būti paskaičiuota pagal detalė sudarančios medžiagos tankį ir tikslius detalės matmenis.

2.4.4. Algoritmo apmokymas

2.4.4.1. Neuroniniai tinklai

Nei vienas iš analizuotų sprendimų egzistuojančių rinkoje neturi galimybės apmokyti valdymo algoritmo. Vienas iš įdomiausių ir perspektyviausių robotų valdymo būdų yra dirbtinis intelektas. Viena žinomiausių dirbtinio intelekto rūšių yra dirbtiniai neuroniniai tinklai (Artificial neural network). Tai yra algoritmas [3], kuris gali būti apmokomas (galimos kelios apmokymo rūšys). Algoritmo įėjimas ir išėjimas yra masyvai normalizuotų skaičių (reikšmės nuo 0 iki 1). Esant pakankamam apmokymo porų skaičiui (įėjimas ir išėjmas) algoritmas gali pateikti gana tikslų rezultatą, pateikus įėjimo masyvą kurio nėra tarp apmokymo duomenų.

Analogiškai visus roboto jutiklių duomenis galima konvertuoti į normalizuotų skaičių masyvą, o aktyvių detalių valdymo duomenis atkurti iš algoritmo išėjimo masyvo. Tokiu būdu apmokius valdymo programoje naudojamą neuroninį tinklą, galima stebėti autonominį roboto elgesį simuliacijos metu.

2.4.4.2. Apmokymo duomenų gavimas

Neuroninio tinklo algoritmo rezultatai yra tuo priimtinesni, kuo daugiau apmokymo duomenų yra pateikta. Reikia sugalvoti tokį apmokymo būdą, kuris leistų vartotojui pateikti kuo daugiau apmokymo porų per kuo trumpesnį laiką. Taip pat reikia apmokymo procesą izoliuoti nuo pačio algoritmo, kad tuo pačiu būdu būtų galima apmokyti skirtingus algoritmus. Be to visus apmokymo duomenis reikia saugoti tokioje struktūroje, kad pakeitus algoritmą būtų galima panaudoti seniau sukurtus apmokymo duomenis.

2.4.4.3. Integravimas

Vienas iš galimų sprendimo būdų yra integruoti apmokymą į pačią simuliaciją. Jos vykdymo metu vartotojas galėtų reikiamu momentu pristabdyti simuliaciją ir grafinės sąsajos pagalba nurodyti roboto būsenas, kurių jis turėtu imtis toje situacijoje. Vartotojas reguliuodamas simuliacijos tempą, pristabdydamas, atsukdamas laiką atgal ar į priekį, galėtų pateikti apmokymo duomenis kritinėse situacijose. Po kiekvieno apmokymo simuliacija naudotų visus apmokymo duomenis ir kuo toliau tuo sklandesnis roboto elgesys būtų stebimas. Mokymas galėtų būti tęsiamas tol kol roboto elgesys tampa prognozuojamas ir atitinka vartotojo poreikius.

Reikės susidurti su problema kad surasti patogų būdą vartotojui nurodyti reikiamą roboto būseną. Reikia nuspręsti kaip bus integruojami neuroniniai tinklai į sistemą. Ar tai bus įtraukta į valdančios programos kontekstą, ar palikti tai realizuoti pačiai valdančiai programai.

2.4.4.4. Apmokymo redagavimas

Vien tik pateikti apmokymo duomenis neužtenka. Dažnai gali pasitaikyti klaidingų duomenų, kurie neigiamai įtakoja roboto elgesį. Gali pasikeisti ir elgesio poreikiai tam tikrose situacijose. Dėl to reikia įgyvendinti vartotojo sąsają, kur jis galėtų pašalinti, modifikuoti, dubliuoti ar kitaip redaguoti apmokymo duomenis. Gali būti numatyta ir galimybė perkelti vieno roboto apmokymo duomenis kitam robotui, transformuojant įėjimo ir išėjimo masyvus pagal robotų detalių skirtumus. Apmokymo duomenys turėtų būti išsaugoti projekto hierarchijoje kaip atskiras komponentas.

2.5. Analitinės dalies išvados

- Robotikos simuliacija yra aktuali tema
- Užsienyje yra sukurta nemažai panašių produktų
- Yra galimas sprendimas neturintis analogų rinkoje
- Įgyvendinant tokią sistemą susiduriama su daug problemų įskaitant konstravimo sąsajos kūrimą, valdančių programų integravimą, realistiškos fizikos simuliaciją bei neuroninių tinklų apmokymo iššūkius

3. PROJEKTINĖ DALIS

3.1. Reikalavimų specifikavimas

3.1.1. Sistemos paskirtis

Robotų simuliatorius yra kuriamas tam kad palengvintų robotų kūrimo procesą. Ši sistema leis kurti prototipus be jokių išorinių priemonių ar kitų išlaidų. Dėl to atsiras galimybė robotikos mėgėjams patiems išbandyti robotų kūrimą bei prisidėti prie robotikos vystymo.

3.1.2. Projekto kūrimo pagrindas

Šiuo metu rinkoje esantys robotų simuliatoriai yra orientuoti į robotikos inžinierius bei profesionalią robotų gamybą. Esant dideliam robotų populiarumui, studentai, moksleiviai ar tiesiog technikos mėgėjai nori susikurti savo robotą. To padaryti jiems dažnai nepavyksta nes reikalingos išlaidos yra neprieinamos, o surasta programinė įranga reikalauja aukštos kvalifikacijos specialisto. Dėl to reikalinga programinė įranga kuri būtų lengvai perprantama, nebrangi, nereikalautų papildomų išlaidų, bet tuo pačiu leistų susikurti realų roboto prototipą.

3.1.3. Sistemos tikslai

Sistema turi:

- teikti bazines robotų prototipų kūrimo funkcijas
- būti prieinama net ir moksleiviams
- būti įrankiu naudojamu robotikos mėgėjų veikloje

3.1.4. Užsakovai, pirkėjai ir kiti sistema suinteresuoti asmenys

Būtina turėti informaciją apie šiuos asmenis, kad išvengti skaudžių netikėtumų.

1. Užsakovas. Sistema kuriama pagal KTU Programų inžinerijos katedros užsakymą.
2. Pirkėjas. Sistema bus parduodama plačiajai auditorijai, kurią sudaro:
    - moksleiviai besidomintys robotika
    - elektronikos, mechanikos ar informatikos studentai
    - technologijų mėgėjai
3. Kiti sprendimus priimantys asmenys. Projekto vadovas – KTU Programų inžinerijos katedros vedėjas doc. dr. Tomas Blažauskas.

3.1.5. Vartotojai

3.1 lentelė Sistemos vartotojų grupės

Vartotojo kategorija	Moksleiviai
Vartotojo sprendžiami uždaviniai	Konstrukcijos modifikavimas, simuliacijos vykdymas
Patirtis dalykinėje srityje	Naujokas
Patirtis informacinėse technologijose	Patyręs
Papildomos vartotojo charakteristikos	Bazinės anglų kalbos žinios
Vartotojo prioritetai	Antraeiliai vartotojai
Vartotojo kategorija	Studentai
Vartotojo sprendžiami uždaviniai	Roboto konstravimas, valdančios programos kūrimas, roboto apmokymas, simuliacijos valdymas

Patirtis dalykinėje srityje	Naujokas
Patirtis informacinėse technologijose	Patyręs
Papildomos vartotojo charakteristikos	Anglų kalbos žinios
Vartotojo prioritetai	Antraeiliai vartotojai
Vartotojo kategorija	Technologijų mėgėjai
Vartotojo sprendžiami uždaviniai	Roboto konstravimas, valdančios programos kūrimas, roboto apmokymas, simuliacijos valdymas
Patirtis dalykinėje srityje	Įprastas darbuotojas
Patirtis informacinėse technologijose	Patyręs
Papildomos vartotojo charakteristikos	Anglų kalbos žinios
Vartotojo prioritetai	Svarbiausi vartotojai

3.1.6. Įpareigojantys apribojimai

3.1.6.1. Apribojimai sprendimui

- Roboto simuliacija turi būti tikroviška. Tai reiškia sukurtas realus prototipo modelis (panaudojus analogiškas medžiagas bei išmatavimus) įprastomis sąlygomis turi elgtis taip kaip vykdant simuliaciją.
- Darbą sistemoje bet kuriuo metu turi būti galima išsaugoti. Jokiu būdu darbo su sistema progresas negali būti prarastas, išskyrus atvejį kai vartotojas specialiai to nori.
- Galutinis produktas turi turėti užbaigtą pavyzdinį projektą (roboto prototipą). Šis pavyzdys turi parodyti visą sistemos funkcionalumą. Jei taip neišeina tai turi būti tiek pavyzdžių kiek reikia visam funkcionalumui atskleisti.
- Sistema turi leisti kurti atskirai roboto konstrukciją ir atskirai valdančią programą. Tai reiškia kad sukurta valdanti programa gali būti panaudota skirtingom konstrukcijom ir atvirkščiai.
- Atskiras kuriamo roboto projekto dalis galima išsaugoti ir užkrauti atskirai. Jei kuriamas prototipas yra sudaromas iš kelių dalių tai kiekviena dalis turi būti atskiriama ir išsaugotoje laikmenoje.
- Turi būti galimybė roboto valdymui naudoti neroninius tinklus, bei galimybė juos apmokyti. Valdanti programa turi galėti lengvai pasiekti neuroninų tinklų teikiamas funkcijas. Simuliacijos vykdymo metu turi būti numatytos priemonės tiems neuroniniams tinklams apmokyti ir mokymo rezultatą išsaugoti.
- Sukurta sistema turi būti išplečiama (galimybė pridėti naujus konstrukcijos komponentus). Tokie dalykai kaip roboto dalys (jutikliai, motorai, konstrukcinės detalės, ...) ar valdančios programos papildomos bibliotekos turi būti lengvai įskiepijami į galutinę sistemos versiją.
- Vidutinis vartotojas turi išmokti pagrindines programos funkcijas greičiau nei per 4 valandas. Po šios trukmės susipažinimo, vartotojas turi mokėti kurti bazinę roboto konstrukciją, modifikuoti valdančią programą bei sukurti minimalią programą nuo pradžių, taip pat mokėti vykdyti bei valdyti simuliaciją.

3.1.6.2. Diegimo aplinka

Programa bus diegiama asmeniniuose kompiuteriuose, naudojančiuose tiek Windows OS, tiek pagrindinėse Linux OS versijose. Grafikos atvaizdavimui programa turi naudoti grafikos spartintuvą su OpenGL palaikymu. Sistema turi galimybę naudoti bei pateikti duomenys tokiu formatu kurį naudoja kompiuterinės grafikos kūrimo sistema Blender.

3.1.6.3. Bendradarbiaujančios sistemos

Kuriama sistema turi būti nepriklausoma nuo aplinkos, ir ja galima naudotis be jokių išorinių sistemų. Esant poreikiui yra galimybė importuoti duomenis iš populiarios grafikos kūrimo sistemos Blender. Šioje programoje sukurtas figūras bus galima įkelti (kaip vientisą konstrukcijos detalę) konstruojant robotą. Taip pat sukonstruotą robotą bus galima eksportuoti į programai Blender suprantamą formatą.

3.1.6.4. Komerciniai specializuoti programų paketai

- Sistema bus kuriama naudojant QT Framework. Tai yra daugiaplatforminis karkasas, atvaizdavimui naudojantis OpenGL tvarkykles.
- Valdančių programų vykdymui simuliacijos metu bus naudojami QPython ir QScript paketai. Jie puikiai integruojasi į QT karkasą ir atliks JavaScript bei Python interpretatoriaus funkciją.
- Fizikos simuliacijai bus naudojamas ODE (Open Dynamics Engine) paketas, nuo kurio priklausys roboto simuliacijos tikroviškumas.

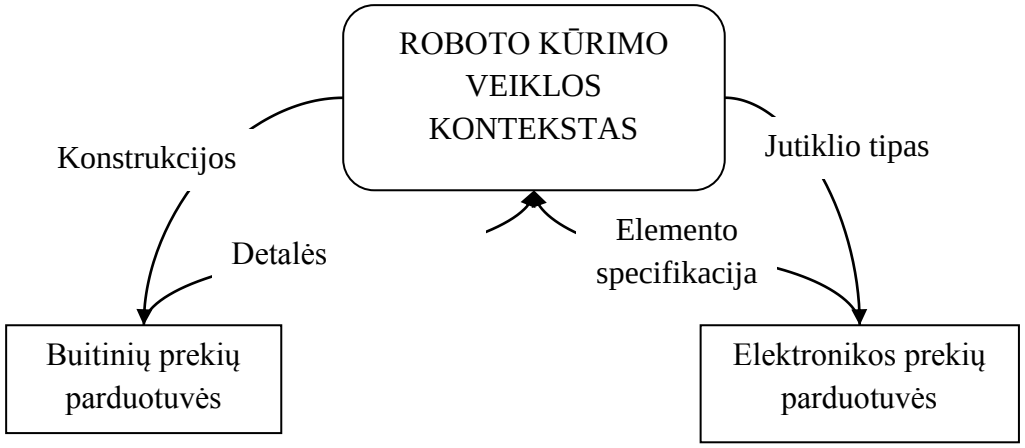
3.1.6.5. Numatoma darbo vietos aplinka

Sistema bus įdiegiama į asmeninę vartotojo darbo vietą. Tai gali būti tiek stacionarus kompiuteris tiek nešiojamas kompiuteris. Sistemos atliekami skaičiavimai turėtų būti optimizuoti kad sumažinti reikalavimus kompiuterio charakteristikoms.

3.1.7. Funkciniai reikalavimai

3.1.7.1. Veiklos kontekstas

Pateiktoje konteksto diagramoje matosi veikloje naudojami informacijos šaltiniai.



3.1 pav. Konteksto diagrama

3.1.7.2. Veiklos padalinimas

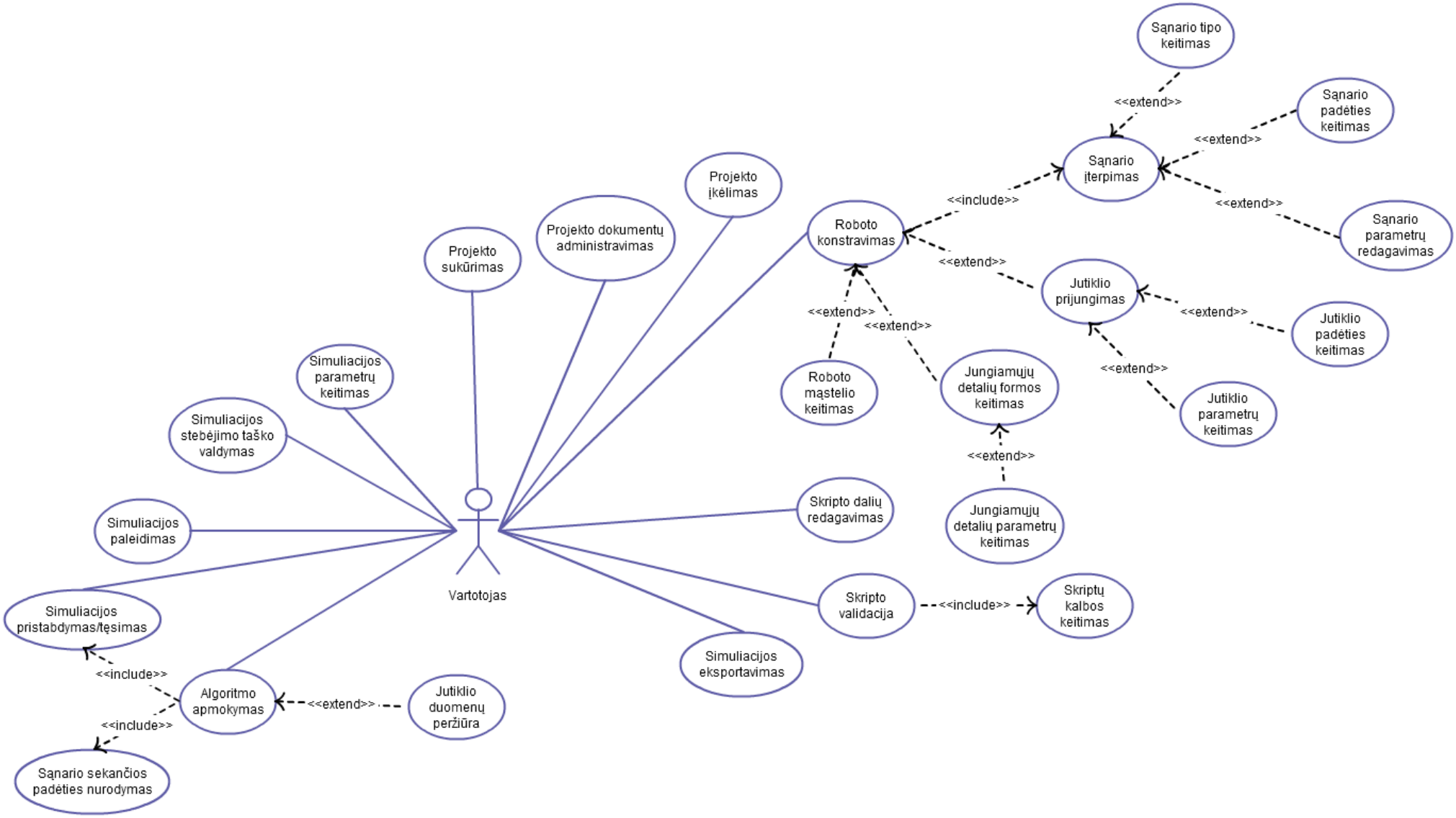
3.2 lentelė Informacijos srautai

Eil.nr.	Įvykio pavadinimas	Įeinantys/išeinantys informacijos srautai
1	Vartotojas renkasi konstrukcijos detalę	Konstrukcijos tipas (out)
2	Buitinių prekių parduotuvė pateikia detalės variantus	Detalių matmenys (in)
3	Vartotojas nori prijungti jutiklį	Elemento tipas (out)
4	Elektronikos prekių parduotuvė pateikia turimų elementų specifikacijas	Elemento specifikacija (in)

3.1.8. Produkto veiklos sfera

3.1.8.1. Sistemos ribos

Kuriama sistema yra skirta asmeniniam kompiuteriui. Ji atlieka įrankių komplekto vaidmenį, todėl ji skirta dirbti vienos rolės vartotojui. Visomis sistemos funkcijomis naudojasi standartinis vartotojas. Sekančiame lape (3.2 pav.) pateikiama sistemos panaudojimo atvejų diagrama. Kadangi yra tik viena vartotojo rolė, tai visi panaudos atvejai yra atliekami standartinio vartotojo. Didžiausia šaka diagramoje yra roboto konstravimo panaudos atvejis. Tai yra kompleksinis panaudos atvejis, į kurį įeina visas roboto konstrukcijos kūrimo etapas.



3.2 pav. Panaudojimo atvejų diagrama

3.1.8.2. Panaudojimo atvejų sąrašas

3.3 lentelė Panaudos atvejai

Eil .nr.	Panaudos atvejis
1	Projekto sukūrimas
2	Projekto dokumentų administravimas
3	Projekto įkėlimas
4	Roboto konstravimas
5	Sąnario įterpimas
6	Sąnario padėties keitimas
7	Sąnario tipo keitimas
8	Sąnario parametrų redagavimas
9	Jungiamųjų detalių formos keitimas
10	Jungiamųjų detalių parametrų keitimas
11	Roboto mastelio keitimas
12	Jutiklio prijungimas
13	Jutiklio padėties keitimas
14	Jutiklio parametrų keitimas
15	Skriptų kalbos keitimas
16	Skripto dalių redagavimas
17	Skripto validacija
18	Simuliacijos parametrų keitimas
19	Simuliacijos paleidimas
20	Simuliacijos pristabdymas/tęsimas
21	Simuliacijos stebėjimo taško valdymas
22	Algoritmo apmokymas
23	Sąnario sekančios padėties nurodymas
24	Jutiklio duomenų peržiūra
25	Simuliacijos eksportavimas

Sistemos panaudojimo atvejai išvardinti 3.3 lentelėje. Panaudojimų atvejams sudaryti pasirinktas toks detalumas kad kiekvieno PA pagrindinį scenarijų sudarytų nemažiau kaip 2 etapai. Panaudojimo atvejų specifikacija pateikta 3.4 lentelėje.

