TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Marti Ingmar Liibert 179288IAIB

ROBOTSÜSTEEMIDE OLEKUAUTOMAATIDE ÕPPIMINE MUDELIPÕHISEKS TESTIMISEKS

Bakalaureusetöö

Juhendaja: Gert Kanter
MSc

Tallinn 2020

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Marti Ingmar Liibert

18.05.2020

Annotatsioon

Robotsüsteemide mudelipõhiseks testimiseks mudelite loomine on ajakulukas osa testimisprotsessist. Kiirendamaks testimisprotsessi, oli käesoleva töö eesmärk laiendada TestIt tööriistakomplekti lihtsalt konfigureeritava ja laiendatava tööriistaga, mis lubab kasutajal automaatselt õppida ja luua testitava robotsüsteemi mudel Uppaal automaadi kujul. Tööriist võimaldab vastavalt konfiguratsioonile avastada roboti olekuruumi, selle suurust vähendada läbi klasterdamise ning teisendada see Uppaal automaadi kujule. Samuti on võimalik lihtsalt konfiguratsiooni kaudu asendada vaikefunktsionaalsused kasutaja loodud ROS (*Robot Operating System*) teenustega. Tööriista toimimist valideeriti kahest patrullrobotist ja sissetungijast koosneval simulatsioonil, kus tööriista poolt automaatselt loodud mudel võimaldas testida patrullrobotite tundmatu objekti tuvastamise funktsionaalsust.

Tarkvara on implementeeritud kasutades ROS vahetarkvara ning toetab robotsüsteeme, mis implementeerivad ROS liidese suhtluseks. Rakenduse lähtekood on kättesaadav GitHubi hoidlas https://github.com/ingmarliibert/testit ning valideerimiseks kasutatud näide on kättesaadav GitHubi hoidlas https://github.com/ingmarliibert/testit-patrol-learn.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 37 leheküljel, 7 peatükki, 27 joonist.

Abstract

Robotic System Automata Learning for Model-Based Testing

Testing autonomous robots is challenging because of the various components they incorporate, some of which are from third parties. For that reason a good method to test these robot systems is integration or system testing by running the whole system. However, testing robots in real life can take an enormous amount of time and be prohibitively costly. For that reason, the TestIt toolkit was created, which helps users to test robots concurrently in simulation.

This thesis introduces a novel extension to the TestIt toolkit, which allows the user to automatically learn the model of the robot as Uppaal automata and use them for model-based testing. The added tool enables the user to explore the state space of the robotic system, decrease its size by using clustering techniques and build Uppaal automata of the reduced size state space. In addition, it is easily extendable with custom functionality such as custom clustering or exploration strategies as ROS services.

The tool is implemented using ROS and enables the user, in conjunction with the rest of the TestIt toolkit, to automatically test robotic systems that implement a ROS interface for communication. The source code of the TestIt toolkit with automated model learning is available in a repository located at https://github.com/ingmarliibert/testit.

The tools are validated by being applied to a simulation of two patrol robots and one intruder robot. The goal is to test the patrol robots' ability of detecting an unknown object planted by the intruder. Testing is done by controlling the intruder robot. The presented tool successfully automatically builds a model for the intruder robot that enables testing this scenario. The example used for validation is located at https://github.com/ingmarliibert/testit-patrol-learn.

The thesis is in Estonian and contains 37 pages of text, 7 chapters, 27 figures.

Lühendite ja mõistete sõnastik

DTRON	*Distributed* TRON, hajus TRON
GTA	*Grand Theft Auto,* arvutimäng
IO	*Input-Output*, sisend-väljund
MBT	*Model-Based Testing*, mudelipõhine testimine
ROS	*Robot Operating System*, robotite operatsioonisüsteem
SUT	*System Under Test,* testitav süsteem
TRON	*Testing Realtime Systems Online*, reaalaja süsteemide onlain testimine

Sisukord

Jooniste loetelu

1 Sissejuhatus

Autonoomsete robotite automaatne testimine on aktuaalne teema. Robotsüsteemid koosnevad tihti paljudest erinevatest komponentidest, millest mitmed on väliste osapoolte poolt loodud. Seetõttu pole võimalik adekvaatselt testida robotsüsteemi toimimist ilma kõiki komponente korraga jooksutamata. Selleks on sobilik simulatsioonis robotsüsteemide testimine. Päriselus testimine pole tihti võimalik, kuna kõigi olukordade testimine võtaks kaua aega ja oleks kulukas. Näiteks on väidetud, et autonoomsete sõidukite töökindluse vähesel määral suurendamine nõuaks sadade miljardite kilomeetrite sõitmist, mis võtaks päriselus isegi sadade autode pideva sõitmise puhul aega tuhandeid aastaid [9, lk 191] . Simulatsioonis saab aga roboteid paralleelselt testida ning teatud tingimustel jooksutada nende simulatsiooni reaalajast kiiremini.

Testimine ja verifitseerimine võib kulutada kuni 50% arendusressursist ja robotsüsteemide puhul isegi rohkem [6, lk 6] . Seetõttu on tähtis võimalikult suures mahus testimist automatiseerida, et arendusprotsessi kiirendada. Autorile teadaolevalt ei leidu siin töös tutvustatavale tööriistale sarnast ning laialtlevinud automaatset üldist testimisraamistikku robotsüsteemide testimiseks. Leidub küll aga näiteid kindlaid juhte testivatest automaatsetest testimissüsteemidest [1] .