3.4 lentelė Detalizuoti panaudos atvejai

1 PANAUDOJIMO ATVEJIS:	Projekto sukūrimas
Tikslas:	Išskirti dokumentų rinkinį būsimam projektui
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	-
Prieš sąlygos:	Atverta programa
Sužadinimo sąlygos:	Vartotojas spaudžia meniu pasirinkimą
Po sąlygos:	Ekrane matomi tik naujo projekto dokumentai
Pagrindinis scenarijus:	Vartotojas prisijungia prie sistemos

	Pasirenkamas meniu punktas
Alternatyvūs scenarijai:	Vartotojas atsivertęs kitą projektą Pasirenkamas meniu punktas

2 PANAUDOJIMO ATVEJIS:	Projekto dokumentų administravimas
Tikslas:	Pridėti ar panaikinti projekto sudedamąsias dalis
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	-
Prieš sąlygos:	Atvertas (sukurtas) projektas
Sužadinimo sąlygos:	Projekto hierarchijoje aktyvuojama kontekstinio meniu komanda
Po sąlygos:	Anuliuota arba pakeista sudedamoji projekto dalis
Pagrindinis scenarijus:	Atverčiamas projektas Sudedamoji projekto dalis pakeičiama dalimi iš failo

3 PANAUDOJIMO ATVEJIS:	Projekto įkėlimas
Tikslas:	Atverti ankščiau sukurtą projektą
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	-
Prieš sąlygos:	Atverta programa
Sužadinimo sąlygos:	Vartotojas spaudžia meniu pasirinkimą
Po sąlygos:	Atvertas pasirinktas projektas
Pagrindinis scenarijus:	Atidaroma programa Pasirenkama įkelti projektą Iš sąrašo parenkamas norimas projektas
Alternatyvūs scenarijai:	Atidaroma programa Pasirenkama įkelti projektą Iš failų sistemos parenkamas projekto failas

4 PANAUDOJIMO ATVEJIS:	Roboto konstravimas
Tikslas:	Sumodeliuoti roboto prototipą, naudojamą simuliacijoje
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	5 Sąnario įterpimas 6 Sąnario padėties keitimas 7 Sąnario tipo keitimas 8 Sąnario parametrų redagavimas 9 Jungiamųjų detalių formos keitimas 10 Jungiamųjų detalių parametrų keitimas 11 Roboto mastelio keitimas 12 Jutiklio prijungimas 13 Jutiklio padėties keitimas 14 Jutiklio parametrų keitimas

Nefunkciniai reikalavimai:	Trimatė sąsaja OpenGL pagrindu
Prieš sąlygos:	Atvertas (sukurtas) projektas
Sužadinimo sąlygos:	Grafinė trimatė sąsaja perjungiama į konstravimo rėžimą
Po sąlygos:	Turimas sąnarių junginys su (arba be) jutikliais, kuriam bus pritaikomas AI skriptas
Pagrindinis scenarijus:	Sukuriamas projektas Pridedami sąnariai Keičiami sąnarių parametrai Keičiami jungiamųjų dalių parametrai Pridedami jutikliai Nustatomas mastelis

5 PANAUDOJIMO ATVEJIS:	Sąnario įterpimas
Tikslas:	Papildyti konstruojamą robotą
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	6 Sąnario padėties keitimas 7 Sąnario tipo keitimas 8 Sąnario parametrų redagavimas
Prieš sąlygos:	Įjungtas roboto konstravimo rėžimas
Sužadinimo sąlygos:	Grafinėje sąsajoje pažymima sąnario vieta
Po sąlygos:	Virtualioje aplinkoje matomas naujas sąnarys
Pagrindinis scenarijus:	Sukuriamas projektas Įjungiamas konstravimo rėžimas Įjungiamas sąnarių pridėjimo įrankis Pažymima vieta erdvėje

6 PANAUDOJIMO ATVEJIS:	Sąnario padėties keitimas
Tikslas:	Perkelti sąnarį į reikiamą vietą
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	5 Sąnario įterpimas
Prieš sąlygos:	Konstravimo rėžime įjungtas perkėlimo įrankis
Sužadinimo sąlygos:	Pele traukiama viena iš krypties rodyklių
Po sąlygos:	Sąnarys perkeltas iš buvusios pozicijos į naują
Pagrindinis scenarijus:	Konstruojamas robotas Perjungiamas perkėlimo įrankis Pažymimas norimas sąnarys Traukiama viena iš krypties rodyklių

7 PANAUDOJIMO ATVEJIS:	Sąnario tipo keitimas
Tikslas:	Virtualiam sąnariui priskirti reikalingą tipą
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	5 Sąnario įterpimas

Prieš sąlygos:	Konstravimo rėžime pažymėtas sąnarys
Sužadinimo sąlygos:	Kontekstiniame meniu parenkamas sąnario tipas
Po sąlygos:	Sąnariui priskirtas atitinkamas tipas
Pagrindinis scenarijus:	Konstruojamas robotas Pažymimas sąnarys Kontekstiniame meniu parenkama skiltis keisti tipą Kontekstiniame meniu parenkamas sąnario tipas

8 PANAUDOJIMO ATVEJIS:	Sąnario parametrų redagavimas
Tikslas:	Sukalibruoti sąnarį atitinkantį realų modelį
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	5 Sąnario įterpimas
Prieš sąlygos:	Įjungtas konstravimo rėžimas
Sužadinimo sąlygos:	Pažymimas sąnarys
Po sąlygos:	Sąnario parametrai atitinka norimą modelį
Pagrindinis scenarijus:	Konstruojamas robotas Pažymimas sąnarys Parametrų panelėje suvedamos norimos parametrų reikšmės

9 PANAUDOJIMO ATVEJIS:	Jungiamųjų detalių formos keitimas
Tikslas:	Priderinti modelio formą realiam robotui
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	4 Roboto konstravimas 10 Jungiamųjų detalių parametrų keitimas
Prieš sąlygos:	Konstravimo rėžime pažymėta jungiamoji dalis
Sužadinimo sąlygos:	Kontekstiniame meniu parenkamas formos šablonas
Po sąlygos:	Jungiamosios dalies forma panaši į norimą
Pagrindinis scenarijus:	Konstruojamas robotas Pažymima jungiamoji dalis Kontekstiniame meniu parenkamas formos šablonas Redaguojami formos parametrai

10 PANAUDOJIMO ATVEJIS:	Jungiamųjų dalių parametrų keitimas
Tikslas:	Priderinti detalės savybes realiam modeliui
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	9 Jungiamųjų detalių formos keitimas
Prieš sąlygos:	Įjungtas konstravimo režimas
Sužadinimo sąlygos:	Pažymėta jungiamoji dalis
Po sąlygos:	Detalės savybės atitinka norimas
Pagrindinis scenarijus:	Konstruojamas robotas Pažymima detalė Parametrų panelėje pakeičiami parametrai

11 PANAUDOJIMO ATVEJIS:	Roboto mastelio keitimas
Tikslas:	Priderinti virtualų modelį realaus pasaulio dydžiams
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	4 Roboto konstravimas
Nefunkciniai reikalavimai:	Ilgis nurodomas centimetrais
Prieš sąlygos:	Netuščias roboto modelis
Sužadinimo sąlygos:	Pažymėjus vieną iš detalių įvedamas realus ilgis
Po sąlygos:	Roboto mastelis virtualioje erdvėje atitinka realaus modelio mastelį
Pagrindinis scenarijus:	Sukonstruojamas robotas Pažymima žinomo ilgio detalė Įvedamas detalės ilgis

12 PANAUDOJIMO ATVEJIS:	Jutiklio prijungimas
Tikslas:	Pridėti papildomą įvestį AI algoritmui
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	4 Roboto konstravimas 13 Jutiklio padėties keitimas 14 Jutiklio parametrų keitimas
Prieš sąlygos:	Konstravimo rėžime pažymėtas sąnarys
Sužadinimo sąlygos:	Kontekstiniame meniu parenkamas jutiklio tipas
Po sąlygos:	Prie sąnario prilipdytas jutiklis
Pagrindinis scenarijus:	Sukonstruojamas robotas Pažymimas norimas sąnarys Kontekstiniame meniu parenkamas jutiklio tipas
13 PANAUDOJIMO ATVEJIS:	Jutiklio padėties keitimas
Tikslas:	Įstatyti jutiklį į reikiamą korpuso vietą
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	12 Jutiklio pajungimas
Prieš sąlygos:	Jutiklis įterptas ir pažymėtas
Sužadinimo sąlygos:	Parenkamas perkėlimo įrankis
Po sąlygos:	Jutiklis prijungtas norimoje vietoje

14 PANAUDOJIMO ATVEJIS:	Jutiklio parametrų keitimas
Tikslas:	Suderinti jutiklį jam skirtai funkcijai
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	12 Jutiklio pajungimas
Prieš sąlygos:	Jutiklis įterptas ir pažymėtas
Sužadinimo sąlygos:	Pažymėtas elementas aktyvuojamas
Po sąlygos:	Norimas jutiklio parametras pakeistas į naują reikšmę

15 PANAUDOJIMO ATVEJIS:	Skriptų kalbos keitimas
Tikslas:	Perjungti skriptų apdorojimo sistemą į norimos kalbos rėžimą.
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	17 Skripto validacija
Prieš sąlygos:	Atvertas projektas
Sužadinimo sąlygos:	Sužadinamas galimų kalbų sąrašas
Po sąlygos:	Atlikta skripto validacija pagal naujai parinktą kalbą. Vykdant simuliaciją bus naudojama naujai parinkta kalba.

16 PANAUDOJIMO ATVEJIS:	Skripto dalių redagavimas
Tikslas:	Sukurti/pakeisti/pataisyti valdymo skripto fragmentą
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	-
Prieš sąlygos:	Aktyvuotas skripto fragmentas
Sužadinimo sąlygos:	Įvesties įrenginio sužadinimas
Po sąlygos:	Pakeistas skripto tekstas, inicijuota skripto validacija

17 PANAUDOJIMO ATVEJIS:	Skripto validacija
Tikslas:	Informuoti vartotoją apie skripte esančias klaidas
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	15 Skripto kalbos keitimas
Prieš sąlygos:	Aktyvuotas skripto fragmentas
Sužadinimo sąlygos:	Automatiškai, pakitus skripto tekstui
Po sąlygos:	Parodytos skripto klaidos

18 PANAUDOJIMO ATVEJIS:	Simuliacijos parametrų keitimas
Tikslas:	Simuliacijos suderinimas reikiamai problemai tirti
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	-
Prieš sąlygos:	Egzistuoja konstrukcija, validus valdymo skriptas
Sužadinimo sąlygos:	Aktyvuojamas simuliacijos vykdymas
Po sąlygos:	Išsaugoti nauji parametrai, simuliacija paruošta vykdymui

19 PANAUDOJIMO ATVEJIS:	Simuliacijos paleidimas
Tikslas:	Stebėti sukurto prototipo elgesį
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	-
Prieš sąlygos:	Aktyvuotas simuliacijos vykdymo rėžimas
Sužadinimo sąlygos:	Vartotojas sužadina simuliacijos vykdymo komandą
Po sąlygos:	Vyksta (stebima) sukurto roboto realaus laiko simuliacija

20 PANAUDOJIMO ATVEJIS:	Simuliacijos pristabdymas/tęsimas
Tikslas:	Atlikti pakeitimus projekte nenutraukiant simuliacijos
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	22 Algoritmo apmokymas
Prieš sąlygos:	Vykdoma simuliacija
Sužadinimo sąlygos:	Vartotojas sužadina simuliacijos pristabdymo komandą
Po sąlygos:	Simuliacijos progresas sustabdytas ir nekinta.

21 PANAUDOJIMO ATVEJIS:	Simuliacijos stebėjimo taško valdymas
Tikslas:	Apžiūrėti robotą ir jo aplinką kitu rakursu
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	-
Prieš sąlygos:	Aktyvus grafinio redaktoriaus langas
Sužadinimo sąlygos:	Įjungiamas stebėjimo taško redagavimo įrankis
Po sąlygos:	Scena stebima iš naujo stebėjimo taško

22 PANAUDOJIMO ATVEJIS:	Algoritmo apmokymas
Tikslas:	Apibrėžti roboto elgseną apmokant valdymo neuroninius tinklus.
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	20 Simuliacijos pristabdymas/tęsimas Jutiklio duomenų peržiūra
Prieš sąlygos:	Simuliacijos vykdymas pristabdytas
Sužadinimo sąlygos:	Įjungiamas apmokymo rėžimas
Po sąlygos:	Tinklas papildytas naujomis įvesties-išvesties poromis

23 PANAUDOJIMO ATVEJIS:	Sąnario sekančios padėties nurodymas
Tikslas:	Nurodyti sekantį roboto žingsnį esamoje situacijoje
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	22 Algoritmo apmokymas
Prieš sąlygos:	Įjungtas apmokymo rėžimas
Sužadinimo sąlygos:	Pažymimas norimas sąnarys
Po sąlygos:	Neuroninis tinklas papildytas naujomis įvesties-išvesties poromis

24 PANAUDOJIMO ATVEJIS:	Jutiklio duomenų peržiūra
Tikslas:	Stebėti informaciją šiuo simuliacijos metu prieinamą valdančiai programai.
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	22 Algoritmo apmokymas
Prieš sąlygos:	Vykdoma/pristabdyta simuliacija
Sužadinimo sąlygos:	Pažymimas jutiklis
Po sąlygos:	Jutiklio išvestis atvaizduojama ekrane skaitine arba grafiko forma

25 PANAUDOJIMO ATVEJIS:	Simuliacijos eksportavimas
Tikslas:	Išsaugoti simuliacijos vaizdo įrašą
Aktoriai:	Vartotojas
Ryšiai su kitais PA:	-
Prieš sąlygos:	Simuliacija paruoša vykdymui
Sužadinimo sąlygos:	Vietoj simuliacijos vykdymo komandos sužadinama eksportavimo komanda
Po sąlygos:	Laikmenoje išsaugotas simuliacijos vaizdo įrašas

3.1.9. Funkciniai reikalavimai

Detalus funkcinių reikalavimų sąrašas pateiktas prieduose (žr. priedas B). Sistemos funkcionalumui keliami tokie reikalavimai:

- Kiekvienam prototipui kuriamas atskiras projektas
- Atidarant projektą, prieš tai buvęs projektas uždaromas
- Projektas turi turėti sudedamųjų dalių hierarchiją
- Kiekvienas hierarchijos komponentas gali būti įkeliamas atskirai
- Kiekvienas hierarchijos komponentas gali būti išsaugomas atskirai
- Yra galimybė išvalyti ir kurti per naujo atskirą hierarchijos komponentą
- Įkeliant projektą iš failo galima pasirinkti kuriuos komponentus įkelti
- Robotas yra konstruojamas trimatėje erdvėje
- Roboto korpusą sudaro skirtingų tipų sąnariai sujungti standžiomis vienalytėmis detalėmis
- Įterpiant sąnarį jis sujungiamas su prieš tai įterptu sąnariu jei nenurodyta kitaip
- Aktyvuojant sąnario įterpimo įrankį galima pasirinkti sąnario tipą
- Objektų perkėlimo įrankis leidžia pakeisti sąnario padėtį
- Keičiant sąnario padėtį keičiasi ir jungiamosios detalės forma
- Įterpto sąnario tipą galima keisti
- Pakeitus sąnario tipą, analogiški sąnario parametrai persikelia į naują tipą
- Sąnario parametrus galima redaguoti
- Sąnario parametrai esantys parametrų panelėje pilnai nusako sąnarį
- Jungiamųjų detalių formą galima pasirinkti iš numatytų šablonų
- Kiekvienas jungiamosios detalės formos šablonas turi savo parametrų grupę
- Jungiamosios detalės parametrus galima redaguoti
- Yra parametrai kurie yra taikomi visos konstrukcijos savybėms nusakyti
- Konstrukcijos mastelį galima keisti pažymint bet kurią jungiamąją detalę ir įvedant jos realų ilgį (metrais)
- Yra galimybė prijungti jutiklius

- Prijungiant jutiklį reikiamas modelis parenkamas iš sąrašo
- Įterpiant jutiklį reikia nurodyti vietą jungiamosios detalės paviršiuje
- Jutiklio vietą galima keisti
- Keičiant jutiklio vietą reikia nurodyti jungiamosios detalės paviršiaus vietą
- Jutiklio parametrus galima keisti
- Kiekvienas jutiklio tipas gali turėti specialius parametrus
- Valdančios programos aprašo kalbą galima keisti pasirenkant iš sąrašo
- Pakeitus skripto kalbą atliekama skripto validacija
- Valdanti programa susideda iš kelių fragmentų
- Valdančios programos kodą galima keisti net ir simuliacijos vykdymo metu
- Išsaugant valdantį skriptą atliekama validacija
- Validacija atliekama kiekvienam skripto fragmentui atskirai
- Simuliacijos parametrus galima keisti
- Tam tikrą simuliacijos parametrų dalį galima keisti jau vykdomai simuliacijai
- Sukonstruotą robotą galima virtualiai išbandyti realiu laiku, vykdant simuliaciją
- To pačio projekto simuliacija kiekvieną kartą turi būti vienoda
- Vykdomą simuliaciją bet kuriuo metu galima sustabdyti
- Sustabdytą simuliaciją galima tęsti
- Simuliacijos stebėjimo tašką galima keisti
- Valdantis skriptas turi galimybę naudoti neuroninį tinklą
- Turi būti priemonė vartotojui apmokyti valdančios programos neuroninius tinklus
- Apmokant robotą vartotojas sustabdęs simuliaciją nurodo kokią pozą robotas turi stengtis užimti esamoje situacijoje
- Apmokant robotą, apmokymo duomenys yra išsaugojami projekto hierarchijos komponente
- Bet kuriuo simuliacijos metu galima peržiūrėti kiekvieno jutiklio išvesties duomenis
- Jutiklio išvesties informaciją galima matyti grafiko forma
- Simuliacijos vaizdo įrašą galima išsaugoti failų sistemoje
- Simuliaciją galima eksportuoti jos vizualiai nestebint

3.1.10. Nefunkciniai reikalavimai

3.1.10.1. Reikalavimai sistemos išvaizdai

- Vartotojo sąsaja turi rodyti tik aktualią informaciją konkrečiai atliekamiems veiksmams
- Panaudoti išsiskleidžiantys ir susiglaudžiantys skydeliai
- Pagrindinis darbo scenarijus nenaudoja modulinių langų
- Neįkyri sąsaja
- Dviejų tipų išvaizdos: profesionali ir meniška

3.1.10.2. Reikalavimai panaudojamumui

- Trimatis redaktorius intuityviai valdomas pele
- Išdėstymas panašus į standartinį grafikos redaktorių
- Galima išmokti dirbti programa bandymo būdu
- Yra galimybė atšaukti paskutinius veiksmus
- Naudojama vienintelė anglų kalba

3.1.10.3. Reikalavimai vykdymo charakteristikoms

- Simuliacija turi būti vykdoma realiu laiku
- Skaičiavimų tikslumas toks kad vizualiai atrodytų realistiška
- Projekto duomenys diske saugomi nekompresuoti
- Operatyvios atminties panaudojimas neviršija 1GB
- Trimačiam atvaizdavimui panaudotas GPU, o ne CPU
- Išsaugant progresą turi būti paliekama ankstesnio išsaugojimo kopija
- Išplečiama turi būti: sąnarių tipai, jutiklių tipai, jungiamųjų detalių formų šablonai, grafinio redagavimo įrankiai, simuliacijos aplinkos, skriptų interpretatoriaus bibliotekos.

3.1.10.4. Reikalavimai veikimo sąlygoms

Programinė įranga skirta asmeniniam kompiuteriui. Vartotojas dažniausiai dirbs namų sąlygomis. Specialių reikalavimų darbo aplinkai nėra.

3.1.10.5. Reikalavimai sistemos priežiūrai

Tolimesnių sistemos pakeitimų nėra numatoma. Išplėtimo galimybės iš anksto inkorporuojamos į kuriamą sistemą. Sistema iškarto projektuojama tiek Windows tiek Linux operacinėms sistemoms.

3.1.10.6. Reikalavimai saugumui

Kuriama sistema yra asmeniniam naudojimui todėl reikalavimai saugumui nėra griežti.

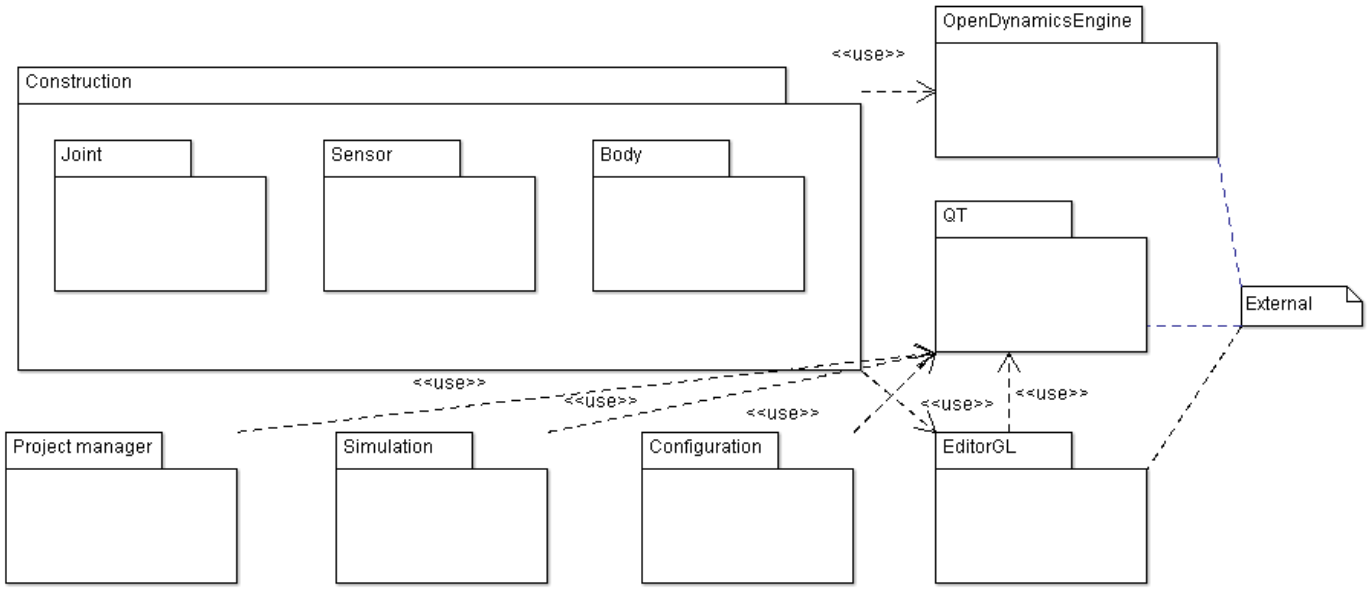
- konfidencialumas – visa saugoma informacija yra vieša ir prieinama bet kuriam kompiuterio vartotojui
- vientisumas – jokia informacija nėra perduodama tinklu, sistema turi užtikrinti kad išsaugoti duomenys bus atveriami tokie kokie buvo išsaugoti
- pasiekiamumas – kadangi viskas vyksta lokaliame kompiuteryje, turi būti užtikrintas 100% pasiekiamumas

3.1.10.7. Teisiniai reikalavimai

Platinant sistemą turi būti pridėti licenzijos failai tiek prie vykdymo failų, tiek prie sistemos naudojamų intelektinių resursų.

3.2. Sistemos statinis vaizdas

3.2.1. Apžvalga

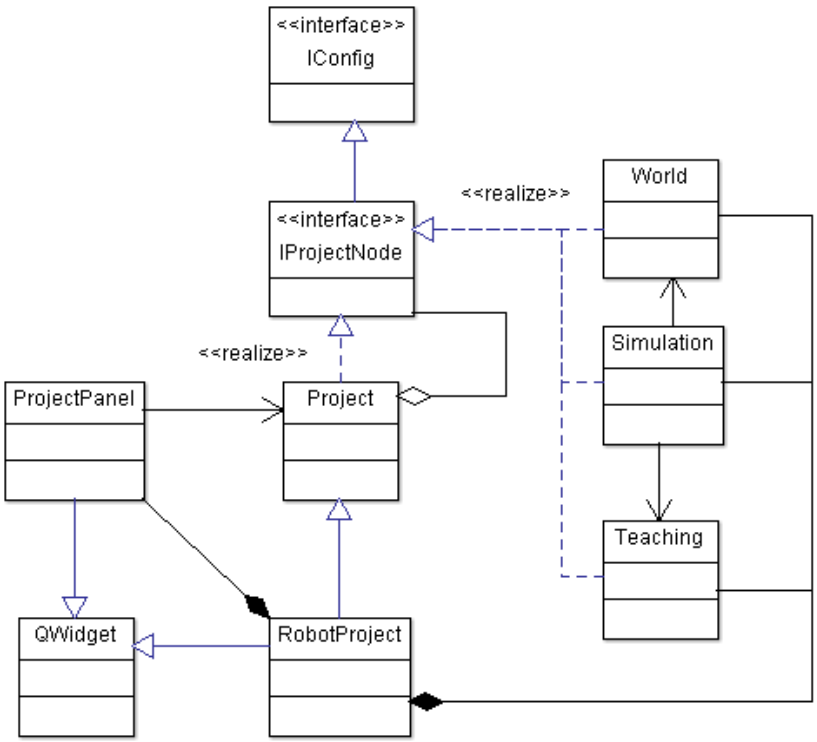


3.3 pav. Paketų diagrama

Paketų diagrama (3.3 pav.) atvaizduoja sistemą sudarančias dalis. Toliau pateikiama kiekvieno paketo supaprastinta klasių diagrama.

3.2.2. Paketų klasių diagramos

Posistemė (3.4 pav.) realizuoja projekto hierarchijos medį.



3.4 pav. Project Manager klasių diagrama

Paketo atsakomybės:

- Hierarchijos medžio atvaizdavimas

- Projekto dalių išvalymas/sukūrimas
- Projekto bei jo komponentų išaugojimas faliluose
- Projekto bei jo komponentų įkėlimas iš failų
- Aktyvaus (pažymėto) komponento sekimas
- Komponento konfigūravimo sąsaja

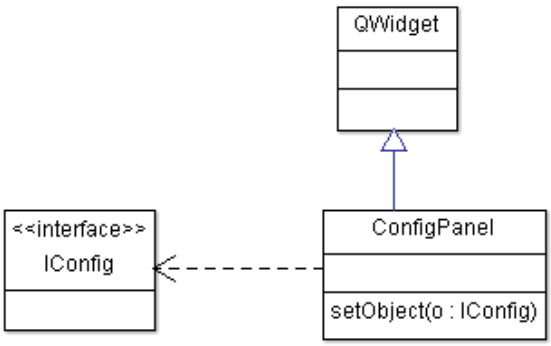
Apribojimai:

- Pats projektas ir jo komponentai traktuojami lygiavertiškai
- Hierarchijos atvaizdavimui naudojamas QT bibliotekos elementas (QWidget)

Positemė naudoja QWidget clasę iš QT bibliotekos. Posistemės sąsajos yra naudojamos kaip pagrindas roboto komponentams (RobotProject, World, Simulation, Teachings). Naudojama failų sistemos peržiūra. Failo įrašymo bei nuskaitymo metu naudojama mažiausiai dvigubai atminties nei užima saugojamas objektas.

Komponentas atvaizduojamas per QWidget sąsają. Visi projekto komponentai turi realizuoti sąsają IProjectNode. Ši sąsaja reikalauja kad komponento objektas būtų serializuojamas, tai yra jį galima pilnai nusakyti vientisu baitų bloku.

Paketas Configuration pateikia sąsają (3.5 pav.) objekto parametrų redagavimui.



3.5 pav. Configuration klasių diagrama

Paketo atsakomybės:

- Atvaizduoti sąrašą objekto redaguojamų parametrų
- Grafinė sąsaja keisti objekto parametrų reikšmėms
- Sklandžiai pereiti nuo vieno objekto parametrų prie kito

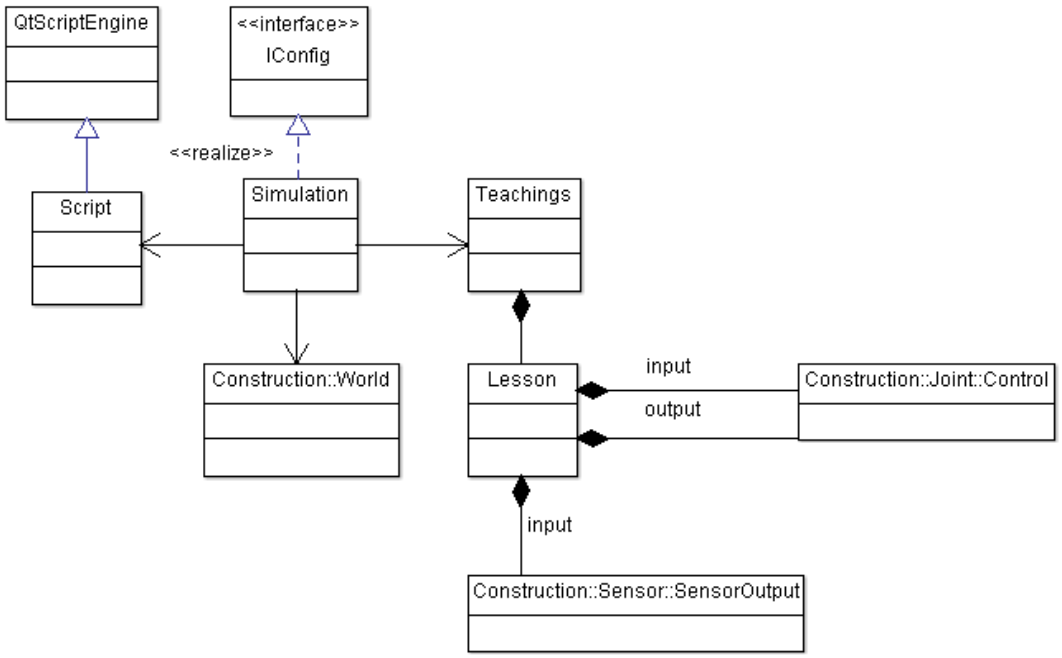
Apribojimai:

- Visi parametrai turėtų būti konvertuojami į tekstą ir atgal
- Parametrų reikšmė išsaugojama kai įvesties laukas praranda fokusavimą

Positemė naudoja QWidget clasę iš QT bibliotekos. Configuracijos sąsają naudoja visi projekto hierarchijos komponentai, fizikines aplinkos klasė (Environment), roboto sąnariai, konstrukcinės detalės bei jutikliai.

Komponentas atvaizduojamas per QWidget sąsają. Bet kuris objektas realizuojantis sąsają IConfig gali būti naudojamas šio įrankio.

Posistemė (3.6 pav.) vykdo simuliacijos iteracijas, interpretuoja valdantį skriptą ir parduoda algoritmo apmokymo duomenis.



3.6 pav. Simulation klasių diagrama

Atsakomybės:

- Sužadina fizikos žingsnio poslinkį
- Validuoja ir vykdo valdančius skriptus
- Išsaugo algoritmo apmokymo duomenis
- Kontroliuoja domenų mainus tarp skripto ir programos objektų

Apribojimai:

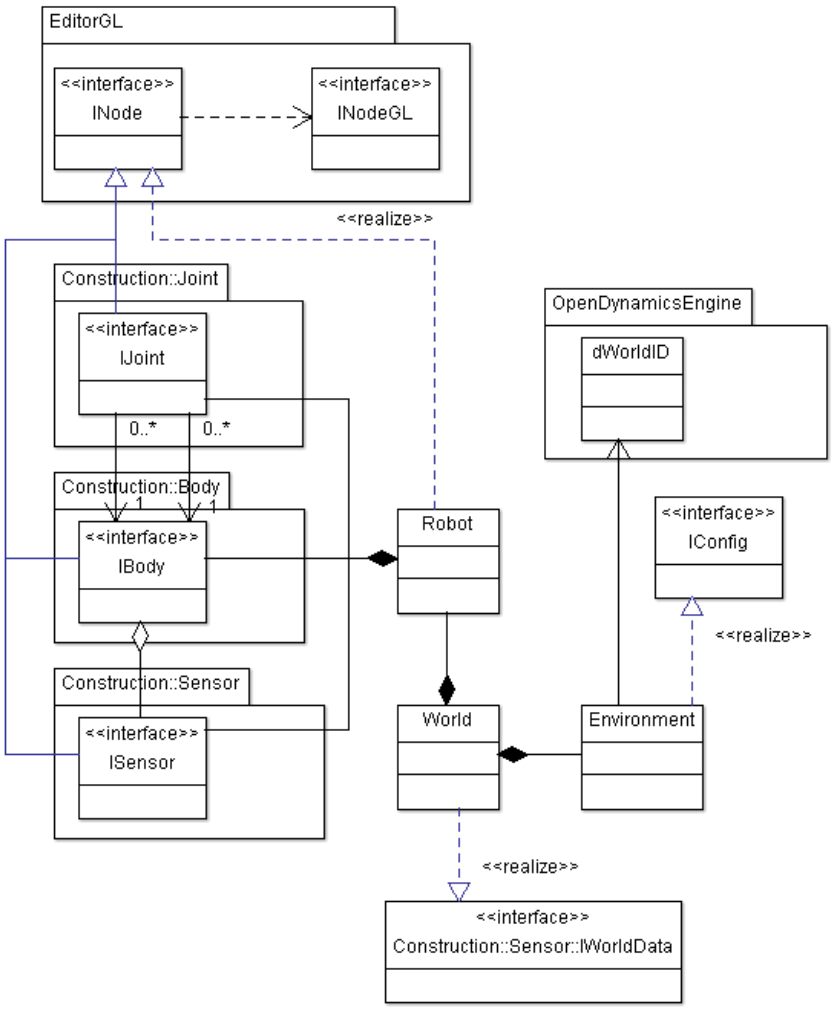
- Simuliacijos objektas yra vienas iš projekto komponentų
- Simuliacijos objekto sukūrimui reikalingi World, bei Teachings projekto komponentai
- Naudojamas vienas skriptų interpretatorius (JavaScript)

Skripto interpretavimui naudojamas QScript komponentas. Ši posistemė surišta su World ir Teachings klasėmis. Naudojama klasė Control iš Sąnario paketo ir klasė SensorOutput iš Sensoriaus paketo.

Posistemė Construction (3.7 pav.) atsakinga už roboto konstravimą bei fizikinių objektų registravimą OpenDynamicsEngine komponente.

Atsakomybės:

- Konstrukcinių detalių pridėjimas/manipuliavimas
- Sąnarių pridėjimas/manipuliavimas
- Jutiklių pridėjimas manipuliavimas
- Fizikinės aplinkos/nustatymų valdymas
- Fizikinių elementų simuliavimas naudojant ODE
- Roboto duomenų adaptavimas jutiklų funkcijoms užtikrinti
- Trimatė vartotojo sąsaja naudojant EditorGL



3.7 pav. Construction klasių diagrama

Apribojimai:

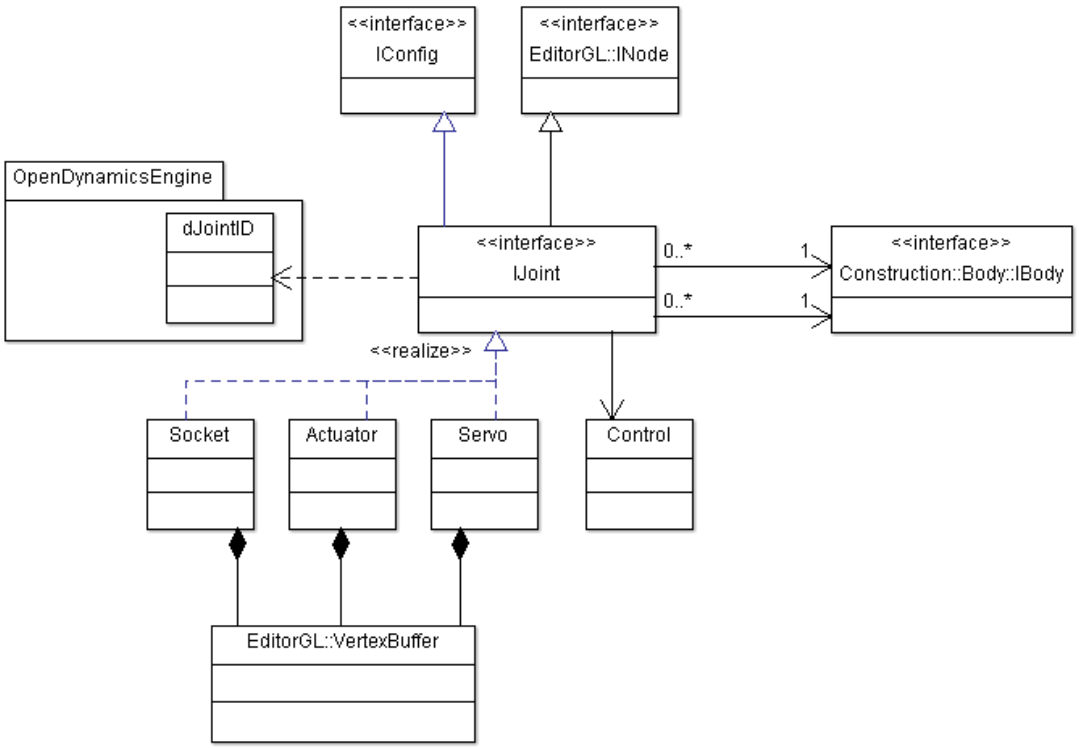
- Detalių tipai priklauso programos kodui, o ne atskiri komponentai
- Tiesioginis pririšimas prie OpenDynamicsEngine komponento
- Su išorę bendraujama per World klasės sąsają

Visa roboto konstrukcija ir aplinkos informacija pasiekiama per klasės World sąsają. Naudojamos ODE komponento funkcijos. Trimatei sąsajai naudojamas EditorGL komponentas. Aplinka konfiguruojama per IConfig sąsają. Visos roboto detalės realizuoja INode sąsają.

Posistemė Joint (3.8 pav.) atsakinga už roboto įvairių tipų sąnarius.

Atsakomybės:

- Sąnario sukūrimas
- Sąnario atvaizdavimas
- Sąnario fizikos simuliavimas naudojant ODE
- Sąsaja sąnariams valdyti



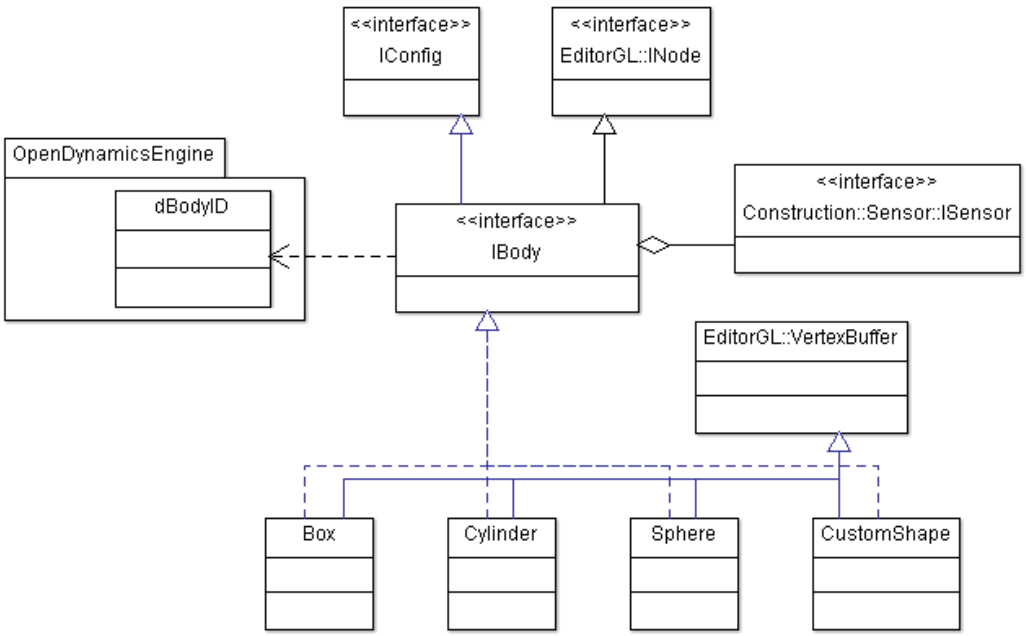
3.8 pav. Joint klasių diagrama

Apribojimai:

- Yra trijų tipų sąnariai (Servo, Actuator, Socket)
- Tiesioginis pririšimas prie OpenDynamicsEngine komponento

Naudoja EditorGL bibliotekos klasę VertexBuffer. Naudoja ODE komponentą. Konfiguruojamas per IConfig sąsają. Sąnarys realizuoja INode sąsają. Priklausoma nuo sąsajos IBody.

Posistemė Body (3.9 pav.) atsakinga už roboto korpuso detales.



3.9 pav. Body klasių diagrama

Atsakomybės:

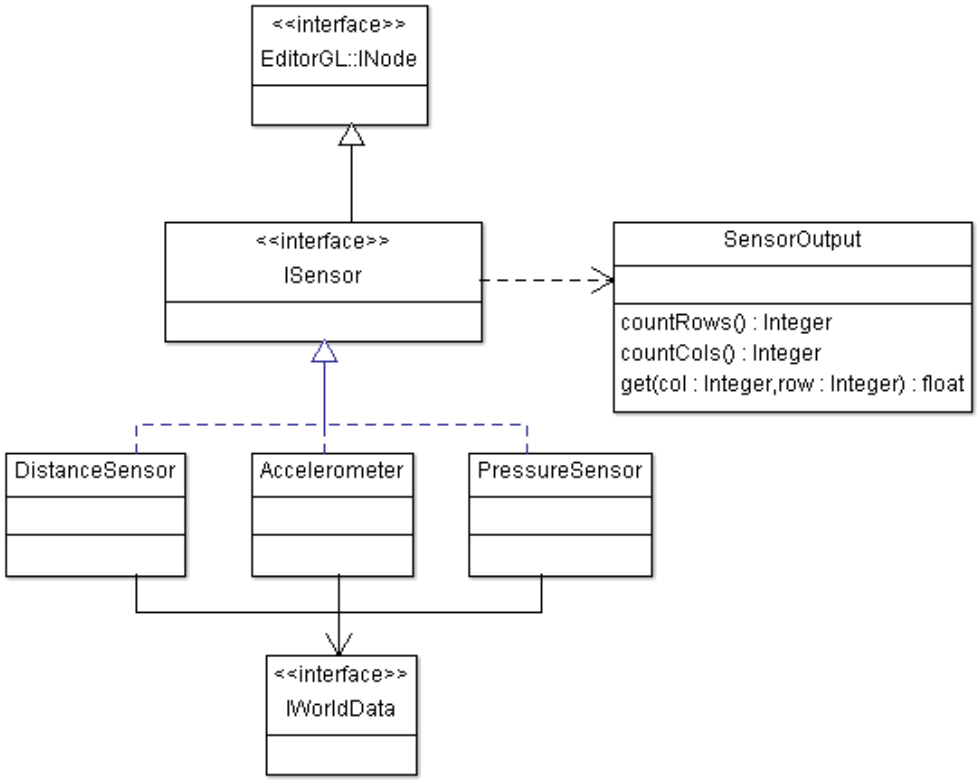
- Korpuso detalės sukūrimas
- Korpuso detalės atvaizdavimas
- Standaus kūno fizikos simuliavimas naudojant ODE

Apribojimai:

- Yra keturių tipų konstrukcijos (Box, Cylinder, Sphere, CustomShape)
- Tiesioginis pririšimas prie OpenDynamicsEngine komponento

Naudoja EditorGL bibliotekos klasę VertexBuffer. Naudoja ODE komponentą. Konfiguruojamas per IConfig sąsają. Sąnarys realizuoja INode sąsają. Priklausomas nuo sąsajos ISensor.

Posistemė Sensor (3.10 pav.) atsakinga už roboto jutiklius.



3.10 pav. Sensor klasių diagrama

Atsakomybės:

- Jutiklių pridėjimas
- Jutiklių atvaizdavimas
- Jutiklio išėjimo duomenų generavimas

Apribojimai:

- Yra trijų tipų jutikliai (DistanceSensor, Accelerometer, PressureSensor)
- Nėra priemonės išėjimo duomenis atvaizduoti grafiškai

Jutiklis realizuoja ISensor, INode sąsajas. Jutikliai išveda duomenis kaip klasės SensorOutput objektus.