TestIt tööriistakomplekt [10] , kuhu loodud tööriist on integreeritud, on mõeldud autonoomsete ROS (*Robot Operating System*) liidesega robotsüsteemide testimiseks musta kasti meetodil. Musta kasti meetodil testides on teada ainult testitava süsteemi sisendid ja väljundid ning testimine toimub saates süsteemile sisendeid ning kontrollides väljundit [10] . Eelneval kujul võimaldas TestIt olemasoleva mudeli põhjal rakendada robotsüsteemide testimiseks mudelipõhist testimist, kuid TestIt komplektis sisalduv adapter võimaldas testitavale süsteemile saata ainult *Move Base* navigatsiooni käske. Antud töö eesmärk oli luua tööriist, mis implementeerib ühe potentsiaalse viisi, kuidas saab õppida etteantud robotsüsteemi testimudel automaatselt Uppaal ajaga automaadi kujul robotsüsteemi olekuruumi avastamise kaudu. Robotsüsteemi

testimudeli automaatne loomine kiirendab mudelipõhist testimisprotsessi, automatiseerides aeganõudva manuaalse tegevuse mudelipõhises testimisprotsessis. Samuti oli eesmärk integreerida selline tööriist ülejäänud TestIt tööriistakomplektiga ning üldistada teisi komplektis olevaid tööriistu võimaldades mudelipõhist testimist lisaks *Move Base* navigatsiooni käskudele kõikvõimalike käsutüüpide puhul. Kolmas eesmärk oli valideerida, kas selle tööriista poolt loodud mudel võimaldab ühte etteantud juhtu sama edukalt testida kui käsitsi loodud mudel.

Töö on jagatud neljaks osaks: analüüs, kus tutvustatakse lahenduse nõudeid, arhitektuur, kus tutvustatakse üldist loodud tööriista loogilist arhitektuuri, teostus, kus tutvustatakse ja põhjendatakse tähtsamate üksikosade teostuse detaile ning valideerimine, kus rakendatakse loodud tööriista ühel näitel, tõestamaks, et loodud tööriist täidab eesmärke.

2 Taust ja olemasolev tarkvara

2.1 Olemasolev tarkvara

Autorile teadaolevalt ei leidu robotsüsteemide automaatseks mudelipõhiseks musta kasti testimiseks mõeldud üldiseid ja konfigureeritavaid tööriistu koos automaatse mudeli õppimisega. Kirjeldatud on küll teisi automaatseid robotsüsteemide testimiseks mõeldud tehnikaid, mis näiteks kasutavad tõenäosuslikku optimeerimist.

Üks esimesi robotsüsteemide automaatseks testimiseks loodud tööriistu kasutab Bayesi optimeerimist, mis integreerib ka dimensionaalsuse vähendamise tehnikaid [7] . TestIt, kuhu on integreeritud ka loodud tööriist, toetab Uppaal automaadil põhinevat mudelipõhist testimist ja testistsenaariumite optimeerimist [10] . Samuti on kirjeldatud automaatseid robotite testimistehnikaid spetsiifiliste süsteemide jaoks, näiteks autonoomse sõiduki testimiseks, kasutades simulatsioonina arvutimängu GTA (*Grand Theft Auto*) 5 [1] .

2.2 ROS

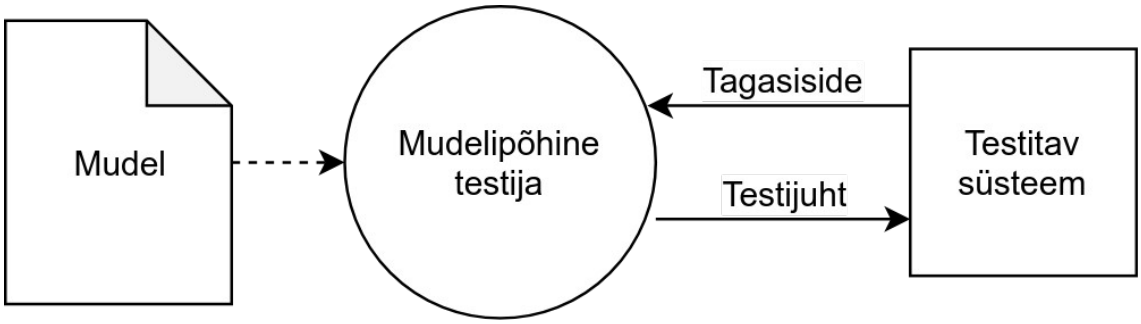
ROS on „paindlik raamistik roboti tarkvara loomiseks” [2] . See on „kogumik tööriistadest ja teekidest ning konventsioonidest, mis aitavad luua keerulist ja robustset robotite tarkvara erinevatele platvormidele” [2] .

ROSi põhiline osa on ühtlustatud sõnumite saatmine erinevate tarkvara osade vahel. ROSis on spetsifitseeritud sõlmed, mille vahel saab sõnumeid saata, kasutades kas rubriike, teenuseid või tegevusi [12] . See funktsionaalsus on tähtis musta kasti testimise puhul, kuna lubab saata käske ja sõnumeid robotitele ühtset moodi.

2.3 Mudelipõhine testimine

Mudelipõhine testimine on testimistehnika, kus kasutatakse formaalseid mudeleid, et juhtida testimisprotsessi. Seda võib vaadelda kui automatiseeritud musta kasti testimist [18, lk 8]. Mudeleid kasutatakse, et spetsifitseerida testitava süsteemi käitumine ning testi eesmärk. Testija genereerib mudelist käivitatavad testijuhud, millega testitakse testitavat süsteemi otse.

MBT (*Model-based testing)* eelised tulevad kõige paremini esile integratsiooni- ja süsteemitestimises, kus pannakse tööle kogu süsteem ning kus testitakse kõiki osasid üksteisega integratsioonis [10] . Mudelipõhist testimist kasutatakse tihti vastavustestimiseks, kus eesmärgiks on kindlustada testitava süsteemi vastavus spetsifikatsioonile ehk mudelile [17] . Joonisel 1 on näha, kuidas suhestuvad mudel, mudelipõhine testija ning testitav süsteem üksteisega.



Joonis 1. Mudelipõhine testimine.

2.3.1 Uppaal ajaga automaat

Kuigi TestIt võimaldab mudelipõhist testimist erinevate vahenditega, siis ametlikult toetatud variant on Uppaal tööriistadega. See tähendab, et testi mudel on Uppaal ajaga automaat [10] .