4. ROBOTO VALDANČIOS PROGRAMOS REALIZAVIMO TYRIMAS

4.1. Tyrimo kontekstas ir tikslas

Tyrimo metu bus lyginami roboto valdančios programos pateikimo būdai skirtingose sistemose. Lyginama sukurta sistema su jau egzistuojančiu komerciniu sprendimu - *Webots*. Bus parinkti keli roboto variantai ir realizuotos valdančios programos abejoms sistemoms. Siekiant kad palyginimas būtų objektyvus bus laikomasi tokių apribojimų:

- robotus gali sudaryti tik paprastos korpuso detalės, rotaciniai varikliai, servo varikliai, atstumo jutiklis
- robotas yra patalpintas stačiakampio gretasienio formos kambaryje
- simuliacijoje vienu metu dalyvauja tik vienas robotas

Šis tyrimas atliekamas tam kad išsiaiškinti ar sukurtą sistemą galima naudoti kuriant roboto valdančius algoritmus ir kiek valdančios programos realizacija panaši į rinkoje esančias komercines sistemas.

Lyginimui pasirinkta komercinė sistema *Webots* dėl to, kad ji turi laisvai prieinamą pavyzdžių archyvą ir pateikia tokių valdančios programos pavyzdžių, kurie tenkina aukščiau apibrėžtus reikalavimus. Taip pat sistema yra šiuo metu aktyviai palaikoma.

4.2. Tyrimo metodas

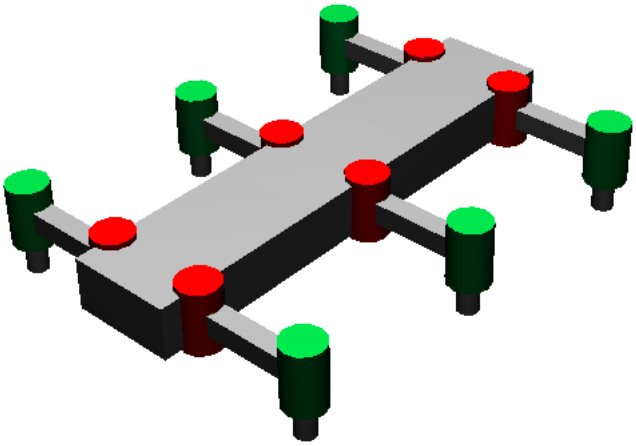
Darbo metu bus tiriami du atvejai. Kiekvienas iš jų bus tiriamas tokia eiga:

1. Apibrėžiama konkreti roboto konstrukcija
2. Apibrėžiamas konkretus roboto elgesys kurį reikia realizuoti
3. Realizuojama grafinė algoritmo reprezentacija pagal *Webots* palaikomą sintaksę
4. Realizuojamas valdančios programos kodas Webots sistemai
5. Realizuojamas valdančios programos kodas naujai sukurtai sistemai
6. Pakeičiamas roboto simuliacijos reikalavimas
7. Pakeičiamos valdančio programos kodo realizacijos abejoms sistemos
8. Įvertinami kiekvienos realizacijos numatyti parametrai
9. Palyginamos realizacijos

4.2.1. Pirmasis atvejis

4.2.1.1. Roboto konstrukcija

Konstrukcija yra šešiakojis robotas. Korpusą sudaro stačiakampė plokštė. Kairėje ir dešinėje pusėje yra po tris kojas. Kiekviena koja prijungta prie servo mechanizmo kurio sukimosi ašis statmena roboto korpusui. Kitame kojos gale yra stūmoklis kurio judėjimo ašis yra priešinga servo variklių sukimosi ašiai. Robotas atvaizduotas 4.1 pav.



4.1 pav. Šešiakojis robotas

4.2.1.2. Roboto elgesys

Robotas turi judėti tiesiai į prieki naudodamas visas šešias kojas. Kojos juda ciklu:

1. koja pasisuka į priekį
2. išskleidžiamas stūmoklis
3. koja pasisuka atgal
4. stūmoklis sutraukiamas

Kiekvienos gretimos kojos fazė skiriasi per 180 laipsnių.

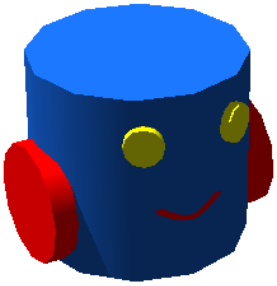
4.2.1.3. Reikalavimo pakeitimas

Roboto priekyje įtaisomas atstumo jutiklis. Robotas juda tiesia linija į priekį tol, kol priartėja prie sienos. Tuomet robotas pakeičia judėjimo kryptį ir juda tiesia linija atgal.

4.2.2. Antrasis atvejis

4.2.2.1. Roboto konstrukcija

Konstrukcija yra cilindras su dviem ratais iš šonų. Sukantis abiem ratams vienodai robotas važiuoja į priekį arba atgal. Sukantis ratams priešingai robotas sukasi į kairę arba į dešinę. Ratus suka motorai su elektroniniu greičio valdikliu. Priekyje įmontuoti du atstumo jutikliai. Tarpas tarp jutiklių 40 laipsnių. Robotas atvaizduotas 4.2 pav.



4.2 pav. Dviratis robotas

4.2.2.2. Roboto elgesys

Robotas turi judėti į priekį išvengdamas kliūčių. Vengimo procedūra tokia:

- jeigu abu jutikliai rodo kliūtį reikia atsitraukiant atgal pasisukti 90 laipsnių į kairę

- jeigu kliūtis tik kairėje/dešinėje pusėje reikia sustoti ir suktis proporcingai pagal atstumą nuo kliūties

4.2.2.3. Reikalavimo pakeitimas

Robotas turi ne du, bet tris ratus. Trečias ratas įtaisytas gale prie servo mechanizmo. Galinis ratas naudojamas vairavimui, o šoniniai ratai varomieji.

4.3. Tyrimo parametrai ir programinė įranga

Valdančios programos realizacija priklauso nuo roboto simuliavimo aplinkos. Lentelėje 4.1 yra palyginami dviejų simuliavimo aplinkų faktoriai įtakojantys valdančios programos realizaciją.

4.1 lentelė Valdančios programos realizacijos savybės

Simuliavimo aplinka / Savybė	Sukurta sistema	Webots
Programavimo kalba	JavaScript	C, VPL
Kodo struktūra	Du fragmentai – pasiruošimas ir ciklo žingsnis	Pilna C programa
Roboto elementų sąsaja	Objektinė sąsaja, objektų sąrašas	Statinės funkcijos

Atsižvelgiant į šias savybes kiekvienam simuliacijos atvejui bus gaunami du JavaScript kodo fragmentai, C programa ir VPL diagrama. Bus vertinami programos kodo parametrai (4.2 lentelė), o VPL diagrama bus kaip palyginimas tarp skirtingų simuliacijos atvejų.

4.2 lentelė Vertinami programos kodo parametrai

Parametras	Įverčio tipas	Prasmė
Eilučių kiekis	Skaičius	Programos sudėtingumas yra proporcingas eilučių skaičiui
Pakeistų eilučių kiekis	Skaičius	Parodo kiek reikia pastangų įvykdyti programos reikalavimo pakeitimą
Pakeisto kodo dalis	Procentas	Parodo kokią įtaką programos kodui turi reikalavimo pasikeitimas

Pakeisto kodo palyginimui naudojamas internetinis įrankis – DiffNow (prieinamas internete http://www.diffnow.com). Šis įrankis parodo:

- pridėtų eilučių kiekį
- ištrintų eilučių kiekį
- pakeistų eilučių kiek

Vertinant pakeistų eilučių kiekį bus sumuojami šie visi trys rodikliai.

4.4. Tyrimo rezultatai

Šiame skyriuje yra pateikiami gauti valdančios programos kodai. Naujai sukurtai sistemai skirtas kodas yra parašytas *JavaScript* programavimo kalba. *Webots* sistemai skirtas kodas turi C programavimo kalbos sintaksę. Taip pat pateikiama *Webots* sistemai tinkama vaizdine programavimo kalba suformuota būsenų diagrama. Kiekvienam simuliacijos atvejui yra pateikiama kodo parametrų suvestinė.

4.4.1. Pirmasis simuliacijos atvejis

4.4.1.1. JavaScript programa

Kadangi roboto judesį galima aprašyti kaip būsenų ciklą, buvo pasinaudota baigtinio automato realizacija (9.3.1 skyrius). Valdymo kodo pasiruošimo dalis pateikta 4.3 pav.

```
function Leg(servo, linearActuator, direction) {
    this.servo = servo;
    this.actuator = linearActuator;
    this.direction = direction;
    this.expand = function() {
        this.actuator.getControl(0).set(0.02);
    };
    this.contract = function() {
        this.actuator.getControl(0).set(-0.02);
    };
    this.spinForward = function() {
        this.servo.getControl(0).set(0.7*this.direction);
    };
    this.spinBackward = function() {
        this.servo.getControl(0).set(-0.7*this.direction);
    };
    this.isExpanded = function() {
        return Math.abs(this.actuator.getControl(0).get() - 0.02) < 0.001;
    }
    this.isContracted = function() {
        return Math.abs(this.actuator.getControl(0).get() - (-0.02)) < 0.001;
    }
    this.isForward = function() {
        return Math.abs(this.servo.getControl(0).get() - (0.7*this.direction)) < 0.01;
    }
    this.isBackward = function() {
        return Math.abs(this.servo.getControl(0).get() - (-0.7*this.direction)) < 0.01;
    }
}
Leg.directions = {
    CLOCKWISE: 1,
    COUNTER_CLOCKWISE: -1
};

function Hexapod(handle) {
    this.prototype = new StateMachine();
    this.prototype.constructor = StateMachine;
    this.constructor = Hexapod;
    this.l1 = new Leg(handle.joints.s1, handle.joints.a1, Leg.directions.CLOCKWISE);
    this.l2 = new Leg(handle.joints.s2, handle.joints.a2, Leg.directions.CLOCKWISE);
    this.l3 = new Leg(handle.joints.s3, handle.joints.a3, Leg.directions.CLOCKWISE);
    this.r1 = new Leg(handle.joints.s4, handle.joints.a4, Leg.directions.COUNTER_CLOCKWISE);
    this.r2 = new Leg(handle.joints.s5, handle.joints.a5, Leg.directions.COUNTER_CLOCKWISE);
    this.r3 = new Leg(handle.joints.s6, handle.joints.a6, Leg.directions.COUNTER_CLOCKWISE);
    this.groupExpanded = [this.l1, this.l3, this.r2];
    this.groupContracted = [this.l2, this.r1, this.r3];

    var extraction = new State(this), contraction = new State(this), swiching = new State(this),
moving = new State(this);
    extraction.enter = function () {
        for (var i=0; i<this._machine.groupContracted.length; i++) {
            this._machine.groupContracted[i].expand();
        }
    }
    contraction.enter = function () {
        for (var i=0; i<this._machine.groupExpanded.length; i++) {
            this._machine.groupExpanded[i].contract();
        }
    }
    swiching.enter = function () {
        var temp = this._machine.groupExpanded;
        this._machine.groupExpanded = this._machine.groupContracted;
        this._machine.groupContracted = temp;
    }
    moving.enter = function () {
        for (var i=0; i<this._machine.groupExpanded.length; i++) {
            this._machine.groupExpanded[i].spinBackward();
        }
        for (var i=0; i<this._machine.groupContracted.length; i++) {
            this._machine.groupContracted[i].spinForward();
        }
    }
```

```
    extraction.addCondition(function(machine) {
        for (var i=0; i<machine.groupContracted.length; i++) {
            if (!machine.groupContracted[i].isExpanded() ) {
                return false;
            }
        }
        return true;
    }, contraction);

    contraction.addCondition(function (machine) {
        for (var i = 0; i < machine.groupExpanded.length; i++) {
            if (!machine.groupExpanded[i].isContracted()) {
                return false;
            }
        }
        return true;
    }, swiching);

    swiching.addCondition(function (machine) { return true; }, moving);

    moving.addCondition(function (machine) {
        for (var i = 0; i < machine.groupExpanded.length; i++) {
            if (!machine.groupExpanded[i].isBackward()) {
                return false;
            }
        }
        for (var i = 0; i < machine.groupContracted.length; i++) {
            if (!machine.groupContracted[i].isForward()) {
                return false;
            }
        }
        return true;
    }, extraction);

    this.start = function () {
        this.setState(extraction);
        extraction.enter();
    }
};
Hexapod.prototype = new StateMachine();

robot = new Hexapod(world.robots[0]);
robot.start();
```

4.3 pav. Pirmos simuliacijos JavaScript kodas - pasiruošimo dalis

Šiuo atveju vykdymo dalis (ciklas) yra triviali – 4.4 pav.

```
// Ciklas
robot.update();
```

4.4 pav. Pirmos simuliacijos JavaScript kodas - vykdymo dalis

4.4.1.2. Webots programa

Kaip atrodo roboto elgesys aprašytas Webots sistemoje yra pateikta 4.5 pav.

```
#include <webots/robot.h>
#include <webots/motor.h>
#include <stdio.h>

#define TIME_STEP 16
#define NUM_MOTORS 12
#define NUM_STATES 6

#define FRONT +0.7
#define BACK  -0.7
#define HI    +0.02
#define LO    -0.02

int main() {
  const char *MOTOR_NAMES[NUM_MOTORS] = {
        "hip_motor_r0",
        "hip_motor_r1",
        "hip_motor_r2",
        "hip_motor_l0",
        "hip_motor_l1",
        "hip_motor_l2",
```

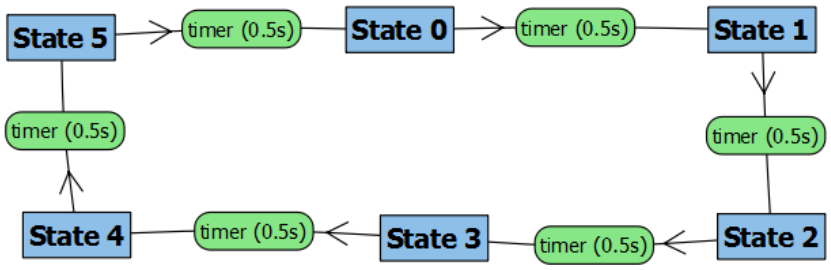
```
        "knee_motor_r0",
        "knee_motor_r1",
        "knee_motor_r2",
        "knee_motor_l0",
        "knee_motor_l1",
        "knee_motor_l2"};
  WbDeviceTag motors[NUM_MOTORS];
  const double pos[NUM_STATES][NUM_MOTORS] = {
        {BACK, FRONT, BACK, -FRONT, -BACK, -FRONT, LO, HI, LO, HI, LO, HI},
        {BACK, FRONT, BACK, -FRONT, -BACK, -FRONT, HI, HI, HI, HI, HI, HI},
        {BACK, FRONT, BACK, -FRONT, -BACK, -FRONT, HI, LO, HI, LO, HI, LO},
        {FRONT, BACK, FRONT, -BACK, -FRONT, -BACK, HI, LO, HI, LO, HI, LO},
        {FRONT, BACK, FRONT, -BACK, -FRONT, -BACK, HI, HI, HI, HI, HI, HI},
        {FRONT, BACK, FRONT, -BACK, -FRONT, -BACK, LO, HI, LO, HI, LO, HI}
  };
  int elapsed = 0;
  int state, i;

  wb_robot_init();

  for (i = 0; i < NUM_MOTORS; i++) {
    motors[i] = wb_robot_get_device(MOTOR_NAMES[i]);
    if (!motors[i]) {
      printf("could not find motor: %s\n",MOTOR_NAMES[i]);
    }
  }

  while(wb_robot_step(TIME_STEP)!=-1) {
    elapsed++;
    state = (elapsed / 25 + 1) % NUM_STATES;
    for (i = 0; i < NUM_MOTORS; i++) {
      wb_motor_set_position(motors[i], pos[state][i]);
    }
  }

  wb_robot_cleanup();

  return 0;
}
```

4.5 pav. Pirmos simuliacijos kodas Webots sistemai

Šioje programoje yra aprašytas kiekvieno variklio būsenų masyvas. Būsenos keičiasi kas nustatytą laiko intervalą.

4.4.1.3. Būsenų diagrama

Kai roboto valdymą galima nustatyti baigtiniu automatu, Webots sistemoje algoritmą galima realizuoti būsenų diagrama (4.6 pav.).



4.6 pav. Pirmosios simuliacijos būsenų diagrama

4.4.2. Pirmojo simuliacijos atvejo modifikacija

4.4.2.1. JavaScript programa

Atliekant pakeitimą programoje (4.7 pav.) reikėjo per naujo aprašyti vieną iš roboto būsenų – kojų perkėlimą. Dėl to teko pakeisti tiek būsenos įėjimo funkciją, tiek būsenos perėjimo sąlygą.

```
function Leg(servo, linearActuator, direction) {
    this.servo = servo;
    this.actuator = linearActuator;
    this.direction = direction;
    this.expand = function () {
```

```
        this.actuator.getControl(0).set(0.02);
    };
    this.contract = function () {
        this.actuator.getControl(0).set(-0.02);
    };
    this.spinForward = function () {
        this.servo.getControl(0).set(0.7 * this.direction);
    };
    this.spinBackward = function () {
        this.servo.getControl(0).set(-0.7 * this.direction);
    };
    this.isExpanded = function () {
        return Math.abs(this.actuator.getControl(0).get() - 0.02) < 0.001;
    }
    this.isContracted = function () {
        return Math.abs(this.actuator.getControl(0).get() - (-0.02)) < 0.001;
    }
    this.isForward = function () {
        return Math.abs(this.servo.getControl(0).get() - (0.7 * this.direction)) < 0.01;
    }
    this.isBackward = function () {
        return Math.abs(this.servo.getControl(0).get() - (-0.7 * this.direction)) < 0.01;
    }
}
Leg.directions = {
    CLOCKWISE: 1,
    COUNTER_CLOCKWISE: -1
};

function Hexapod(handle) {
    this.prototype = new StateMachine();
    this.prototype.constructor = StateMachine;
    this.constructor = Hexapod;
    this.l1 = new Leg(handle.joints.s1, handle.joints.a1, Leg.directions.CLOCKWISE);
    this.l2 = new Leg(handle.joints.s2, handle.joints.a2, Leg.directions.CLOCKWISE);
    this.l3 = new Leg(handle.joints.s3, handle.joints.a3, Leg.directions.CLOCKWISE);
    this.r1 = new Leg(handle.joints.s4, handle.joints.a4, Leg.directions.COUNTER_CLOCKWISE);
    this.r2 = new Leg(handle.joints.s5, handle.joints.a5, Leg.directions.COUNTER_CLOCKWISE);
    this.r3 = new Leg(handle.joints.s6, handle.joints.a6, Leg.directions.COUNTER_CLOCKWISE);
    this.sensor = handle.sensors.ds;
    this.groupExpanded = [this.l1, this.l3, this.r2];
    this.groupContracted = [this.l2, this.r1, this.r3];
    this.direction = 1;

    var extraction = new State(this), contraction = new State(this), swiching = new State(this),
moving = new State(this);
    extraction.enter = function () {
        for (var i = 0; i < this._machine.groupContracted.length; i++) {
            this._machine.groupContracted[i].expand();
        }
    }
    contraction.enter = function () {
        for (var i = 0; i < this._machine.groupExpanded.length; i++) {
            this._machine.groupExpanded[i].contract();
        }
    }
    swiching.enter = function () {
        var temp = this._machine.groupExpanded;
        this._machine.groupExpanded = this._machine.groupContracted;
        this._machine.groupContracted = temp;
    }
    moving.enter = function () {
        if (this._machine.getDirection() > 0) {
            backwardGroup = this._machine.groupExpanded;
            forwardGroup = this._machine.groupContracted;
        } else {
            backwardGroup = this._machine.groupContracted;
            forwardGroup = this._machine.groupExpanded;
        }
        for (var i = 0; i < backwardGroup.length; i++) {
            backwardGroup[i].spinBackward();
        }
        for (var i = 0; i < forwardGroup.length; i++) {
            forwardGroup[i].spinForward();
        }
    }

    extraction.addCondition(function (machine) {
        for (var i = 0; i < machine.groupContracted.length; i++) {
            if (!machine.groupContracted[i].isExpanded()) {
                return false;
```

```
            }
        }
        return true;
    }, contraction);

    contraction.addCondition(function (machine) {
        for (var i = 0; i < machine.groupExpanded.length; i++) {
            if (!machine.groupExpanded[i].isContracted()) {
                return false;
            }
        }
        return true;
    }, swiching);

    swiching.addCondition(function (machine) { return true; }, moving);

    moving.addCondition(function (machine) {
        if (machine.getDirection() > 0) {
            backwardGroup = machine.groupExpanded;
            forwardGroup = machine.groupContracted;
        } else {
            backwardGroup = machine.groupContracted;
            forwardGroup = machine.groupExpanded;
        }
        for (var i = 0; i < backwardGroup.length; i++) {
            if (!backwardGroup[i].isBackward()) {
                return false;
            }
        }
        for (var i = 0; i < forwardGroup.length; i++) {
            if (!forwardGroup[i].isForward()) {
                return false;
            }
        }
        return true;
    }, extraction);

    this.start = function () {
        this.setState(extraction);
        extraction.enter();
    };
    this.isWall = function () {
        return this.sensor.getControl(0).get() < 0.1;
    };
    this.setDirection = function (direction) {
        this.direction = direction;
    };
    this.getDirection = function () {
        return this.direction;
    };
};
Hexapod.prototype = new StateMachine();

robot = new Hexapod(world.robots[0]);
robot.start();
```

4.7 pav. Pirmosios simuliacijos modifikacijos JavaScript kodas – pasiruošimo dalis

Vykdymo dalyje prisidėjo nauja sąlyga kliūties aptikimui tam kad pakeisti roboto judėjimo kryptį.

```
// Ciklas
robot.update();
if (robot.isWall()) {
    robot.setDirection(-1);
}
```

4.8 pav. Pirmosios simuliacijos modifikacijos JavaScript kodas – vykdymo dalis

4.4.2.2. Webots programa

Keičiant šią programą reikėjo būsenų perėjimo funkciją pakeisti taip, kad nuo sąlygos priklausytų būsenos perėjimo kryptis.

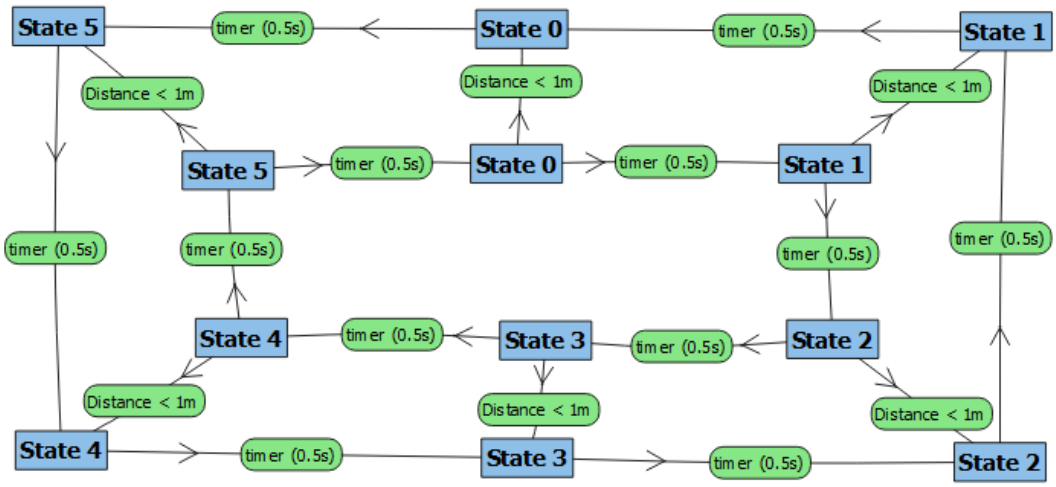
```
#include <webots/robot.h>
#include <webots/motor.h>
#include <stdio.h>

#define TIME_STEP 16
```

```
#define NUM_MOTORS 12
#define NUM_STATES 6

#define FRONT +0.7
#define BACK  -0.7
#define HI    +0.02
#define LO    -0.02

int main() {
  const char *MOTOR_NAMES[NUM_MOTORS] = {
        "hip_motor_r0",
        "hip_motor_r1",
        "hip_motor_r2",
        "hip_motor_l0",
        "hip_motor_l1",
        "hip_motor_l2",
        "knee_motor_r0",
        "knee_motor_r1",
        "knee_motor_r2",
        "knee_motor_l0",
        "knee_motor_l1",
        "knee_motor_l2"};
  WbDeviceTag motors[NUM_MOTORS];
  const double pos[NUM_STATES][NUM_MOTORS] = {
        {BACK, FRONT, BACK, -FRONT, -BACK, -FRONT, LO, HI, LO, HI, LO, HI},
        {BACK, FRONT, BACK, -FRONT, -BACK, -FRONT, HI, HI, HI, HI, HI, HI},
        {BACK, FRONT, BACK, -FRONT, -BACK, -FRONT, HI, LO, HI, LO, HI, LO},
        {FRONT, BACK, FRONT, -BACK, -FRONT, -BACK, HI, LO, HI, LO, HI, LO},
        {FRONT, BACK, FRONT, -BACK, -FRONT, -BACK, HI, HI, HI, HI, HI, HI},
        {FRONT, BACK, FRONT, -BACK, -FRONT, -BACK, LO, HI, LO, HI, LO, HI}
  };
  int elapsed = 0;
  int state, i;
  int direction = 1;
  double ds_value = 0;

  wb_robot_init();
  WbDeviceTag ds = wb_robot_get_device("ds");
  wb_distance_sensor_enable(ds, TIME_STEP);

  for (i = 0; i < NUM_MOTORS; i++) {
    motors[i] = wb_robot_get_device(MOTOR_NAMES[i]);
    if (!motors[i]) {
      printf("could not find motor: %s\n",MOTOR_NAMES[i]);
    }
  }

  while(wb_robot_step(TIME_STEP)!=-1) {
    elapsed++;
      if (elapsed >= 25) {
            elapsed = 0;
            state += direction;
            if(state < 0) {
                  state = NUM_STATES-1;
            } else if (state >= NUM_STATES) {
                  state = 0;
            }
      }
    for (i = 0; i < NUM_MOTORS; i++) {
      wb_motor_set_position(motors[i], pos[state][i]);
    }
      ds_value = wb_distance_sensor_get_value(ds);
      if (ds_value > 500) {
            direction = -1;
      }
  }

  wb_robot_cleanup();

  return 0;
}
```

4.9 pav. Pirmosios simuliacijos modifikacijos kodas Webots sistemai

4.4.2.3. Būsenų diagrama

Būsenų diagrama pasipildė identišku būsenų ciklu tik į perėjimai apkeisti į priešingą pusę. Abu ciklai apjungti perėjimo funkcijomis iš pirmo ciklo į antrąjį.



4.10 pav. Pirmosios simuliacijos modifikacijos būsenų diagrama

4.4.3. Pirmojo simuliacijos atvejo rezultatų rodikliai

4.3 lentelė Pirmo simuliacijos atvejo rezultatai

	Kodo eilučių kiekis	Pakeista eilučių	Pakeista kodo dalis
JavaScript programa	113	37	32.7%
Webots programa	52	17	32.7%

4.4.4. Antrasis simuliacijos atvejis

4.4.4.1. JavaScript programa

Programos kodą (4.11 pav.) sudaro jutiklio aprašymas, variklio aprašymas ir paprastos roboto funkcijos leidžiančios atlikti manevrus ir nustatyti kliūties vietą reliatyviai nuo roboto.

```
function Eye(handle) {
    this.sensor = handle;
    this.isWall = function () {
        return this.getDistance() < 0.8;
    }
    // Distance in meters
    this.getDistance = function () {
        var minV = this.sensor.getParam("min_value"), maxV = this.sensor.getParam("max_value")
        return minV + this.sensor.getControl(0).get() * (maxV - minV);
    }
};
function Wheel(motor, direction) {
    this.handle = motor;
    this.direction = direction;

    this.setSpeed = function (koef) {
        koef = Math.max(-1.0, Math.min(1.0, koef * this.direction));
        var value = (1.0 + koef) / 2;
        this.handle.getControl(0).set(value);
    };
};
Wheel.directions = {
    CLOCKWISE: 1,
    COUNTER_CLOCKWISE: -1
};
function Cylinder(handle) {
    this.lEye = new Eye(handle.sensors.dsL);
    this.rEye = new Eye(handle.sensors.dsR);
    this.lWheel = new Wheel(handle.joints.motor1, Wheel.directions.CLOCKWISE);
    this.rWheel = new Wheel(handle.joints.motor2, Wheel.directions.COUNTER_CLOCKWISE);

    this.goForward = function () {
        this.lWheel.setSpeed(1.0);
        this.rWheel.setSpeed(1.0);
    };

};
```

```
robot = new Cylinder(world.robots[0]);
robot.goForward();
```

4.11 pav. Antrosios simuliacijos JavaScript kodas – pasiruošimo dalis

Vykdymo dalyje (4.12 pav.) realizuotos perėjimo funkcijos tarp roboto būsenų.

```
// Ciklas
if (robot.rEye.isWall()) {
    if (robot.lEye.isWall()) {
        this.lWheel.setSpeed(-1.0);
        this.rWheel.setSpeed(-0.5);
    } else {
        this.lWheel.setSpeed( Math.max(-1, -0.1/robot.rEye.getDistance()) );
        this.rWheel.setSpeed( Math.min(0.7, 0.1 / robot.lEye.getDistance())+0.3 );
    }
} else if (robot.lEye.isWall()) {
    this.lWheel.setSpeed( Math.min(0.7, 0.1 / robot.rEye.getDistance()) + 0.3 );
    this.rWheel.setSpeed( Math.max(-1, -0.1 / robot.lEye.getDistance()) );
} else {
    robot.goForward();
}
```

4.12 pav. Antrosios simuliacijos JavaScript kodas – vykdymo dalis

4.4.4.2. Webots programa

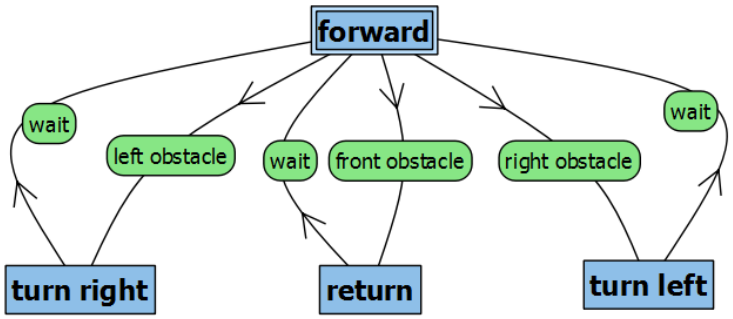
Programa kiekvienos ciklo iteracijos metu tikrina perėjimo funkcijas ir pasikeitus būsenai siunčia atitinkamas komandas roboto varikliams.

```
#include <webots/robot.h>
#include <webots/differential_wheels.h>
#include <webots/distance_sensor.h>

#define SPEED 60
#define TIME_STEP 64

int main()
{
  wb_robot_init(); /* necessary to initialize webots stuff */

  /* Get and enable the distance sensors. */
  WbDeviceTag ds0 = wb_robot_get_device("ds0");
  WbDeviceTag ds1 = wb_robot_get_device("ds1");
  wb_distance_sensor_enable(ds0, TIME_STEP);
  wb_distance_sensor_enable(ds1, TIME_STEP);

  while(wb_robot_step(TIME_STEP)!=-1) {

    /* Get distance sensor values */
    double ds0_value = wb_distance_sensor_get_value(ds0);
    double ds1_value = wb_distance_sensor_get_value(ds1);

    /* Compute the motor speeds */
    double left_speed, right_speed;
    if (ds1_value > 500) {

        /*
         * If both distance sensors are detecting something, this means that
         * we are facing a wall. In this case we need to move backwards.
         */
        if (ds0_value > 500) {
            left_speed = -SPEED;
            right_speed = -SPEED / 2;
        } else {

            /*
             * We turn proportionnaly to the sensors value because the
             * closer we are from the wall, the more we need to turn.
             */
            left_speed = -ds1_value / 10;
            right_speed = (ds0_value / 10) + 5;
        }
    } else if (ds0_value > 500) {
        left_speed = (ds1_value / 10) + 5;
        right_speed = -ds0_value / 10;
    } else {

        /*
         * If nothing has been detected we can move forward at maximal speed.
```

```
         */
        left_speed = SPEED;
        right_speed = SPEED;
    }

    /* Set the motor speeds. */
    wb_differential_wheels_set_speed(left_speed, right_speed);
  }

  return 0;
}
```

4.13 pav. Antrosios simuliacijos Webots kodas

4.4.4.3. Būsenų diagrama



4.14 pav. Antrosios simuliacijos būsenų diagrama

Būsenų diagrama sudaryta iš centrinės būsenos „forward“ ir likusių būsenų, iš kurių visada grįžtama į centrinę būseną po nustatyto laiko intervalo.

4.4.5. Antrojo simuliacijos atvejo modifikacija

4.4.5.1. JavaScript programa

```
function Eye(handle) {
    this.sensor = handle;
    this.isWall = function () {
        return this.getDistance() < 0.8;
    }
    // Distance in meters
    this.getDistance = function () {
        var minV = this.sensor.getParam("min_value"), maxV = this.sensor.getParam("max_value")
        return minV + this.sensor.getControl(0).get() * (maxV - minV);
    }
};
function Wheel(motor, direction) {
    this.handle = motor;
    this.direction = direction;

    this.setSpeed = function (koef) {
        koef = Math.max(-1.0, Math.min(1.0, koef * this.direction));
        var value = (1.0 + koef) / 2;
        this.handle.getControl(0).set(value);
    };
};
Wheel.directions = {
    CLOCKWISE: 1,
    COUNTER_CLOCKWISE: -1
};
function Steer(servo) {
    this.handle = servo;
    this.steerLeft = function (rate) {
        this.handle.getControl(0).set(0.5+rate*0.5);
    }
    this.steerRight = function (rate) {
        this.handle.getControl(0).set(0.5-rate*0.5);
    }
};
function Cylinder(handle) {
    this.lEye = new Eye(handle.sensors.dsL);
    this.rEye = new Eye(handle.sensors.dsR);
    this.lWheel = new Wheel(handle.joints.motor1, Wheel.directions.CLOCKWISE);
    this.rWheel = new Wheel(handle.joints.motor2, Wheel.directions.COUNTER_CLOCKWISE);
```

```
    this.steer = new Steer(handle.joints.steer);

    this.setSpeed = function (speed) {
        this.lWheel.setSpeed(speed);
        this.rWheel.setSpeed(speed);
    };
};
robot = new Cylinder(world.robots[0]);
robot.setSpeed(1);
```

4.15 pav. Antrosios simuliacijos modifikacijos JavaScript kodas – pasiruošimo dalis

```
// Ciklas
if (robot.rEye.isWall()) {
    if (robot.lEye.isWall()) {
        robot.steer.steerRight(0.75);
        robot.setSpeed(-0.5);
    } else {
        robot.steer.steerLeft(Math.min(1, 0.1 / robot.rEye.getDistance()));
        robot.setSpeed(0.8);
    }
} else if (robot.lEye.isWall()) {
    robot.steer.steerRight(Math.min(1, 0.1 / robot.lEye.getDistance()));
    robot.setSpeed(0.8);
} else {
    robot.setSpeed(1);
}
```

4.16 pav. Antrosios simuliacijos modifikacijos JavaScript kodas – vykdymo dalis

4.4.5.2. Webots programa

```
#include <webots/robot.h>
#include <webots/differential_wheels.h>
#include <webots/distance_sensor.h>

#define SPEED 60
#define TIME_STEP 64
#define LEFT +0.7
#define RIGHT -0.7
#define MIDDLE 0

int main()
{
  wb_robot_init(); /* necessary to initialize webots stuff */

  /* Get and enable the distance sensors. */
  WbDeviceTag ds0 = wb_robot_get_device("ds0");
  WbDeviceTag ds1 = wb_robot_get_device("ds1");
  WbDeviceTag steer = wb_robot_get_device("steer");
  wb_distance_sensor_enable(ds0, TIME_STEP);
  wb_distance_sensor_enable(ds1, TIME_STEP);

  while(wb_robot_step(TIME_STEP)!=-1) {

    /* Get distance sensor values */
    double ds0_value = wb_distance_sensor_get_value(ds0);
    double ds1_value = wb_distance_sensor_get_value(ds1);

    /* Compute the motor speeds */
    double steer_angle, speed;
    if (ds1_value > 500) {

        /*
         * If both distance sensors are detecting something, this means that
         * we are facing a wall. In this case we need to move backwards.
         */
        if (ds0_value > 500) {
                  speed = -SPEED;
                  steer_angle = RIGHT; // inversed as it drives backward
        } else {
            /*
             * We turn proportionnaly to the sensors value because the
             * closer we are from the wall, the more we need to turn.
             */
                  speed = SPEED * 0.8;
                  steer_angle = MIDDLE + (LEFT-MIDDLE)*ds1_value/1000;
        }
    } else if (ds0_value > 500) {
        speed = SPEED * 0.8;
            steer_angle = MIDDLE - (MIDDLE-RIGHT)*ds0_value/1000;
    } else {
```

```
        /*
         * If nothing has been detected we can move forward at maximal speed.
         */
        speed = SPEED;
            steer_angle = MIDDLE;
    }

    /* Set the motor speeds. */
    wb_differential_wheels_set_speed(speed, speed);
      wb_motor_set_position(steer, steer_angle);
  }

  return 0;
}
```

4.17 pav. Antrosios simuliacijos modifikacijos Webots kodas

4.4.5.3. Būsenų diagrama

Modifikacijos būsenų diagrama yra lygiai tokia pati kaip originalios simuliacijos (4.14 pav.). Nors perėjimai tarp būsenų yra tokie patys, pačių būsenų realizacijos (roboto įrenginių padėtys) skiriasi.

4.4.6. Antrojo simuliacijos atvejo rezultatų rodikliai

4.4 lentelė Pirmo simuliacijos atvejo rezultatai

	Kodo eilučių kiekis	Pakeista eilučių	Pakeista kodo dalis
JavaScript programa	41	22	53.7%
Webots programa	35	15	42.9%

4.5. Rezultatų įvertinimas

Valdymo kodo parametrų suvestinė pateikta 4.5 lentelėje.

4.5 lentelė Gauti valdymo kodų parametrai

	Kodo eilučių kiekis	Pakeista eilučių	Pakeista kodo dalis
Pirmasis simuliacijos atvejis			
JavaScript programa	113	37	32.7%
Webots programa	52	17	32.7%
Antrasis simuliacijos atvejis			
JavaScript programa	41	22	53.7%
Webots programa	35	15	42.9%

Iš gautų rezultatų matome kad pirmasis atvejis yra sudėtingesnis, nes JavaScript programos kodo eilučių skaičius yra ~2.8 karto didesnis, o Webots – apie 1.5 karto didesnis nei antrojo simuliacijos atvejo.

Abiem atvejais JavaScript programos kodas yra ženkliai ilgesnis nei C programos kodas. Tačiau tai nereiškia kad JavaScript programa yra sudėtingesnė, nes JavaScript yra objektinė programavimo kalba, o C yra procedūrinė programavimo kalba.

Labiau reikšmingas rodiklis yra pakeista programos kodo dalis. Jis leidžia objektyviai palyginti abiejų sistemų valdymo programos lankstumą. Kadangi robotikos dalykinė sritis diktuoja dinamišką valdymo programos kūrimą (paremtą prototipais), tai programos lankstumas yra labai svarbus faktorius lyginant valdančios programos kūrimo sudėtingumą.

Pirmuoju atveju pakeista kodo dalis abejose sistemose buvo tokia pati – 32.7%. Nors JavaScript programoje yra daugiau kodo eilučių, reikalavimų pasikeitimas turėjo tokią pačią įtaką abejų sistemų valdančioms programoms.

Antruoju atveju JavaScript programa nuo reikalavimų pasikeitimo nukentėjo šiek tiek labiau – 53.7%, kai tuo tarpu Webots programa pasikeitė 42.9%. Skirtumas nėra labai žymus, nes šiuo atveju

matavimo paklaida yra gerokai didesnė. Paklaidą lemia sąlyginai mažas absoliutus programos kodo eilučių skaičius.

Įvertinant aprašytus rezultatus galima teigti kad JavaScript programos lankstumas yra labai artimas Webots roboto valdymo programos lankstumui. Jeigu komercinę Webots sistemą galima įsisavinti mokant vaikus programavimo, tai turėtų būti galima įsisavinti ir naujai sukurtą roboto simuliavimo aplinką.

5. DARBAS SU SISTEMA

5.1. Sistemos funkcinis aprašymas

Sistema veikia kaip robotų kūrimo IDE. Programoje yra integruoti tokie įrankiai:

- Projektų valdymo (kūrimas, pildymas, išsaugojimas, užkrovimas)
- Sąnarių konstravimo grafinė sąsaja
- Jutiklių pridėjimo grafinė sąsaja
- Skriptų redaktorius
- Simuliacijos parametrų redaktorius
- Simuliacijos vykdymo (valdymo) grafinė sąsaja
- Algoritmo (neuroninių tinklų) apmokymo grafinė sąsaja
- Simuliacijos eksportavimas

Sistema skirta dirbti vienos rolės vartotojui, tačiau skirtingus projekto etapus gali daryti atitinkamas kvalifikacijas turintys asmenys. Standartinis darbo su sistema scenarijus yra:

1. Projekto sukūrimas
2. Roboto konstravimas
3. Jutiklių prijungimas
4. Algoritmo įvedimas skriptų kalba
5. Simuliacijos parametrų nustatymas
6. Algoritmo apmokymas
7. Simuliacijos vykdymas
8. Simuliacijos eksportavimas

5.2. Vartotojo vadovas

5.2.1. Grafinė sąsaja

Grafinė vartotoja sąsaja yra sudaryta iš kelių dalių ir apima visus įrankius kurie yra naudojami darbo metu. Programos langas yra padalintas per pusę į dvi pagrindines sąsajos dalis. Sąsajos komponentai pažymėti 5.1 pav.