Uppaal modelleerimiskeel põhineb ajaga automaatidel. Ajaga automaat on lõplik automaat koos kella muutujatega. Uppaalis on süsteem modelleeritud kui paralleelsete automaatide võrk. Mudelis on lisaks diskreetsed muutujad, mis on osa olekust. Olek on defineeritud kui kõigi süsteemis olevate automaatide asukohad, kellade ning teiste muutujate väärtused. Uppaal tööriistade hulka kuuluvad ka verifitseerimise tööriist ning mudeli kontrollija, mille juurde kuulub ka Uppaali päringukeel [5] .

2.3.2 Uppaal TRON ja DTRON

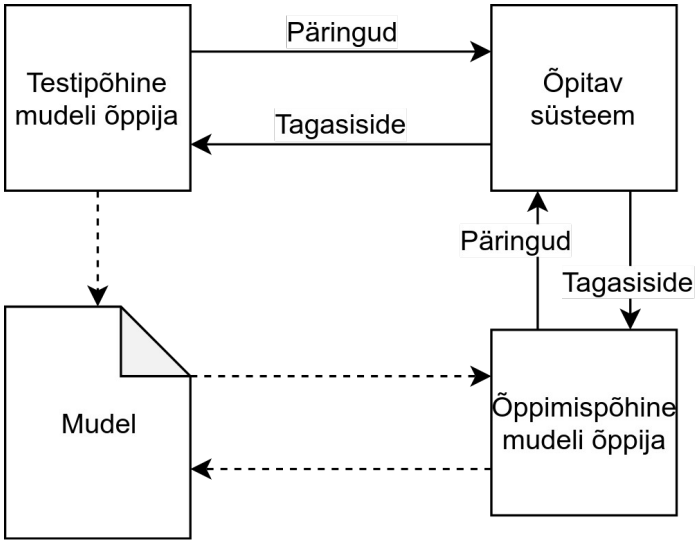
Uppaal TRON (*Testing Realtime Systems ONline*) on Uppaalil põhinev onlain testimistööriist [8] , mis võimaldab Uppaal mudeli järgi läbi viia vastavustesti. TRON genereerib mudeli järgi kanalite signaalid ja seob need muutujatega ning kontrollib testitava süsteemi poolt saadetavaid vastuseid. Suhtlus TRONi ja testitava süsteemi vahel toimub adapterite abil [16] .

DTRON (*Distributed Testing Realtime systems ONline*) on teek, mis võtab Uppaal TRONist sõnumid vastu, töötleb neid ning saadab need laiali Spread võrku kindlatel kanalitel. Samuti kuulab ta Spread võrku saadetud sõnumeid ning saadab TRONi tagasi vastused [4] .

2.4 Mudeli õppimine mudelipõhiseks testimiseks

Mudeli õppimist mudelipõhiseks testimiseks jaotatakse kaheks: testipõhiseks õppimiseks ja õppimispõhiseks testimiseks. Testipõhine õppimine sisaldab endas testitava süsteemi poole päringute saatmist. Õppimispõhine testimine sisaldab endas olemasolevate mudelite põhjal paremate mudelite loomist [3, lk 1-2] .

Loodud tööriist kasutab erinevates etappides mõlemat lähenemist. Algselt avastab tööriist robotsüsteemi olekuruumi, saates selle poole käske ehk päringuid, ja konstrueerib mudeli vastavalt robotsüsteemi vastustele. Järgnevalt võimaldab tööriist korrastada mudelit, kasutades olemasolevat mudelit, et leida mudeli olekute vahel uusi üleminekuid. Mudeli korrastamiseks kasutatakse ka robotsüsteemiga suhtlemist, mitte puhast eelneva mudeli pealt õppimist, kuigi tööriista kergelt konfigureeritavus võimaldab kasutajal ka selle etapi ise implementeerida vastavalt vajadustele. Joonisel 2 on näha, kuidas suhestuvad omavahel loodud süsteemis testipõhine mudeli õppimine ning õppimispõhine testimine.



Joonis 2. Mudeli õppimine.

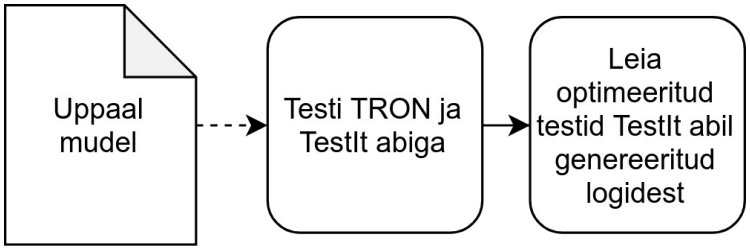
2.5 TestIt

TestIt on tööriistakomplekt, mis võimaldab paralleelset ROS robotsüsteemide testimist ning testistsenaariumite tõenäosuslikku optimeerimist. TestIt võimaldab ka mudelipõhist testimist, mille eelduseks on juba olemasolev Uppaal mudel. TestIt DTRON adapter toetab Uppaal DTRONiga suhtluseks praegu ainult *Move Base* navigatsiooni käske erinevates vormides. TestIt tööriistakomplektis sisaldub ka logimise funktsionaalsus, mis võimaldab määrata, milliste robotsüsteemi rubriikide ja teenuste sõnumid logitakse logifaili [10] .

3 Analüüs

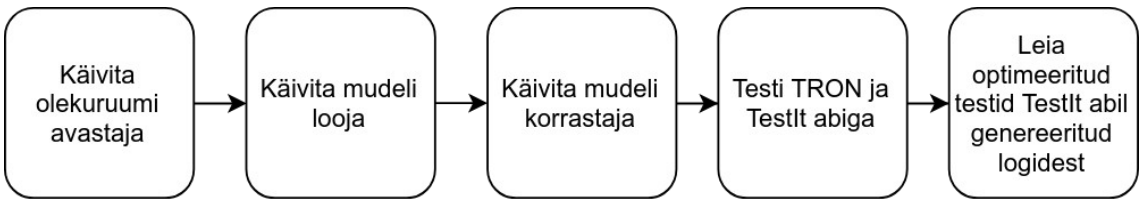
TestIt tarkvara võimaldab mudeli alusel testida multirobotsüsteemi. Sisendiks on robotsüsteemi mudel Uppaal automaadi kujul, mida läbib Uppaal TRON. TestIt pakub vahekihi ehk adapteri Uppaal DTRONist edastatud sõnumitele, mille abil edastatakse sisendid robotsüsteemile. Seejärel võimaldab TestIt genereerida mudeli läbimisel tekkinud logide põhjal tõenäosusliku mudeli, mida saab kasutada, et leida kiiremini üles koodikatet maksimeerivad kohad testis.

Praegusel kujul on TestIt mudelipõhiseks testimiseks mõeldud tööriistade rakendamise eelduseks juba olemasolev Uppaal mudel. Samuti võimaldab TestIt Uppaal DTRONi adapter edastust ainult kindlatel sündmustel, mis suhtlevad *Move Base* navigatsiooni teegiga robotsüsteemi osast. Joonisel 3 on näha TestIt töövoolu enne loodud tööriista.



Joonis 3. TestIt töövool enne loodud tööriista.

Eesmärgiks on teha robotsüsteemi mudelite õppimine ja loomine automaatseks ning loodud mudeli põhjal võimaldada Uppaal TRON abil kõikvõimalike sündmusi edastada robotsüsteemile, et seda testida. Robotsüsteem tähendab siin ühest või rohkemast robotist koosnevat süsteemi koos keskkonnaga. Lisaks on eesmärgiks loodav funktsionaalsus integreerida TestIt tööriistakomplekti osaks. Joonisel 4 on näha TestIt töövoolu peale loodud tööriista.



Joonis 4. TestIt töövool pärast loodud tööriista.

3.1 Funktsionaalsed nõuded

Süsteem peab võimaldama robotsüsteemi olekuruumi avastamist ning selle põhjal mudeli loomist Uppaal ajaga automaadi kujul. Sealjuures on robotsüsteem nii-öelda must kast. Seetõttu peab robotsüsteemiga suhtlus toimima kasutaja poolt spetsifitseeritud kanaleid mööda robotitele sisendeid saates.

Olekuruumi peab saama avastada mitmel robotil ja mitmel sisenditüübil korraga, et saaks luua tervete robotsüsteemide mudeleid, mitte ainult üksikute robotite mudeleid.

Süsteem peab avastatud olekuruumi suurust vähendama kasutaja spetsifitseeritud limiitides, et loodava mudeli läbimist hilisemates testimise faasides kiirendada.

Süsteem peab olema kasutaja poolt kergesti konfigureeritav. Konfigureeritavad peavad olema sisendid, mille järgi robotsüsteemi juhitakse, ning sisendid, mille järgi ekvivalentsiklasse luuakse. Samuti peab saama konfiguratsioonis välja vahetada süsteemi vaikeosade alla kuuluvad ekvivalentsiklasside leidja ja robotitele sisendite saatmise strateegia, et vajadusel saaks kergelt vaikesüsteemi funktsionaalsuse poolt mittevastavatele vajadustele süsteemi kohandada. Sellest tulenevalt peavad vähemalt ekvivalentsiklasside leidja ja sisendite genereerimise strateegia olema teistest sõltumatud eraldiseisvad süsteemi osad.

Genereeritav mudel peab olema Uppaal ajaga automaadi kujul, et saaks käsitsi kergelt mudelit muuta ja verifitseerida Uppaal tööriistade abil ning kasutada Uppaal TRONi, et läbida mudelit robotsüsteemi testimiseks.

Kuna olekuruumi avastamine võib ajapiirangutest tingituna olla mitte täielik, siis peab loodud mudelit saama hilisemalt korrastada, nii et leitakse mudeli olekute vahel uusi üleminekuid. Uute olekutevaheliste üleminekute leidmine suurendab mudelipõhise testija valikuid ning aitab vältida tupikusse jäämist.

Eelnevatest nõuetest lähtuvalt peab süsteemil olema kolm põhilist osa: olekuruumi avastaja, mudeli looja ja mudeli korrastaja.

3.2 Mittefunktsionaalsed nõuded

Süsteem peab olema implementeeritud kasutades ROS vahetarkvara. Nõue tuleneb sellest, et TestIt tööriistakomplekt ise kasutab ka ROS vahetarkvara. Samuti on ROS vabavaraline standard robotsüsteemide loomiseks. Sellest tulenevalt peavad süsteemi kasutavad robotsüsteemid võimaldama suhtlust ROS vahendite abil. See tähendab, et robotsüsteemile saab sisendeid saata ja robotsüsteemilt tagasisidet saada ROS rubriikide (*topic*) või teenuste (*service*) kaudu.

Samuti peavad põhilised süsteemi osad olema implementeeritud Python programmeerimiskeeles, kuna see on üks keeltest, mida ROS toetab, ning see võimaldab vajalikke dünaamilisi toiminguid nagu konfiguratsioonis määratud rubriikide tüüpide dünaamiline importimine programmi ning lihtne stringitöötlus konfiguratsiooni käsitlemiseks.

Süsteemi toimimine TestIt-iga integratsioonis peab olema kaetud automatiseeritud integratsioonitestidega, et tagada lihtne viis kontrollida süsteemi korrektset toimimist.