- Kairėje pusėje yra trimatis redaktorius (1)
- Dešinėje pusėje yra projekto hierarchijos redaktorius (2) ir pagrindinis meniu (3)
- Šalia projekto hierarchijos yra projekto nustatymai (4)
- Trimatis redaktorius sudarytas iš pagrindinio lango (1), įrankių juostos (6) ir kontekstinio meniu (5)

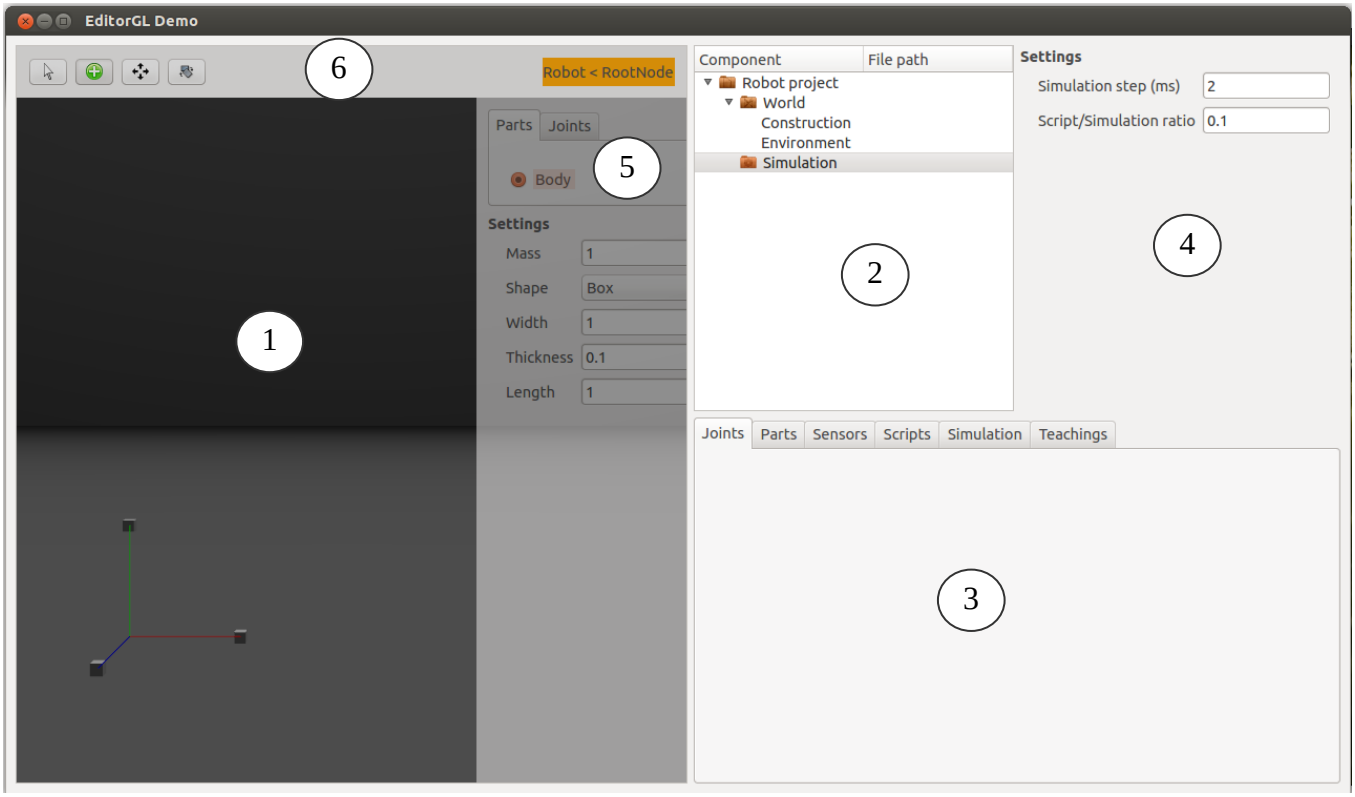
Įrankiu juosta leidžia suaktyvinti vieną iš keturių redagavimo įrankių:

 Pažymėjimo įrankis. Jis leidžia pele pažymėti vieną ar kelias detales. Pažymėta detalė(-ės) gali būti manipuliuojama kitais įrankiais. Taip pat kontekstiniame meniu esantis parametrų redaktorius leidžia keisti pažymėtos detalės parametrus.

 Padėties keitimo įrankis. Jo pagalba galima keisti pažymėtos(-tų) detalės(-ių) padėtį erdvėje X, Y, bei Z ašimis.

 Pavertimo įrankis. Leidžia pasukti pažymėtą detalę norimu kampu apie savo masės centrą.

 Įterpimo įrankis. Jo pagalba pridedamos naujos detalės įskaitant sąnarius, korpusus bei jutiklius. Aktyvavus šį įrankį kontekstiniame meniu atsiveria įterpiamos detalės parametrai. Norimą detalę reikia pasirinkti iš sąrašo pagal kategoriją. Kiekvienas detalės tipas turi skirtingus parametrus. Su tais pačiais nustatymais galima įterpti norimą kiekį detalės kopijų.



5.1 pav. Vartotojo sąsajos komponentai

5.2.2. Darbas su programa

Tik įjungus programą yra sukuriamas tuščias projektas su standartiniais nustatymais. Trimatis redaktorius būna konstravimo būsenoje. Redaktoriaus būsena priklauso nuo pagrindinio meniu pasirinkimo.

Redaktorius yra konstravimo būsenoje kai pagrindiniame meniu yra atverta:

- „Joints“ skiltis
- „Parts“ skiltis
- „Sensors“ skiltis

Redaktorius yra simuliacijos būsenoje kai pagrindiniame meniu yra atverta:

- „Scripts“ skiltis
- „Simulation“ skiltis
- „Teachings“ skiltis

Redaktoriui esant konstravimo būsenoje galimi tokie veiksmai:

- naujų detalių įterpimas
- detalių ištrynimas
- detalių padėties keitimas
- detalių parametrų keitimas

Simuliacijos būsenoje šių veiksmų atlikti neleidžiama, tačiau galima:

- pažymėjus aktyvią detalę, valdyti jos būseną
- grafiškai valdyti roboto detales taip apmokant valdantį algoritmą

Tiek konstravimo, tiek ir simuliacijos būsenoje galima keisti stebėjimo taško padėtį bei kampą. Taip pat galima pasirinkti kurį (vieną iš trijų) stebėjimo tašką naudoti. Tai atliekama klavišo „c“ paspaudimu.

Norint keisti projekto ar tam tikros projekto dalies parametrus reikia projekto hierarchijoje pele pažymėti norimą šaką. Šalia esantis parametrų redaktorius atvaizduos visas galimas tos dalies parinktis. Pavyzdžiui pasirinkus šaką „Simulation“ galima keisti kaip dažnai bus vykdomi fizikos skaičiavimai ir kaip dažnai bus kviečiamas aprašytas valdymo kodas.

Norint išsaugoti projektą ar jo šaką reikia pelės dešiniu klavišu spragtelėti ant norimos šakos. Atsidariusiame konteksto meniu reikia pasirinkti išsaugoti šaką arba įkelti šaką. Tuomet atsivers failų paieškos langas ir prašys jūsų nurodyti norimą failo vietą.

5.2.3. Valdymo klavišai

- „c“ - perjungia naudojamą stebėjimo kamerą į sekančią
- „TAB“ - eina gilyn pažymėtos detalės hierarchijoje
- „Shift“ + „TAB“ - kyla aukštyn šiuo metu atidarytos detalės hierarchijoje
- „Escape“ - atšaukia pradėtą bet neužbaigtą veiksmą, pavyzdžiui detalės paslinkimą
- „MMB“ - judinant pelę stebėjimo kameros vaizdas slenkamas atitinkama kryptimi
- „Shift“ + „MMB“ - judinant pelę keičiamas stebėjimo kampas
- „Ctrl“ + „MMB“ - judinant pelę vaizdas pritraukiamas arba atitolinamas
- „RMB“ - veiksmo klavišas, naudojamas bet kuriai manipuliacijai atlikti ar detalei pažymėti
- „Del“ - panaikina šiuo metu pažymėtą detalę

5.2.4. Detalės pažymėjimas

Detalei pažymėti turi būti suaktyvintas žymėjimo įrankis. Detalė pažymima ant jos užvedus pele ir spragtelėjus dešinį pelės klavišą. Visos kitos pažymėtos detalės atžymimos. Norint išlaikyti pažymėtas detales reikia prieš spragtelėjant pelės klavišu laikyti įspaudus „Shift“ klavišą.

5.2.5. Detalės įterpimas

Kad įterpti korpuso detalę reikia suaktyvinti įterpimo įrankį. Kontekstiniame meniu reikia parinkti norimą detalės tipą ir nustatyti reikiamus parametrus. Nuspaudus dešinį pelės klavišą trimačio redaktoriaus lange pasimato detalės vaizdas. Kol pelės klavišas įspaustas galima keisti detalės padėtį judinant pelę bei atšaukti įterpimą spustelėjus „Escape“ klavišą. Atleidus pelės dešinį klavišą detalė įterpiama.

Norint įterpti sąnarį reikia pirmiausia suaktyvinti žymėjimo įrankį ir pažymėti dvi korpuso detales. Tada suaktyvinti įterpimo įrankį ir pasirinkti norimą detalės tipą. Tokiu būdu įterptas sąnarys sujungs dvi pažymėtas detales. Norint įterpti sąnarį būtina pažymėti dvi korpuso detales, nes kitaip įterpti nepavyks.

5.2.6. Manipuliavimas detalėmis

Manipuliuoti detalėmis galima dviem įrankiais. Padėties keitimo įrankiu ir pavertimo įrankiu. Abu manipuliavimo įrankiai veikia tokiu pačiu principu. Pirmiausia reikia suaktyvinti pažymėjimo įrankį ir pažymėti norimą detalę ar detales. Tuomet reikia suaktyvinti reikiamą manipuliavimo įrankį. Tai padarius redaktoriuje pasirodys toks ženklas:

Matant tokį ženklą galima manipuliuoti pažymėtomis detalėmis. Manipuliuojama spaudžiant dešinį pelės klavišą ir stumiant pelę viena iš parodytų krypčių. Kuo toliau stumsime pelę tuo didesnis detalės pokytis ta ašimi. Padėties keitimo įrankis pastums detalę nurodyta ašimi, o pavertimo įrankis pasuks detalę apie nurodytą ašį prieš laikrodžio rodyklę. Kad būtų aiškiau kokia kryptimi reikia stumti detalę, redaktoriaus kairiajame apatiniame kampe visada rodoma trijų ašių sistema. Ji būna pasukta taip kad atspindėtų erdvės ašių kryptis duotuoju momentu.

5.2.7. Simuliacijos valdymas

Kad pradėti ar sustabdyti simuliaciją, pagrindiniame meniu reikia parinkti skyrių „Simulation“. Tokiu būdu redaktorius persijungia į simuliacijos rėžimą ir pagrindiniame meniu pasimato simuliacijos valdymo mygtukai.

- Klavišas „Play“ pradeda vykdyti simuliaciją realiu laiku. Taip pat šis mygtukas naudojamas pratęsti simuliacijai jeigu ji buvo pristabdyta.
- Klavišas „Pause“ pristabdo simuliaciją jos nenutraukiant. Esant pristabdytai simuliacijai galima įtakoti aktyvių sąnarių būsenas kai tą sunku padaryti vykstant simuliacijai.
- Klavišas „Stop“ sustabdo simuliaciją ir grąžina ją į pradinę būseną. Pradinė būsena yra tokia kokia yra paliekama konstravimo etape.

6. IŠVADOS

1. Išanalizavus esamas roboto simuliavimo aplinkas, nustatyta kad šiuo metu rinkoje yra tiek komercinių, tiek ir atviro kodo robotų simuliavimo aplinkų. Kiekviena turi privalumų tam tikrose srityse, tačiau dažniausiai pasitaikantis trūkumas yra per daug sudėtingas roboto kūrimo procesas arba komplikuota vartotojo sąsaja.
2. Norint supaprastinti roboto kūrimo procesą, pasiūlyta roboto valdymo aprašymui naudoti JavaScript scenarijus. Taip pat siūloma apriboti skirtingų detalių kiekį ir jų configuracijos apimtį.
3. Siekiant išsiaiškinti pasiūlymų tinkamumą buvo suprojektuota ir realizuota elgesiu paremtos robotikos simuliavimo aplinka, pasižyminti vientisa vartotojo sąsaja, nesudėtingu roboto konstravimu ir simuliacijos valdymu.
4. Eksperimentinių tyrimų metu įvertintas sistemos valdančios programos lankstumas naujai sukurtoje sistemoje ir komercinėje sistemoje Webots. Pirmojo eksperimento metu (4.2.1 skyrius) roboto specifikacijos pakeitimas įtakojo 32.7% valdančios programos kodo eilučių pasikeitimą abejose sistemose. Antrojo eksperimento metu (4.2.2 skyrius) pokytis Webots sistemoje buvo 42.9%, o sukurtoje sistemoje – 53.7%. Toks skirtumas nėra didelis, nes paklaidą lemia sąlyginai mažas absoliutus programos kodo eilučių skaičius.
5. Darbo rezultatai paskelbti 19-oje kasmetinėje tarpuniversitetinėje „Informacinė visuomenė ir universitetinės studijos“ konferencijoje.

7. LITERATŪRA

1. Renata Burbaitė. Using Robots as Learning Objects for Teaching Computer Science. X World Conference on Computers in Education, p. 109, 2013.
2. Andreas Birk. Behavior-based Robotics, its scope and its prospects. The 24th Annual Conference of the IEEE Industrial Electronics Society, IECON'98, IEEE Press, p. 1-2, 1998.
3. AAN- Simon O. Haykin. Neural Networks and Learning Machines, McMaster University, Canada, p. 19-27, 2008
4. Dynamics Simulation. Russel Smith, 2004. [žiūrėta 2014-01-22]. Prieiga per internetą: http://www.ode.org/slides/parc/dynamics.pdf
5. Douglas Crockford, "JavaScript: The Good Parts," O'Reilly Media, Inc., California, United States, pp. 2-4, May 2008.
6. ORCA robot simulation and navigation software. [žiūrėta 2014-01-20]. Prieiga per internetą: http://www.ics.forth.gr/~xmpalt/research/orca/
7. Blender For Robotics. [žiūrėta 2014-01-20]. Prieiga per internetą: http://wiki.blender.org/index.php/Community:Science/Robotics
8. Webots: robot simulator. [žiūrėta 2014-01-20]. Prieiga per internetą: http://www.cyberbotics.com/overview
9. Coppelia Robotics v-rep. [žiūrėta 2014-01-20]. Prieiga per internetą: http://coppeliarobotics.com/
10. Robologix. [žiūrėta 2014-01-20]. Prieiga per internetą: https://www.robologix.com/programming_robologix.php

8. PRIEDAI

PRIEDAS A

Informatikos mokymas naudojant robotikos modeliavimo aplinką

Dainius Vaikšnys
Programų inžinerijos katedra
Kauno technologijos universitetas
Kaunas, Lietuva
El. paštas: dainius.vaiksnys@gmail.com

***Santrauka*—Lietuvos mokyklose supažindinimas su programavimu pradedamas nuo Komenskio Logo [1]. Tokiu būdu yra sėkmingai skatinamas kūrybingumas bei susidomėjimas informacinėmis technologijomis. Straipsnyje yra pateikiama alternatyva - virtuali roboto modeliavimo aplinka. Šios sistemos suteikti įgūdžiai turi daugiau praktinio panaudojimo galimybių. Moksleiviai susipažįsta su šiuolaikine scenarijų kalba - JavaScript, bei trimatės grafikos naudojimu informacinėse technologijose. Yra keletas konkuruojančių tokio tipo sistemų, kurių kiekviena turi privalumų tam tikruose panaudojimo atvejuose.**

Reikšminiai žodžiai— virtualūs robotai, informatikos pamokos, realaus laiko modeliavimas.

I. ĮVADAS

Informatikos specialistų paklausa šiandien yra viena didžiausių iš visų specialybių. Lietuvoje yra jaučiamas programuotojų trūkumas. Informatikos studijas baigia nepakankamas kiekis studentų. Reikalingi pokyčiai švietimo sistemoje kad paskatinti moksleivių susidomėjimą programavimu.

Siūlomas sprendimas yra informatikos pamokose naudoti roboto modeliavimo aplinką. Tokiu būdu moksleiviai mokomi programuoti rašant roboto valdančios programos kodą. Tai skatina vaikų susidomėjimą tiek pačia robotikos sritimi, tiek ir programavimu. Toks programavimas yra patrauklesnis, nes yra aiškus atgalinis ryšys ir skatinantis rezultatas.

Šis sprendimas yra kaip palankesnė alternatyva realiai robotikos įrangai nes biudžetinės įstaiga negalėtų sau leisti įsigyti net vienos roboto platformos, nekalbant apie visos klasės paruošimą. Virtuali modeliavimo aplinka išsprendžia ir problemą dėl įrangos sugadinimo, nes mokyklinio amžiaus vaikai ne visada elgiasi atsargiai su svetimu turtu.

Straipsnyje aprašomos egzistuojančios programų sistemos. Taip pat detalizuojamas siūlomas sprendimas. Pagrindinis jo privalumas - tikslinis vartotojas yra moksleivis. Vėliau yra tiriama kokią pedagoginę naudą suteikia tokia sistema. Pateikiamas pamokos scenarijus, aprašoma pavyzdinė užduotis.

II. ROBOTO MODELIAVIMO APLINKA

Roboto kūrimas yra nepaprastai brangus procesas. Sukonstruoti prototipą sunaudojama daug lėšų, o paaiškėjus trūkumams gali tekti jį perdaryti ne vieną kartą. Nemažiau sudėtingas yra algoritmų kūrimas roboto valdymui. Išbandyti valdančią programą su realiu prototipu sudėtinga, nes klaidos gali sukelti tiek smulkių gedimų tiek stipriai apgadinti įrangą.

Visa tai palengvinti yra naudojama programinė įranga modeliuojanti robotą virtualioje erdvėje. Tokia modeliavimo aplinka leidžia sukonstruoti virtualų prototipą, medžiagoms neišleidžiant nei lito. Jis taip pat leidžia apsirašyti valdymo algoritmus ir vykdant modeliavimą stebėti roboto elgseną virtualioje erdvėje. Tokiu būdu nerizikuojant padaryti žalos fiziniam turtui.

A. Roboto modelis

Kuriant virtualų roboto modelį iškyla kelios problemos – detalių formos aproksimacija ir aktyvių (judančių) dalių parametrų atitikimas. Pasyvių detalių forma yra svarbi kontaktui su kitais objektais. Pagrindinis uždavinys yra pateikti tokį konstravimo būdą, kad vartotojas galėtų ir sparčiai sumodeliuoti sugalvotą apytikslį robotą, ir atkartoti griežtai specifikuotas formas iš brėžinių.

Dažniausiai naudojamos robotų aktyvios detalės yra:

- Servo mechanizmai – tai įrenginiai kurie gali sukimo ašį atsukti į tiksliai nurodytą kampą leistiname diapazone.
- Varikliai – jų yra keletas tipų [2], jie gali sukti ašį į vieną arba abi puses, pastoviu greičiu arba pastovia jėga.
- Stūmokliai – atlieka postūmį vienoje ašyje į vieną ar kitą pusę.

B. Jutiklių modeliavimas

Nei vienas robotas neapsieina be jutiklių. Jutikliai perteikia robotui informaciją apie jo aplinką. Yra labai platus spektras robotikoje naudojamų jutiklių [3]. Reikia atrinkti iš visų jutiklių labiausiai reikalingus ir palikti sistemoje galimybę pridėti naujų jutiklių modelių. Jutiklio modelis turi generuoti tam tikro standartizuoto formato duomenis realiai

interpretuodamas aplinką. Šiuos duomenis galės pasiekti vartotojo pateiktas valdymo skriptas.

Kitas uždavinys yra sugalvoti mechanizmą jutiklių pridėjimui prie roboto konstrukcijos. Vartotojas turi galėti paprastai nurodyti jutiklio poziciją bei stebėjimo kampą. Reikalingas jutiklių generuojamų duomenų atvaizdavimo būdas, kad vartotojas galėtų kalibruoti jutiklius, bei susipažinti su jų veikimo principu.

A. *Roboto valdymas*

Bet kurio roboto pagrindas yra valdanti programa. Jos esmė yra valdyti robotą siekiant kažkokio tikslo ar vykdant apibrėžtą funkciją, panaudojant jutiklių generuojamus duomenis. Virtualiam robotui valdyti taip pat reikalinga valdanti programa. Pagrindinis skirtumas yra tas, kad realaus roboto programą riboja techninė skaičiavimo įranga (mikroprocesoriaus architektūra, registrai ir tt.). Yra labai daug skirtingų techninių įrangų su skirtingais kodo kompiliatoriais, kurių programų kodai tarpusavyje nesuderinami. Tuo tarpu modeliuojamas robotas tokių apribojimų neturi ir valdančią programą riboja tik kuriama programinė įranga.

Reikia sugalvoti metodą kaip vartotojo pateikiama valdanti programa įsiterpia į roboto modeliavimą. Turi būti sukurtą sąsaja valdančiai programai gauti jutiklių duomenis. Taip pat reikalinga sąsaja valdančiai programai valdyti aktyvias roboto detales. Kadangi modeliavimas susideda iš daugybės iteracijų laike, tikslinga yra valdančią programą išskaidyti į tris dalis:

- Pasiruošimas (Setup). Čia programa apsirašys reikalingas duomenų struktūras, suteiks reikalingiems kintamiesiems pradines reikšmes.
- Vykdymas (Process). Šioje dalyje aprašytas kodas bus vykdomas kiekvienos iteracijos metu. Programa turės priėjimą prie jutiklių duomenų, ir turės galimybę keisti aktyvių detalių būsenas.
- Apmokymas (Teaching). Ši dalis bus vykdoma kuomet vartotojas modeliavimo metu atliks apmokymą. Tai reiškia pateiks reikiamas roboto detalių būsenas esant dabartiniams jutiklių duomenims bei detalių būsenoms.

Pasiruošimas atskirtas nuo vykdymo dėl to, kad šis kodas vykdomas tik vieną kartą ir nustato pradinę modeliuojamos sistemos būseną. Apmokymui skirta programos dalis nėra tiesiogiai susijusi su roboto elgesiu. Apmokymo kodas yra iškviečiamas asinchroniškai ir nepriklausomai nuo vykdymo ciklo. Ši valdančios programos dalis nėra privaloma ir dažniausiai yra naudojama kai roboto elgesį valdo dirbtiniai neuroniniai tinklai.

Svarbu nuspręsti koks programos kontekstas bus pasiekiamas vartotojo pateiktai valdančiai programai. Įvairiems robotikos uždaviniams reikalingi įvairūs skaičiavimai. Jiems atlikti galima panaudoti egzistuojančias bibliotekas, tokias kaip:

- Matematinės funkcijos
- Skaičių matricų uždaviniai
- Geometriniai skaičiavimai
- Vaizdo apdorojimo funkcijos
- Neuroninių tinklų algoritmai

II. ESAMŲ SPRENDIMŲ ANALIZĖ

Šiame straipsnyje nagrinėjama roboto modeliavimo aplinka yra priskiriama elgesiu paremtos robotikos (behavior-based robotics) kategorijai [7]. Šios kategorijos modeliavimo aplinkų yra sukurta ir pramoniniam lygiui, ir mėgėjiškam vartotojui.