4 Arhitektuur

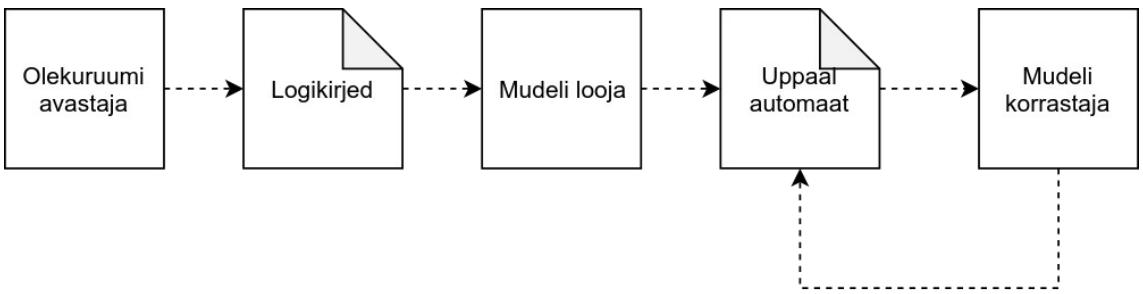
Süsteem koosneb kolmest peamisest komponendist: olekuruumi avastaja, mudeli looja ning mudeli korrastaja.

Olekuruumi avastaja saadab spetsifitseeritud strateegia alusel multirobotsüsteemile signaale kasutades ROS rubriike. Olekuruumi avastaja käivitatakse TestIt töö stsenaariumina, mille käigus logitakse konfigureeritud rubriikide ning teenuste sõnumid ja vastused logifailidesse. Teisisõnu olekuruumi avastaja töö tulemuseks on logifail.

Mudeli looja kasutab sama logifaili, et luua selle alusel mudel. Selle lähenemise eeliseks on see, et mudeli looja ei sõltu olekuruumi avastajast, vaid mudeli looja sisendiks saab anda mistahes logifaili, mille alusel mudel luuakse.

Mudeli korrastaja kasutab loodud mudelit, et leida mudeli olekute vahel uusi võimalikke üleminekuid ning korrigeerida ajastust.

Joonisel 5 on näha süsteemi põhiliste osade väljundeid ja suhet üksteisesse.



Joonis 5. Süsteemi komponendid ja nende vaheline suhtlus.

4.1 Avastaja

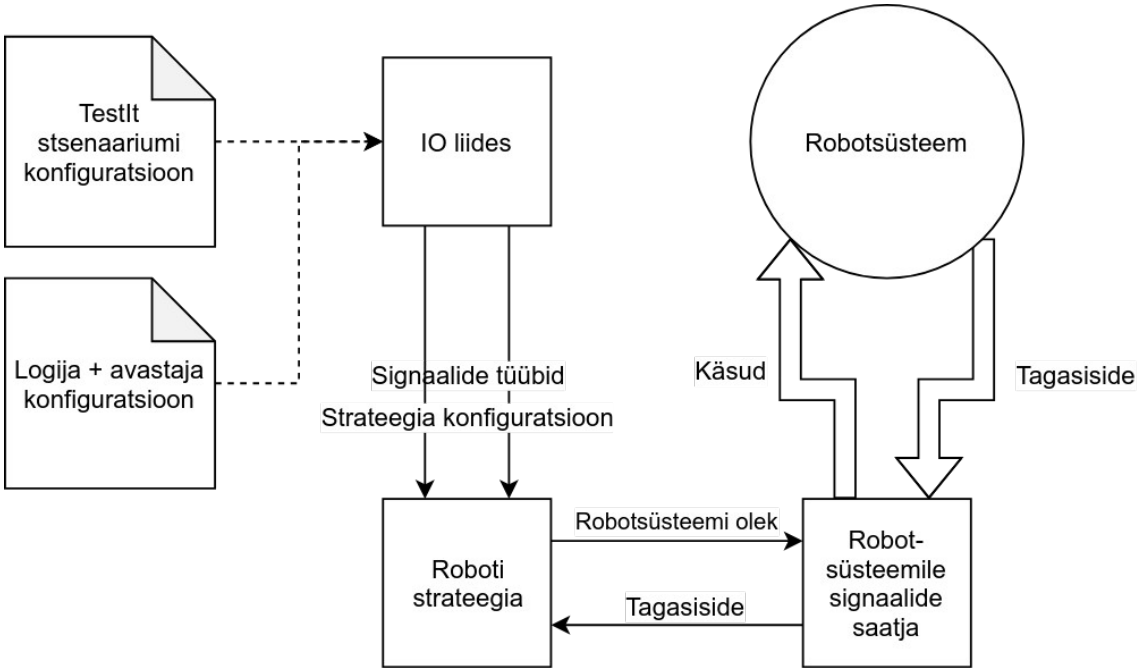
Olekuruumi avastaja koosneb kolmest eraldiseisvast osast: IO (*Input-Output*) liidesest, strateegiast ning robotsüsteemile käskude saatjatest.

IO liides võtab sisendiks TestIt stsenaariumi konfiguratsiooni, logija ja avastaja konfiguratsiooni ning initsialiseerib strateegia ja robotsüsteemile signaalide saatjad vastavalt konfiguratsioonis määratud signaalide tüüpidele ja avastaja väärtustele.

Strateegia võtab sisendiks logi konfiguratsioonis määratud signaalide tüübid ja nende avastamise strateegia ning pakub meetodi, mis tagastab järgmise signaali väärtused, mida robotile saata. Strateegia saab kergelt välja vahetada kasutaja loodud strateegia vastu konfiguratsioonis.

Robotsüsteemile signaalide saatja võtab sisendiks strateegia ja küsib strateegialt robotite signaalide väärtused, mida robotitele saata, ja edastab need robotitele.

Joonisel 6 on näha avastaja komponente ja nende omavahelise suhtluse skeemi.



Joonis 6. Avastaja komponendid ja nende vaheline suhtlus.

4.2 Mudeli looja

Mudeli looja koosneb neljast eraldiseisvast osast: logifaili ja konfiguratsiooni protsessorist, ekvivalentsiklasside leidjast, olekumasina ja Uppaal automaadi ehitajatest.

Logifaili ja konfiguratsiooni protsessor võtab sisendiks TestIt stsenaariumi konfiguratsiooni, logija ja avastaja konfiguratsiooni ning logifaili ja annab väljundiks

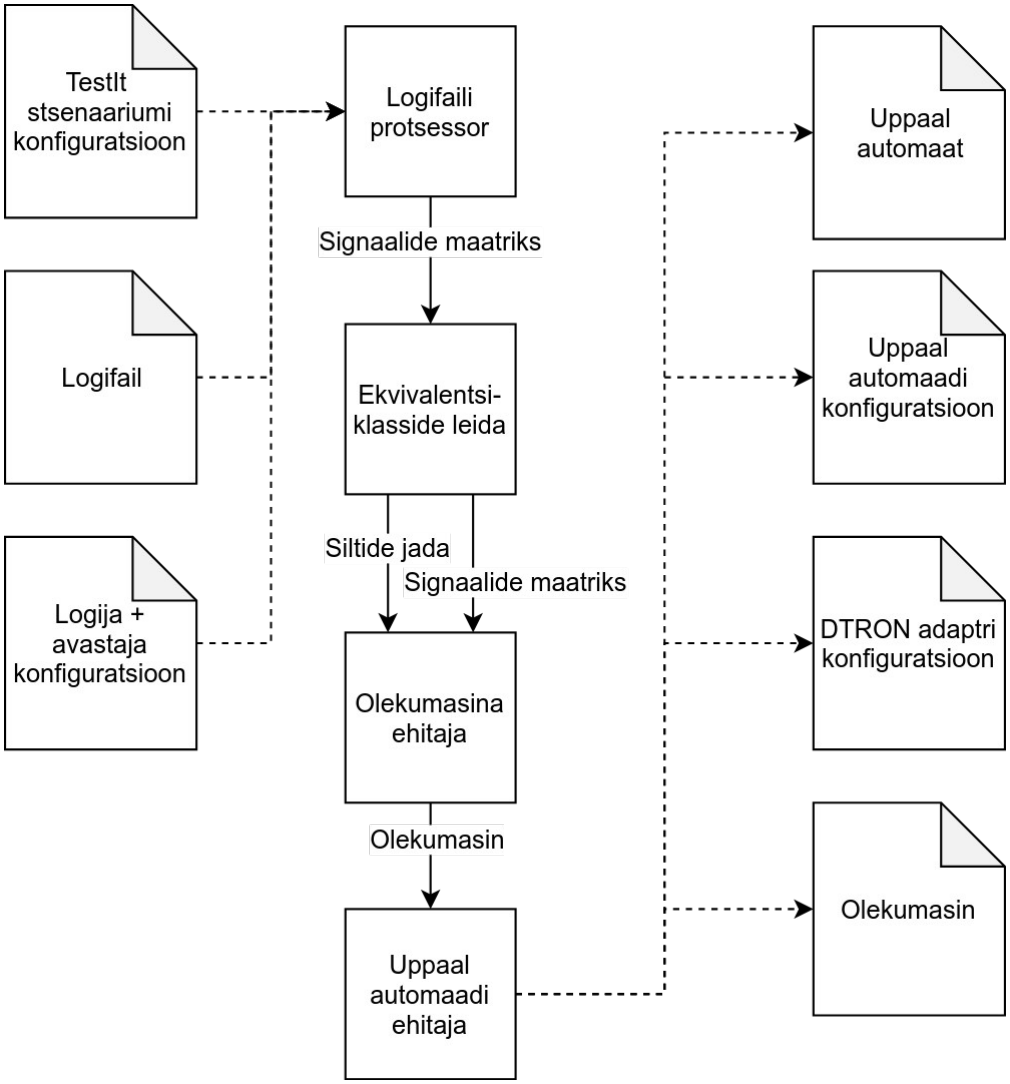
maatriksi robotile saadetud käskude väljade väärtustest. Maatriksi ridadeks on sünkroniseeritud käskude andmed.

Ekvivalentsiklasside leidja võtab sisendiks väärtuste maatriksi ja annab väljundiks jada siltidest, kus ühele maatriksi reale vastab üks silt, ning sõnastiku igale sildile vastavast ekvivalentsiklassi väärtusest.

Olekumasina ehitaja võtab sisendiks siltide jada, sõnastiku igale sildile vastavast ekvivalentsiklassi väärtusest ja jada sünkroniseeritud signaalide tüüpidest ja annab väljundiks olekumasina, mis on defineeritud antud töös kui kolmik üleminekufunktsioonist, lähteolekust, olekufunktsioonist. Üleminekufunktsioon määrab ära, millisest olekust saab millisesse olekusse liikuda, mis käsuga ja ajaga. Olekufunktsioon määrab ära, milline väärtus vastab millisele olekule.

Uppaal automaadi ehitaja võtab sisendiks olekumasina ning teisendab selle Uppaal ajaga automaadi kujule ning väljastab ka vajalikud konfiguratsioonid TestIt DTRONi adapteri tööks.