A. *Atviro kodo sprendimai*

- *Gazebo.* Šis sprendimas [8] yra daugelio robotų modeliavimo aplinka lauko salygomis. Jis geba modeliuoti robotų populiaciją, jutiklius ir objektus trimačiame pasaulyje. Modeliavimo aplinka generuoja tiek realistiškus jutiklių parodymus, tiek adekvačias sąveikas tarp objektų (standaus kūno fizikos modeliavimas [9]).
- *LpzRobots.* Tai yra Leipcigo universiteto mokslinių tyrimų projektas [10]. Jį sudaro daug subprojektų, iš kurių svarbiausi yra selforg – karkasas robotų valdymo kūrimui, ir ode_robots – trimatė, fiziškai realistiška roboto modeliavimo aplinka.
- *ORCA Simulator for Robotics.* ORCA modeliavimo aplinka [11] yra galinga platforma robotų modeliavimui, kurią naudoja ne vienas pasaulyje žinomas projektas (GNOSYS, INDIGO, XENIOS, FIRST MM). Sistema palaiko individualias roboto bei aplinkos konfigūracijas. Sistemos principas yra centrinis serveris, prie kurio jungiasi visi moduliai per TCP-IP protokolą.
- *Blender for Robotics.* Blender yra trimatės grafikos kūrimo sistema, kuri turi daugybę modulių ir yra pilnai plečiama. Vienas pagrindinių robotikos komponentų yra Blender Game Engine, skirtas žaidimų kūrimui. Jo pagalba yra vykdomas roboto modeliavimas. Įvairias robotikos puses įgyvendina tam skirti robotikos moduliai [12].

B. *Komerciniai sprendimai*

- *Microsoft Robotics Developer Studio.* Tai yra vienas didžiausių robotikos projektų. Jis apima tiek programinės įrangos kūrimą robotams, tiek ir fizinį roboto konstravimą. Pagrindinis privalumas yra tas kad sukurta valdymo programa yra lengvai perkeliama į fizinį robotą. Be to modeliuojamus jutiklius galima įsigyti kaip ir kitas robotų dalis. Sistema taip pat stipriai supaprastina programos kūrimą pateikdama tokius įrankius kaip kodo generavimas naudojant grafinę sąsają.
- *Webots.* Tai [13] yra kūrimo aplinka skirta programuoti ir modeliuoti mobilius robotus. Vartotojas gali sukurti sudėtingas robotikos konfigūracijas su vienu ar keliais vienodais ar skirtingais robotais bendroje aplinkoje. Vartotojas gali parinkti objektų savybes, tokias kaip forma, spalva, tekstūra, masė, trintis ir panašiai.

- *V-REP*. VREP (virtuali robotų eksperimentavimo platforma) [14] yra naudojama sparčiam algoritmų kūrimui, gamyklos automatizavimo modeliavimui, sparčiam prototipavimui ir verifikacijai, su robotika susijusiam švietimui, saugumo užtikrinimui ir tt. Kiekvienas modelis sistemoje gali būti atskirai valdomas įterptu skriptu, įskiepiu, nuotoliniu API klientu ar kitais būdais.
- *RoboLogix*. Šis sprendimas [15] yra skirtas pramoninių robotų programų testavimui. Su šia sistema galima apmokyti, testuoti, vykdyti, derinti programas parašytas penkių-ašių pramoniniam robotui. Taikymo sritys apima rūšiavimą, produktų sukrovimą, virinimą, dažymą ir tt., leidžiant sugalvoti savo aplinkas ar naujas taikymo sritis.

A. *Siūlomas sprendimas*

Visi minėti sprendimai yra solidžios sistemos, tačiau nei viena iš jų nėra specializuota mokyklinio amžiaus vaikams. Pagrindinės kliūtys trukdančios įsisavinti šiuos sprendimus švietimo sistemoje:

- Roboto detalės yra analogai realių robotikoje naudojamų komponentų. Detalių yra daug ir jos sudėtingos, turi daug parametrų. Neturint patirties robotų kūrime yra sunku išmokti dirbti su tokia sistema.
- Nėra grafinės vartotojo sąsajos arba sistema sudaryta iš kelių dalių, turinčių skirtingas vartotojo sąsajas.

Studijų metu buvo sukurta roboto modeliavimo aplinka tinkama naudoti kaip informatikos mokymo priemonė. Šiai sistemai buvo keliami tokie tikslai:

- teikti bazines robotų prototipų kūrimo funkcijas
- būti prieinama ir moksleiviams – nesunku naudotis sistema
- tapti pagrindiniu įrankiu robotikos mėgėjų veikloje

Sukurta sistema yra įrankių rinkinys turintis bendrą vartotojo sąsają, leidžiančią atlikti visus roboto prototipo kūrimo etapus. Modeliavimo aplinka pasižymi tokiomis savybėmis:

- Kiekvienas prototipo komponentas (konstrukcija, valdymo programa, modeliavimo savybės) gali būti įkeliamos atskirai. Tai reiškia kad norint sukurti valdymo algoritmą nereikia konstruoti savo roboto, galima įsikelti kitų sukonstruotus variantus.
- Modeliavimui naudojamas fizikos variklis Open Dynamics Engine[4]. Tai plačiai naudojamas variklis, užtikrinantis tikslų realaus laiko modeliavimą.
- Valdančiam skriptui aprašyti naudojama JavaScript scenarijų kalba. Ši kalba šiuo metu yra populiariausia [5] programavimo kalba GitHub bendruomenėje. Be to programos kodo nereikia kompiliuoti, todėl jis gali būti keičiamas nenutraukiant modeliavimo.
- Trimatės grafikos redaktorius yra atskiras programos komponentas, todėl redaktorių galima atnaujinti neperkompiliuojant pačios programos. Šis redaktorius yra naudojamas tiek konstruojant robotą, tiek ir peržiūrint modeliavimą.

Didelis siūlomos sistemos privalumas yra daugelio platformų palaikymas. Programa veikia Windows, Linux/X11 ir Mac OS X operacinėse sistemose. Todėl įrengti tinkamas darbo vietas galima ženkliai pigiau, kompiuteriuose diegiant nemokamas operacines sistemas.

II. PEDAGOGINĖ NAUDA

Idėja dėl robotikos panaudojimo mokant moksleivius informatikos nėra nauja. Yra atlikta mokslinių tyrimų analizuojančių galimybes panaudojant robotus kaip mokymo objektus. Straipsnyje [6] iš X World Conference on Computers in Education yra apibendrinama kad toks būdas ne tik padidina mokymo efektyvumą, bet ir suteikia moksleiviams praktinių įgūdžių labai reikalingų šiandieninėje rinkoje.

Roboto modeliavimo aplinkos idėja skiriasi tuo kad nereikalauja didelių lėšų aprūpinant mokyklas robotikos technika. Idėjos realizacija būtų ženkliai pigesnė, nes nereikia nuolatos keisti susidėvėjusios įrangos. Sistemos palaikymas yra centralizuotas ir bet kokie atnaujinimai įsisavinami visose įstaigose. Didžiausia problema būtų naujos švietimo programos parengimas. Siūlomą sistemą galima panaudoti tiek mokant programavimo, tiek fizikos, tiek ir braižybos.

A. *Mokymas programuoti*

Roboto valdymas yra aprašomas JavaScript programavimo kalba. Valdymą galima aprašyti tiek procedūriškai, tiek ir pasitelkiant visas objektinio programavimo galimybes. Galima pasiūlyti kelis pamokos scenarijus priklausomai nuo moksleivių paženginio lygio. Bet kokiu atveju mokytojas turi būti pasiruošęs pilną roboto konstrukciją. Trečios-ketvirtos klasių moksleiviams galimas toks scenarijus:

1. Mokytojas pasiruošia nesudėtingą veikiančią valdymo programą.
2. Moksleiviai stebi modeliavimą ir bando suprasti kaip veikia programa.
3. Pateikiama užduotis šiek tiek pakeisti programą taip kad pasikeistų roboto elgsena.
4. Atsiskaitymas gali būti tiek individualus pademonstravimas, tiek ir programos kodo pateikimas mokytojui.

Tokio mokymo privalumas yra toks kad moksleivis supažindinamas jau su tvarkingai parašytu programos kodu bei teisingomis programavimo praktikomis. Jei mokinys pats pradėtų rašyti savo kodą jis būtu sutelktas į funkcionalumą ir būtų ignoruojamas labai svarbus aspektas - programavimo stilius.

Moksleivis galėtų gauti užduotį pakeisti [programa 1] taip kad robotas galiausiai atsidurtų 10 metrų į šiaurę nuo pradinės padėties. Sunkesnė užduotis būtų papildomas reikalavimas jog roboto nuvažiuotas bendras atstumas nepakistų.

```
var robot = new RiderRobot(world.robots[0]);

robot.drive = function(distance) {
  this.distance.reset();
  if (distance < 0.0000001) {
    return;
  }
  while(this.distance.get() < distance) {
    var speed = 0.8;
    var partLeft = 1-
(this.distance.get()/distance);
    if (partLeft < 0.2) {
      speed = min(0.8, 5*partLeft*partLeft);
    }
    this.motor.setSpeed(speed);
  }
};

for (var i=0; i<3; ++i) {
  robot.rotateTo(0);
  robot.drive(10);
  robot.rotateTo(180);
  robot.drive(10);
}
```

Programa 1. Pavyzdinė valdymo programa

```
function SearchRobot(genericRobot) {

}
var robot=new SearchRobot(world.robots[0]);
var list = [];
for (var i=0; i<360; i+=5) {
  robot.rotateTo(i);
  var objects = robot.findFlags();
  for (var j=0; j<objects.length; j++) {
    var color = objects[j].getColor();
    if (objects[j].getSize() > 0.5 &&
        color.red > 0.9) {
      list.push(objects[j]);
    }
  }
}
console.log(list);
```

Programa 2. Nepilna programa

Vėlesnių klasių moksleiviams gali būti pateikiamas aukšto lygio programos kodas bet be klasių realizacijos [programa 2]. Patiems moksleiviams reikia realizuoti žemesnio lygio funkcijas kad programa veiktų.

Tokiu būdu galima sugalvoti pačių įvairiausių užduočių skirtingo amžiaus moksleiviams. Moksleiviai bus kur kas labiau suinteresuoti įvykdyti užduotis nes yra laukiamas galutinis rezultatas.

JavaScript kalba leidžia mokyti programavimo pradedant tokiomis temomis kaip kintamieji ir jų aritmetika, ciklai, funkcijos, ir pereiti prie sudėtingesnių, objektinio programavimo temų [16], tokių kaip paveldėjimas, enkapsuliacija ar polimorfizmas.

I. IŠVADOS

1. Apžvelgtos roboto modeliavimo aplinkos galimybės švietimo sistemoje. Sprendimas turi pranašumų prieš kitas alternatyvas. Realūs robotikos įrenginiai yra per brangūs išlaikyti globaliu Lietuvos mastu, o tradicinės programavimo priemonės (Logo, Pascal) nesuteikia praktinių įgūdžių šiandieninėje darbo rinkoje.
2. Yra pasiūlyta programinė įranga kuri leidžia konstruoti virtualų robotą ir modeliuoti jo elgseną. Robotą valdanti programa aprašoma JavaScript programavimo kalba. Ši priemonė gali būti naudojama mokyti moksleivius programuoti. Sistemos pedagoginis efektyvumas yra tolesnių mokslinių tyrimų objektas.
3. Pateikti programavimo užduočių pavyzdžiai parodo kaip galima šį sprendimą panaudoti mokant programavimo. Užduotys gali būti sudarytos pasitelkiant bet kurias JavaScript kalbos konstrukcijas. Mokytojas gali pasiruošti pilnai veikiančią programą ir paprašyti pakeisti programą taip, kad gauti skirtingą rezultatą. Taip pat galima sudaryti tik aukšto lygio logikos realizaciją, paliekant moksleiviams patiems aprašyti trūkstamas funkcijas ar klases.

LITERATŪRA

[1] Logo.lt – Vėžliuko namai Lietuvoje. [žiūrėta 2014-01-24]. Prieiga per internetą: http://www.logo.lt/komenskio.htm

[2] Types of Electric Motors. [žiūrėta 2014-01-22]. Prieiga per internetą: http://www.ece.uah.edu/courses/material/EE410-Wms2/Electric%20motors.pdf

[3] Types of Robot Sensors. [žiūrėta 2014-01-22]. Prieiga per internetą: http://www.robotplatform.com/knowledge/sensors/types_of_robot_sensors.html

[4] Dynamics Simulation. Russel Smith, 2004. [žiūrėta 2014-01-22]. Prieiga per internetą: http://www.ode.org/slides/parc/dynamics.pdf

[5] The RedMonk Programming Language Rankings. Stephen O'Grady, 2014. [žiūrėta 2014-01-26]. Prieiga per internetą: http://redmonk.com/sogrady/2014/01/22/language-rankings-1-14/

[6] Renata Burbaitė. Using Robots as Learning Objects for Teaching Computer Science. X World Conference on Computers in Education, 2013.

[7] Ronald C. Arkin. Behavior-Based Robotics. Massachusetts Institute of Technology, 1998.

[8] Overview – Gazebo Wiki. [žiūrėta 2014-01-18]. Prieiga per internetą: http://gazebosim.org/wiki/Overview

[9] Rigid Body Collisions. [žiūrėta 2014-01-21]. Prieiga per internetą: http://www.myphysicslab.com/collision.html

[10] Research Network for Self-Organization of Robot Behavior. [žiūrėta 2014-01-20]. Prieiga per internetą: http://robot.informatik.uni-leipzig.de/software/?lang=en

[11] ORCA robot simulation and navigation software. [žiūrėta 2014-01-20]. Prieiga per internetą: http://www.ics.forth.gr/~xmpalt/research/orca/

[12] Blender For Robotics. [žiūrėta 2014-01-20]. Prieiga per internetą: http://wiki.blender.org/index.php/Community:Science/Robotics

[13] Webots: robot simulator. [žiūrėta 2014-01-20]. Prieiga per internetą: http://www.cyberbotics.com/overview

[14] Coppelia Robotics v-rep. [žiūrėta 2014-01-20]. Prieiga per internetą: http://coppeliarobotics.com/

[15] Robologix. [žiūrėta 2014-01-20]. Prieiga per internetą: http://www.robologix.com/

[16] Douglas Crockford, "JavaScript: The Good Parts," O'Reilly Media, Inc., California, United States, pp. 2-4, May 2008.

PRIEDAS B

Lentelė A1 Funkcinių reikalavimų aprašas

Reikalavimas #:	1	Reikalavimo tipas:	Panaudojimo atvejis #:	1
Aprašymas:	Kiekvienam prototipui kuriamas atskiras projektas			
Pagrindimas:	Prototipą sudaro daug dalių, kurios turi būti apjungtos į projektą			
Šaltinis	Analitikas			
Tikimo kriterijus:	Kiekvienas veiksmas gali būti atliekamas tik kažkuriam vienam projektui, kuris yra saugomas failų laikmenoje kaip atskiras loginis vienetas.			
Užsakovo tenkinimas:	1		Užsakovo nepatenkinimas:	5
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	2	Reikalavimo tipas:	Panaudojimo atvejis #:	1
Aprašymas:	Atidarant projektą, prieš tai buvęs projektas uždaromas			
Pagrindimas:	Kadangi projektą sudaro keletas dalių, vartotojas gali susimaišyti jei vienu metu bus atverti keli projektai			
Šaltinis	Analitikas			
Tikimo kriterijus:	Vienu metu gali būti atidarytas tik vienas projektas			
Užsakovo tenkinimas:	2		Užsakovo nepatenkinimas:	4
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	3	Reikalavimo tipas:	Panaudojimo atvejis #:	2
Aprašymas:	Projektas turi turėti sudedamųkų dalių hierarchiją			
Pagrindimas:	Atskiros projekto dalys gali buti individualiai pakeičiamos			
Šaltinis	Analitikas			
Tikimo kriterijus:	Vartotojas mato projekto hierarchijos medį			
Užsakovo tenkinimas:	2		Užsakovo nepatenkinimas:	4
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	4	Reikalavimo tipas:	Panaudojimo atvejis #:	2
Aprašymas:	Kiekvienas hierarchijos komponentas gali būti įkeliamas atskirai			
Pagrindimas:	Ta pati prototipo dalis gali būti panaudota skirtinguose robotų modeliuose			
Šaltinis	Analitikas			
Tikimo kriterijus:	Projekto hierarchijos komponentai turi pasirinktį įkelti iš failo			
Užsakovo tenkinimas:	3		Užsakovo nepatenkinimas:	2
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	5	Reikalavimo tipas:	Panaudojimo atvejis #:	2
Aprašymas:	Kiekvienas hierarchijos komponentas gali būti išsaugomas atskirai			
Pagrindimas:	Vartotojas tam tikrą prototipo dalį gali pasidalinti su kitais vartotojais			
Šaltinis	Analitikas			
Tikimo kriterijus:	Projekto hierarchijos komponentai turi pasirinktį „Saugoti kaip...“			
Užsakovo tenkinimas:	3		Užsakovo nepatenkinimas:	2
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	6	Reikalavimo tipas:	Panaudojimo atvejis #:	2
Aprašymas:	Yra galimybė išvalyti ir kurti per naujo atskirą hierarchijos komponentą			
Pagrindimas:	Kitu atveju vartotojui reiktu perkurti visą projektą jei jam nepatiko kuri nors viena dalis			
Šaltinis	Analitikas			
Tikimo kriterijus:	Projekto hierarchijos komponentai turi pasirinktį „išvalyti“			
Užsakovo tenkinimas:	2		Užsakovo nepatenkinimas:	2
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	7	Reikalavimo tipas:	Panaudojimo atvejis #:	3
Aprašymas:	Įkeliant projektą iš failo galima pasirinkti kuriuos komponentus įkelti			
Pagrindimas:	Sutaupoma laiko, nes nereikia įkėlinėti kiekvienos dalies atskirai			
Šaltinis	Analitikas			
Tikimo kriterijus:	Jei pasirenkama įkelti visą projektą, prieš tai buvęs projektas uždaromas. Jei pažymimi atskiri komponentai – projektai suliejami (pažymėtos dalys įkeliamos ant viršaus esamo projekto atitinkamų dalių)			
Užsakovo tenkinimas:	3		Užsakovo nepatenkinimas:	2
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	8	Reikalavimo tipas:	Panaudojimo atvejis #:	4
Aprašymas:	Robotas yra konstruojamas trimatėje erdvėje			
Pagrindimas:	Vartotojui nereikia papildomų programų roboto konstravimui			
Šaltinis	Analitikas			
Tikimo kriterijus:	Yra roboto konstravimo rėžimas, kuriame yra manipuliuojama trimatėje erdvėje			
Užsakovo tenkinimas:	3		Užsakovo nepatenkinimas:	4
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	9	Reikalavimo tipas:	Panaudojimo atvejis #:	4
Aprašymas:	Roboto korpusą sudaro skirtingų tipų sąnariai sujungti standžiomis vienalytėmis detalėmis.			
Pagrindimas:	Simuliuojama realaus roboto konstrukcija taikant mažiausią įtaką darančius apribojimus.			
Šaltinis	Analitikas			
Tikimo kriterijus:	Sąnariai gali kelių tipų, atitinkančių realių robotų detales. Kiekvienas sąnarys yra sujungtas bent su vienu kitu sąnariu vientisa jungiamąja detale.			
Užsakovo tenkinimas:	3		Užsakovo nepatenkinimas:	5
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	10	Reikalavimo tipas:	Panaudojimo atvejis #:	5
Aprašymas:	Įterpiant sąnarį jis sujungiamas su prieš tai įterptu sąnariu jei nenurodyta kitaip.			
Pagrindimas:	Vartotojas gali greičiau modeliuoti vientisą korpusą.			
Šaltinis	Analitikas			
Tikimo kriterijus:	Pridėjus naują sąnarį, įterpiama jungiamoji detalė su numatytais parametrais. Šios detalės galai sujungti su įterptu sąnariu bei prieš tai buvusiu sąnariu.			
Užsakovo tenkinimas:	4		Užsakovo nepatenkinimas:	4
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	11	Reikalavimo tipas:	Panaudojimo atvejis #:	5
Aprašymas:	Aktyvuojant sanario įterpimo įrankį galima pasirinkti sąnario tipą.			
Pagrindimas:	Sąnariai gali būti skirtingų tipų			
Šaltinis	Analitikas			
Tikimo kriterijus:	Sąnarių įterpimo įrankis turi parametrą nurodantį sąnario tipą. Įterpiami sąnariai atitinka parametruose nurodytą tipą.			
Užsakovo tenkinimas:	2		Užsakovo nepatenkinimas:	5
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	12	Reikalavimo tipas:	Panaudojimo atvejis #:	6
Aprašymas:	Objektų perkėlimo įrankis leidžia pakeisti sąnario padėtį.			
Pagrindimas:	Būtina konstravimo priemonė.			
Šaltinis	Analitikas			
Tikimo kriterijus:	Aktyvavus perkėlimo įrankį, sąnario poziciją galima keisti tiek su pele, tiek naudojant parametrų skydelį.			
Užsakovo tenkinimas:	2		Užsakovo nepatenkinimas:	5
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	13	Reikalavimo tipas:	Panaudojimo atvejis #:	6
Aprašymas:	Keičiant sąnario padėtį keičiasi ir jungiamosios detalės forma.			
Pagrindimas:	Vartotojui nereikia dvigubo darbo keičiant kartu ir jungiamosios detalės.			
Šaltinis	Analitikas			
Tikimo kriterijus:	Pakeitus sąnario padėtį, jungiamoji detalė deformuojasi taip kad abu jos galai liestųsi su atitinkamais sąnariais			
Užsakovo tenkinimas:	2		Užsakovo nepatenkinimas:	5
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	14	Reikalavimo tipas:	Panaudojimo atvejis #:	7
Aprašymas:	Įterpto sąnario tipą galima keisti			
Pagrindimas:	Konstruojamo roboto reikalavimai gali keistis			
Šaltinis	Analitikas			
Tikimo kriterijus:	Įterpto sąnario tipą galima keisti bet kuriuo metu			
Užsakovo tenkinimas:	3		Užsakovo nepatenkinimas:	4
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	15	Reikalavimo tipas:	Panaudojimo atvejis #:	7
Aprašymas:	Pakeitus sąnario tipą, analogiški sąnario parametrai persikelia į naują tipą			
Pagrindimas:	Sutaupo vartotojo laiką suvedinėjant parametrus			
Šaltinis	Analitikas			
Tikimo kriterijus:	Naujo tipo sąnario parametrai kurie yra buvusio tipo analogai gauna reikšmes iš buvusio tipo. Nauji parametrai įgauna numatytasias reikšmes.			
Užsakovo tenkinimas:	4		Užsakovo nepatenkinimas:	2
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	16	Reikalavimo tipas:	Panaudojimo atvejis #:	8
Aprašymas:	Sąnario parametrus galima redaguoti			
Pagrindimas:	Kiekvienas sąnarys gali turėti unikalius parametrus			
Šaltinis	Analitikas			
Tikimo kriterijus:	Sąnario parametrus galima bet kuriuo metu redaguoti			
Užsakovo tenkinimas:	2		Užsakovo nepatenkinimas:	5
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	17	Reikalavimo tipas:	Panaudojimo atvejis #:	8
Aprašymas:	Sąnario parametrai esantys parametrų panelėje pilnai nusako sąnarį			
Pagrindimas:	Varotojui paliekama didesnė konstravimo laisvė			
Šaltinis	Analitikas			
Tikimo kriterijus:	Suvedus sąnario parametrus vienodai su kito sąnario parametrais, jis taps identiškas kitam sąnariui ir atsidurs to sąnario vietoje.			
Užsakovo tenkinimas:	2		Užsakovo nepatenkinimas:	2
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	18	Reikalavimo tipas:	Panaudojimo atvejis #:	9
Aprašymas:	Jungiamųjų detalių formą galima pasirinkti iš numatytų šablonų			
Pagrindimas:	Nepatyrusiam vartotojui sunku pačiam nubraižyti detalią formą			
Šaltinis	Analitikas			
Tikimo kriterijus:	Jungiamoji detalė turi parametrą nusakantį formos šabloną.			
Užsakovo tenkinimas:	4		Užsakovo nepatenkinimas:	3
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	19	Reikalavimo tipas:	Panaudojimo atvejis #:	9
Aprašymas:	Kiekvienas jungiamosios detalės formos šablonas turi savo parametrų grupę			
Pagrindimas:	Detalių forma turi būti pritaikoma vartotojo poreikiams			
Šaltinis	Analitikas			
Tikimo kriterijus:	Kiekvieną formos šabloną galima deformuoti keičiant šablono parametrus			
Užsakovo tenkinimas:	2		Užsakovo nepatenkinimas:	4
Priklausomybės:	18, 20	Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	20	Reikalavimo tipas:	Panaudojimo atvejis #:	10
Aprašymas:	Jungiamosios detalės parametrus galima redaguoti			
Pagrindimas:	Kiekviena jungiamoji detalė gali turėti unikalius parametrus			
Šaltinis	Analitikas			
Tikimo kriterijus:	Jungiamosios detalės parametrus galima bet kuriuo metu redaguoti			
Užsakovo tenkinimas:	2		Užsakovo nepatenkinimas:	5
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	21	Reikalavimo tipas:	Panaudojimo atvejis #:	11
Aprašymas:	Yra parametrai kurie yra taikomi visos konstrukcijos savybėms nusakyti			
Pagrindimas:	Ta pati konstrukcija gali turėti skirtingas savybes			
Šaltinis	Analitikas			
Tikimo kriterijus:	Yra parametrų skydelis su visos konstrukcijos parametrais			
Užsakovo tenkinimas:	4		Užsakovo nepatenkinimas:	2
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	22	Reikalavimo tipas:	Panaudojimo atvejis #:	11
Aprašymas:	Konstrukcijos mąstelį galima keisti pažymint bet kurią jungiamąją detalę ir įvedant jos realų ilgį (metrais)			
Pagrindimas:	Vartotojas gali lengvai ir pakankamai tiksliai nurodyti konstrukcijos mąstelį			
Šaltinis	Analitikas			
Tikimo kriterijus:	Grafinis redaktorius turi įrankį mąsteliui keisti			
Užsakovo tenkinimas:	4		Užsakovo nepatenkinimas:	2
Priklausomybės:	21	Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	23	Reikalavimo tipas:	Panaudojimo atvejis #:	12
Aprašymas:	Yra galimybė prijungti jutiklius			
Pagrindimas:	Kiekvienas robotas neapsieina be jutiklių			
Šaltinis	Analitikas			
Tikimo kriterijus:	Grafinis redaktorius turi įrankį jutiklių prijungimui			
Užsakovo tenkinimas:	1		Užsakovo nepatenkinimas:	5
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	24	Reikalavimo tipas:	Panaudojimo atvejis #:	12
Aprašymas:	Prijungiant jutiklį reikiamas modelis parenkamas iš sąrašo			
Pagrindimas:	Jutiklių gali būti įvairių, labai skirtingų tipų			
Šaltinis	Analitikas			
Tikimo kriterijus:	Pridėjimo įrankis turi sąrašo tipo parametrą jutiklio tipui nurodyti.			
Užsakovo tenkinimas:	4		Užsakovo nepatenkinimas:	4
Priklausomybės:	23	Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	25	Reikalavimo tipas:	Panaudojimo atvejis #:	12
Aprašymas:	Įterpiant jutiklį reikia nurodyti vetą jungiamosios detalės paviršiuje.			
Pagrindimas:	Jutikliai turi būti surišti su jungiamosiomis detalėmis			
Šaltinis	Analitikas			
Tikimo kriterijus:	Jutiklis yra surišamas su jungiamąja detale ir juda kartu su ja			
Užsakovo tenkinimas:	3		Užsakovo nepatenkinimas:	3
Priklausomybės:	23	Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	26	Reikalavimo tipas:	Panaudojimo atvejis #:	13
Aprašymas:	Jutiklio vietą galima keisti			
Pagrindimas:	Tinkamiausia jutiklio vieta nusakoma ekspermentiškai.			
Šaltinis	Analitikas			
Tikimo kriterijus:	Perkėlimo įrankis leidžia pakeisti jutiklio vietą			
Užsakovo tenkinimas:	2		Užsakovo nepatenkinimas:	5
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	27	Reikalavimo tipas:	Panaudojimo atvejis #:	13
Aprašymas:	Keičiant jutiklio vietą reikia nurodyti jungiamosios detalės paviršiaus vietą			
Pagrindimas:	Jutiklis yra surištas su jungiamąja detale			
Šaltinis	Analitikas			
Tikimo kriterijus:	Nauja jutiklio pozicija yra ant jungiamosios detalės paviršiaus			
Užsakovo tenkinimas:	2		Užsakovo nepatenkinimas:	3
Priklausomybės:	26	Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	28	Reikalavimo tipas:		Panaudojimo atvejis #:	14
Aprašymas:	Jutiklio parametrus galima keisti				
Pagrindimas:	Kiekvienas jutiklis gali būti unikalus				
Šaltinis	Analitikas				
Tikimo kriterijus:	Jutiklio parametrus galima keisti parametrų skydelyje				
Užsakovo tenkinimas:	1			Užsakovo nepatenkinimas:	4
Priklausomybės:			Konfliktai:	Nėra	
Papildoma medžiaga:					
Istorija:	Užregistruotas 2013m. balandžio 29d.				