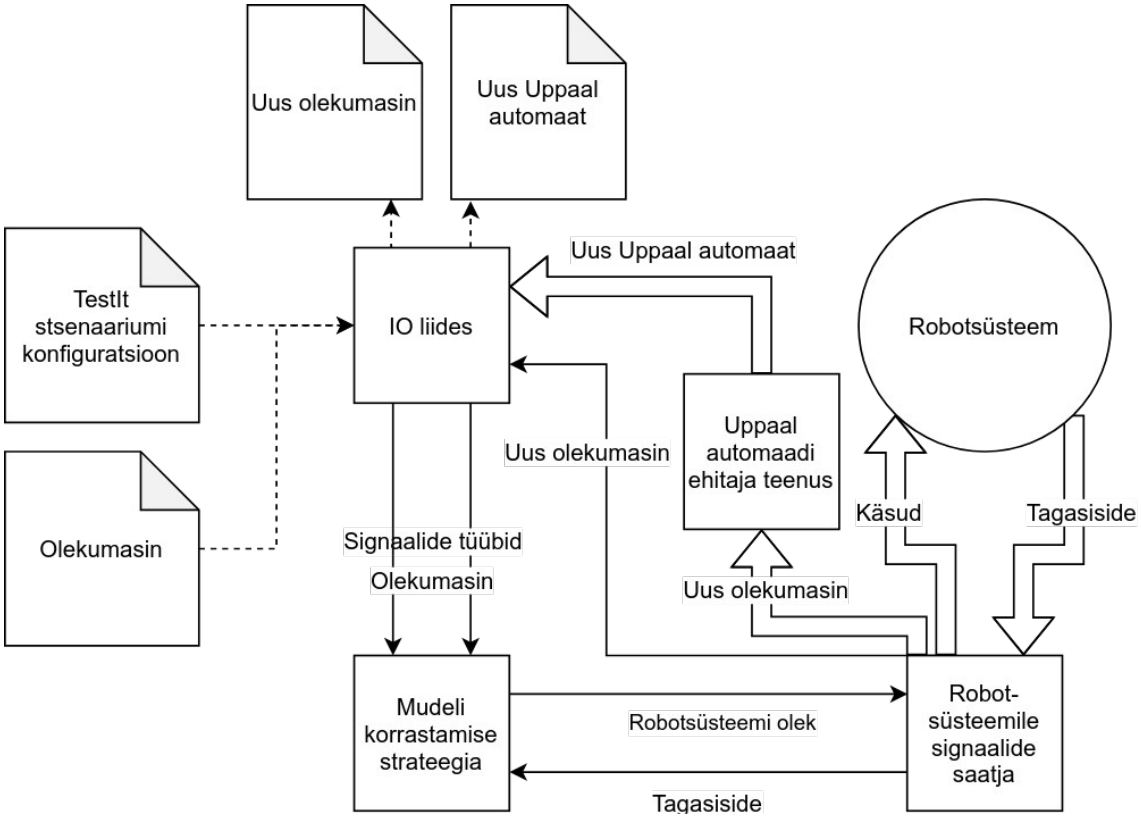
Joonisel 7 on näha mudeli looja komponente ning nende omavahelise suhtluse skeemi.



Joonis 7. Mudeli looja komponendid ja nende omavaheline suhtlus.

4.3 Mudeli korrastaja

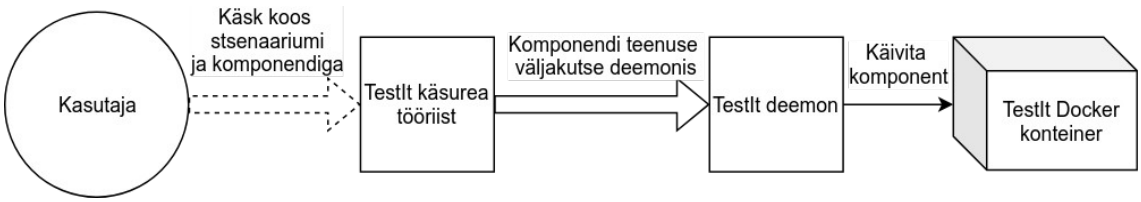
Mudeli korrastaja sisendiks on olekumasin. Olekumasina põhjal saadab see robotsüsteemile käske, et teha mudelit paremaks, ühendades lähedasi olekuid. Samuti korrigeerib see üleminekute ajapiiranguid. Väljundiks on uus Uppaal mudel ning olekumasin. Mudeli korrastaja kasutab ära avastaja funktsionaalsust, asendades avastamise strateegia mudeli korrastamise strateegiaga. Joonisel 8 on näha, kuidas mudeli korrastaja laiendab avastaja funktsionaalsust, asendades strateegia ning lisades pöördumise Uppaal automaadi ehitaja teenuse poole.



Joonis 8. Mudeli korrastaja komponendid ja nende omavaheline suhtlus.

4.4 TestIt integratsioon

TestIt käsurea tööriist võtab argumendina sisse väljakutsutava funktsionaalsuse nimetuse ning stsenaariumi nimetuse ning kutsub vastavalt TestIt deemonis avastaja, mudeli korrastaja või mudeli loomise teenuse välja. TestIt deemon paneb omakorda TestIt Dockeri konteineris [10] vastava stsenaariumi ja funktsionaalsuse tööle. Loodud tööriista integratsioon TestIt-iga toetab kõiki TestIt paralleelselt käitamise funktsionaalsusi. Joonisel 9 on näha skeemi, kuidas käivitatakse õige komponent TestIt Dockeri konteineris.



Joonis 9. TestIt integratsioon loodud süsteemiga.

5 Teostus

5.1 Avastaja

Avastaja võtab sisendiks stsenaariumi konfiguratsiooni ning logija ja avastaja konfiguratsiooni. Logija konfiguratsioonis on määratud, milliste sisendite ehk rubriikide ja teenuste sõnumeid TestIt logija logib. Avastaja režiimis saab ka iga sisendi juures määrata avastamise strateegia konfiguratsiooni.

Avastaja strateegia konfiguratsioon (vt Lisa 2) sisaldab endas võimalust määratleda

- kas jätkatakse logifaili järgi avastamist, nii et logikirjetes olevad olekud loetakse läbituks (joonisel 10 väli *explore.continue*)
- mis muutujaid suurendatakse või vähendatakse ning kui suurte sammude kaupa (joonisel 10 väljad *explore.variables*)
- millised on konstandid ja nende väärtused (joonisel 10 väljad *explore.constants*)
- millised on sünkroniseeritud sisendid (joonisel 10 väljad *syncedExploreTopics*)

```yaml
configuration:
  syncedExploreTopics:
    - [0, 1]
  ...
  inputs:
    - ...
      explore:
        continue: true
        variables:
          - field: "pose.position.x"
            step: 1
            initial: 2
          ...
        constants:
          - field: "pose.orientation.z"
            value: 0
          ...
```

Joonis 10. Avastaja konfiguratsiooni näide.