Reikalavimas #:	29	Reikalavimo tipas:		Panaudojimo atvejis #:	14
Aprašymas:	Kiekvienas jutiklio tipas gali turėti specialius parametrus				
Pagrindimas:	Imituojami realūs jutikliai				
Šaltinis	Analitikas				
Tikimo kriterijus:	Jutiklio tipai turi atskirą parametrų grupę parametrų skydelyje				
Užsakovo tenkinimas:	3			Užsakovo nepatenkinimas:	2
Priklausomybės:	28		Konfliktai:	Nėra	
Papildoma medžiaga:					
Istorija:	Užregistruotas 2013m. balandžio 29d.				

Reikalavimas #:	30	Reikalavimo tipas:		Panaudojimo atvejis #:	15
Aprašymas:	Valdančios programos aprašo kalbą galima keisti pasirenkant iš sąrašo				
Pagrindimas:	Skirtingi vartotojai turi patirties skirtingose programavimo kalbose				
Šaltinis	Analitikas				
Tikimo kriterijus:	Prie skripų redagavimo panelės yra kalbos pasirinkimo sąrašas				
Užsakovo tenkinimas:	5			Užsakovo nepatenkinimas:	1
Priklausomybės:			Konfliktai:	Nėra	
Papildoma medžiaga:					
Istorija:	Užregistruotas 2013m. balandžio 29d.				

Reikalavimas #:	31	Reikalavimo tipas:		Panaudojimo atvejis #:	15
Aprašymas:	Pakeitus skripto kalbą atliekama skripto validacija				
Pagrindimas:	Vartotojas turi matyti ar jo programos aprašas tenkina naujai parinktą kalbą				
Šaltinis	Analitikas				
Tikimo kriterijus:	Pakeitus kalbą vartotojas informuojamas apie programos aprašo tinkamumą				
Užsakovo tenkinimas:	2			Užsakovo nepatenkinimas:	2
Priklausomybės:	30		Konfliktai:	Nėra	
Papildoma medžiaga:					
Istorija:	Užregistruotas 2013m. balandžio 29d.				

Reikalavimas #:	32	Reikalavimo tipas:	Panaudojimo atvejis #:	16
Aprašymas:	Valdanti programa susideda iš kelių fragmentų			
Pagrindimas:	Skirtingi valdančios programos fragmetai yra vykdomi skirtingu simuliacijos metu			
Šaltinis	Analitikas			
Tikimo kriterijus:	Atskiri programos fragmetai turi atskiras redagavimo sritis			
Užsakovo tenkinimas:	4		Užsakovo nepatenkinimas:	2
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	33	Reikalavimo tipas:	Panaudojimo atvejis #:	16
Aprašymas:	Valdančios programos kodą galima keisti net ir simuliacijos vykdymo metu			
Pagrindimas:	Simuliacijos metu pastebėta klaidinga elgsena gali būti ištaisyta neperleidžiant simuliacijos per naujo			
Šaltinis	Analitikas			
Tikimo kriterijus:	Simuliacijos metu paredaguotas kodas pakeičia buvusį kodą nuo pat išsaugojimo momento			
Užsakovo tenkinimas:	5		Užsakovo nepatenkinimas:	1
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	34	Reikalavimo tipas:	Panaudojimo atvejis #:	17
Aprašymas:	Išsaugant valdantį skriptą atliekama validacija			
Pagrindimas:	Simuliaciją galima vykdyti tik su validžiu kodu			
Šaltinis	Analitikas			
Tikimo kriterijus:	Jei išsaugant kodą jis nėra validus, vartotojas apie tai informuojamas			
Užsakovo tenkinimas:	3		Užsakovo nepatenkinimas:	3
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	35	Reikalavimo tipas:	Panaudojimo atvejis #:	17
Aprašymas:	Validacija atliekama kiekvienam skripto fragmentui atskirai			
Pagrindimas:	Vartotoją domina tik šiuo metu keičiamo kodo validumas			
Šaltinis	Analitikas			
Tikimo kriterijus:	Informacija apie validumą pateikiama tik susijusi su redaguojamu fragmentu			
Užsakovo tenkinimas:	2		Užsakovo nepatenkinimas:	1
Priklausomybės:	34	Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	36	Reikalavimo tipas:		Panaudojimo atvejis #:	18
Aprašymas:	Simuliacijos parametrus galima keisti				
Pagrindimas:	Vartotojui paliekama laisvė keisti simuliacijos vykdymo savybes				
Šaltinis	Analitikas				
Tikimo kriterijus:	Prieš vykdant simuliaciją vartotojui atverčiama simuliacijos parametrų panelė				
Užsakovo tenkinimas:	3			Užsakovo nepatenkinimas:	3
Priklausomybės:			Konfliktai:	Nėra	
Papildoma medžiaga:					
Istorija:	Užregistruotas 2013m. balandžio 29d.				

Reikalavimas #:	37	Reikalavimo tipas:		Panaudojimo atvejis #:	18
Aprašymas:	Tam tikrą simuliacijos parametrų dalį galima keisti jau vykdomai simuliacijai				
Pagrindimas:	Yra simuliacijos parametrų, kurie turi būti skirtingi tamtikruose simuliacijos etapuose				
Šaltinis	Analitikas				
Tikimo kriterijus:	Simuliacijos vykdymo metu dalis parametrų esančių panelėje yra redaguojami				
Užsakovo tenkinimas:	2			Užsakovo nepatenkinimas:	2
Priklausomybės:	36		Konfliktai:	Nėra	
Papildoma medžiaga:					
Istorija:	Užregistruotas 2013m. balandžio 29d.				

Reikalavimas #:	38	Reikalavimo tipas:		Panaudojimo atvejis #:	19
Aprašymas:	Sukonstruotą robotą galima virtualiai išbandyti realiu laiku, vykdant simuliaciją.				
Pagrindimas:	Tai yra kuriamos sistemos tikslas				
Šaltinis	Analitikas				
Tikimo kriterijus:	Sukonstruotas robotas yra simuliuojamas realiu laiku naudojant pateiktą valdančią programą. Simuliacijos fizika atitinka realaus pasaulio dėsnius.				
Užsakovo tenkinimas:	2			Užsakovo nepatenkinimas:	5
Priklausomybės:			Konfliktai:	Nėra	
Papildoma medžiaga:					
Istorija:	Užregistruotas 2013m. balandžio 29d.				

Reikalavimas #:	39	Reikalavimo tipas:	Panaudojimo atvejis #:	19
Aprašymas:	To pačio projekto simuliacija kiekvieną kartą turi būti vienoda			
Pagrindimas:	Vartotojas turi matyti pasikeitimą kurį įtakojo tik jo atliktos modifikacijos			
Šaltinis	Analitikas			
Tikimo kriterijus:	Nekeičiant projekto ir jo nustatymų kiekvieną kartą vykdoma simuliacija yra vienoda			
Užsakovo tenkinimas:	4		Užsakovo nepatenkinimas:	3
Priklausomybės:	38	Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	40	Reikalavimo tipas:	Panaudojimo atvejis #:	20
Aprašymas:	Vykdomą simuliaciją bet kuriuo metu galima sustabdyti			
Pagrindimas:	Sustabdžius simuliaciją vartotojas gali atlikti pakeitimus ir stebėti kaip keičiasi simuliacija			
Šaltinis	Analitikas			
Tikimo kriterijus:	Simuliacijos sustabdymai neturi jokios įtakos tolesniam simuliacijos vykdymui			
Užsakovo tenkinimas:	4		Užsakovo nepatenkinimas:	4
Priklausomybės:	38	Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	41	Reikalavimo tipas:	Panaudojimo atvejis #:	20
Aprašymas:	Sustabdytą simuliaciją galima tęsti			
Pagrindimas:	Vartotojas gali stebėti atliktų pakeitimų įtaką			
Šaltinis	Analitikas			
Tikimo kriterijus:	Simuliacija yra tęsema lygiai nuo to momento kada buvo sustabdyta			
Užsakovo tenkinimas:	4		Užsakovo nepatenkinimas:	4
Priklausomybės:	40	Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	42	Reikalavimo tipas:	Panaudojimo atvejis #:	21
Aprašymas:	Simuliacijos stebėjimo tašką galima keisti			
Pagrindimas:	Roboto elgsenos neįmanoma ištirti tik iš vieno rakurso			
Šaltinis	Analitikas			
Tikimo kriterijus:	Keičiant stebėjimo tašką keičiasi simuliacijos rakursas			
Užsakovo tenkinimas:	3		Užsakovo nepatenkinimas:	4
Priklausomybės:		Konfliktai:	Nėra	
Papildoma medžiaga:				
Istorija:	Užregistruotas 2013m. balandžio 29d.			

Reikalavimas #:	43	Reikalavimo tipas:		Panaudojimo atvejis #:	22
Aprašymas:	Valdantis skriptas turi galimybę naudoti neuroninį tinklą				
Pagrindimas:	Neuroniniai tinklai gali būti labai naudinga priemonė robotų kūrime				
Šaltinis	Analitikas				
Tikimo kriterijus:	Skripto interpretatorius turi integruotą neuroninių tinklų biblioteką				
Užsakovo tenkinimas:	5			Užsakovo nepatenkinimas:	1
Priklausomybės:			Konfliktai:	Nėra	
Papildoma medžiaga:					
Istorija:	Užregistruotas 2013m. balandžio 29d.				

Reikalavimas #:	44	Reikalavimo tipas:		Panaudojimo atvejis #:	22
Aprašymas:	Turi būti priemonė vartotojui apmokyti valdančios programos neuroninius tinklus				
Pagrindimas:	Neuroniniai tinklai yra neveiksmingi kol neapmokyti				
Šaltinis	Analitikas				
Tikimo kriterijus:	Simuliacijos metu vartotojas gali nurodyti robotui kaip toliau elgtis. Ši informacija yra išsaugoma neuroniniame tinkle				
Užsakovo tenkinimas:	5			Užsakovo nepatenkinimas:	1
Priklausomybės:	43		Konfliktai:	Nėra	
Papildoma medžiaga:					
Istorija:	Užregistruotas 2013m. balandžio 29d.				

Reikalavimas #:	45	Reikalavimo tipas:		Panaudojimo atvejis #:	23
Aprašymas:	Apmokant robotą vartotojas sustabdęs simuliaciją nurodo kokią pozą robotas turi stengtis užimti esamoje situacijoje				
Pagrindimas:	Vartotojas gali intuityviai apmokyti robotą				
Šaltinis	Analitikas				
Tikimo kriterijus:	Yra grafinis įrankis sekančios pozicijos nurodymui, kuris perduoda apmokymo duomenis apmokymo skriptui				
Užsakovo tenkinimas:	5			Užsakovo nepatenkinimas:	1
Priklausomybės:	40		Konfliktai:	Nėra	
Papildoma medžiaga:					
Istorija:	Užregistruotas 2013m. balandžio 29d.				

Reikalavimas #:	46	Reikalavimo tipas:		Panaudojimo atvejis #:	23
Aprašymas:	Apmokant robotą, apmokymo duomenys yra išsaugojami projekto hierarchijos komponente				
Pagrindimas:	Tuos pačius apmokymo duomenis galima pateikti kitam neuroniniam tinklui ar kitam apmokymo skriptui				
Šaltinis	Analitikas				
Tikimo kriterijus:	Kiekvienas apmokymas yra išsaugotas projekte neprarandęs informacijos				
Užsakovo tenkinimas:	4			Užsakovo nepatenkinimas:	1
Priklausomybės:	45		Konfliktai:	Nėra	
Papildoma medžiaga:					
Istorija:	Užregistruotas 2013m. balandžio 29d.				

Reikalavimas #:	47	Reikalavimo tipas:		Panaudojimo atvejis #:	24
Aprašymas:	Bet kuriuo simuliacijos metu galima peržiūrėti kiekvieno jutiklio išvesties duomenis				
Pagrindimas:	Patogi priemonė derinti jutiklių parametrus bei valdantį skriptą				
Šaltinis	Analitikas				
Tikimo kriterijus:	Simuliacijos metu pažymėjus jutiklį matoma jo išvesties informacija				
Užsakovo tenkinimas:	4			Užsakovo nepatenkinimas:	2
Priklausomybės:			Konfliktai:	Nėra	
Papildoma medžiaga:					
Istorija:	Užregistruotas 2013m. balandžio 29d.				

Reikalavimas #:	48	Reikalavimo tipas:		Panaudojimo atvejis #:	24
Aprašymas:	Jutiklio išvesties informaciją galima matyti grafiko forma				
Pagrindimas:	Jei simuliacija nesustabdyta, skaitinės jutiklių reikšmės neturi prasmės				
Šaltinis	Analitikas				
Tikimo kriterijus:	Visi jutikliai turintys nedaugiau trijų išvesties parametrų saugo reikšmių istoriją ir gali būti atvaizduoti grafiko forma				
Užsakovo tenkinimas:	5			Užsakovo nepatenkinimas:	1
Priklausomybės:	47		Konfliktai:	Nėra	
Papildoma medžiaga:					
Istorija:	Užregistruotas 2013m. balandžio 29d.				

Reikalavimas #:	49	Reikalavimo tipas:		Panaudojimo atvejis #:	25
Aprašymas:	Simuliacijos vaizdo įrašą galima išsaugoti failų sistemoje				
Pagrindimas:	Vartotojas nori projekto rezultatus pasidalinti su kitais asmenimis				
Šaltinis	Analitikas				
Tikimo kriterijus:	Simuliacijos vaizdo įrašas išsaugotas failų sistemoje ir yra peržiūrimas standartinėmis vaizdo peržiūros programomis				
Užsakovo tenkinimas:	3			Užsakovo nepatenkinimas:	2
Priklausomybės:	38		Konfliktai:	Nėra	
Papildoma medžiaga:					
Istorija:	Užregistruotas 2013m. balandžio 29d.				

Reikalavimas #:	50	Reikalavimo tipas:		Panaudojimo atvejis #:	25
Aprašymas:	Simuliaciją galima eksportuoti jos vizualiai nestebint				
Pagrindimas:	Vartotojui nereikia laukti realaus simuliacijos laiko kad gautų jos vaizdo įrašą				
Šaltinis	Analitikas				
Tikimo kriterijus:	Nurodyta simuliacijos trukmė yra įvykdoma greičiau nei realiu laiku, o vaizdo įraše matoma realaus laiko simuliacija.				
Užsakovo tenkinimas:	3			Užsakovo nepatenkinimas:	1
Priklausomybės:	49		Konfliktai:	Nėra	
Papildoma medžiaga:					
Istorija:	Užregistruotas 2013m. balandžio 29d.				