Avastaja vaikimisi strateegia on implementeeritud sügavuti otsinguna, kus käskude väljade väärtusi suurendatakse ja vähendatakse konfiguratsioonis määratud sammude kaupa. Laiuti otsing on ebaoptimaalne sellises kontekstis, kuna robot peab sel juhul iga sammu juures tagasi minema eelmise sammu juurde, et järgmist sammu täita, mis võtab rohkem aega. Võimalike järgmiste sammude hulgast sügavuti otsingus valitakse esimesena see, mis on juba läbi käidud teekonnast kõige kaugemal, et maksimeerida olekuruumi katvust. Avastaja konfiguratsioonis saab spetsifitseerida iga käsu muutuja sammu ning samuti on võimalus jätkata avastamist eelmisest pooleli jäänud kohast.

Strateegia võtab sisendiks maatriksi sünkroniseeritud käskude sammudest, kus üks rida vastab ühe käsu sammudele, ning pakub meetodi, mis tagastab iga väljakutse peale järgmise oleku. Samuti pakub strateegia meetodi, mis võtab sisendina eelmise tagastatud oleku õnnestumise staatuse. Strateegia tagastatud olek loetakse õnnestunuks, kui üks käsk sünkroniseeritud käskudest õnnestus. Sünkroniseeritud käske käsitletakse strateegias korraga, kuna siis saab kontrollida, kas antud olekute kombinatsioonis on enne oldud.

Strateegiat saab kergelt TestIt stsenaariumi konfiguratsioonis asendada kasutaja looduga (vt Lisa 1). Kasutaja loodud strateegia peab sisaldama ROS rubriiki (konfiguratsioonis

väli *moveStrategyInitTopic*), kuhu saadetakse strateegia initsialiseerimise info ehk avastaja konfiguratsioonis sisalduv info töödeldud kujul. Lisaks peab strateegia sisaldama ROS teenust (konfiguratsioonis väli *moveStrategyService*), kuhu sisendina antakse eelmise käsu tagasiside ning väljundiks on järgmine robotsüsteemi olek.

Strateegia on omakorda sisendiks robotsüsteemile käskude saatjatele, mis küsivad iga sünkroniseeritud käskude komplekti kohta strateegialt uue oleku ning transleerivad oleku robotsüsteemi käskudeks. Avastaja on mitmelõimeline: iga sünkroniseeritud käskude komplekti kohta pannakse tööle ühes lõimes üks robotsüsteemile käskude saatja. Sünkroniseeritud käsud täidetakse nii, et iga käsukomplekt saadetakse korraga diskreetsetes olekutes ehk ei juhtu seda, et ühe rubriigi käsu täitmise ajal saadetakse teise rubriigi käsk, vaid oodatakse ära mõlema käsu täitmine ja siis saadetakse järgmised käsud. Kui käsud ei ole sünkroniseeritud, siis täidetakse neid ülejäänud süsteemi olekust sõltumatult ja järgmine käsk saadetakse kohe peale eelmise sama käsu täitmise lõppemist.

5.2 Mudeli looja

Mudeli loomisel on sisendiks logifail, logija, avastaja ja mudeli loomise konfiguratsioon. Logifailis on rida iga robotile saadetud sõnumi ja robotilt saadud vastuse kohta.

5.2.1 Konfiguratsioon

Mudeli loomise konfiguratsioonis saab määratleda ära mudeli loomisel olekuruumi vähendamise kordaja maksimaal- ja minimaalmäära (vt Lisa 2). Logija konfiguratsioonis saab iga sisendi juures määratleda ära väljad, mille järgi klasterdatakse, ning mudeli loomisel kasutatava käskude väljade väärtuste eraldusvõime, kuna Uppaal TRON toetab ainult täisarvulisi väärtusi. Samuti saab iga sisendi juures määrata, kas sisend on loodud Uppaal automaadis ajastatud ehk kas ühest olekust teise üleminekule on seatud ajapiirang. Ajapiirangule saab määrata konfiguratsioonis ka varu, mis liidetakse juurde logist võetud väärtuste põhjal arvutatud ajapiirangule.

Klasterdamiseks kasutatavate väljade valikus on muutujad, konstandid ning lisaks ka ajatemplid ja tagasiside. Tagasiside korral saab määrata, mis väärtusteks kodeeritakse õnnestumine ning ebaõnnestumine. Lisaks saab määrata millised väljad normaliseeritakse teiste väljade väärtuste järgi. Normaliseerimine on vajalik, kuna ühe välja väärtused võivad olla hoopis teises suurusjärgus kui mõne teise välja väärtused. Sellisel juhul oleks klasterdamisel ilma normaliseerimata suuremas suurusjärgus olevatel välja väärtustel suurem kaal. Joonisel 11 on näha näidet ühe sisendi mudeli looja konfiguratsioonist, kus on määratud, mille järgi klasterdamine toimub ning kas sisend on loodavas mudelis ajastatud ning mis on ajavaru.

```
cluster:
  by:
    variables: [0, 1]
    feedback:
      success: 1
      fail: 0
    timestamp: "feedback"
  normalise: ["feedback", "timestamp"]
model:
  timed: True
  timeBuffer: 40
```

Joonis 11. Klasterdaja konfiguratsiooni näide.

5.2.2 Ekvivalentsiklasside leidmine

Sünkroniseeritud sõnumid pannakse kokku ja leitakse ekvivalentsiklassid nendele korraga. Selle põhjuseks on, et ekvivalentsiklasside leidmisel peab arvestama terviklikku infot. Juhul, kui kahest käsust esimese käsu samade väärtuste juures üks kord teine käsk õnnestub ja teine kord ei õnnestu ning arvestada käskude väärtusi eraldi, peaks mõlemad esimese käsu väärtused ühte ekvivalentsiklassi panema, kui tegelt need pole võrdväärsed.

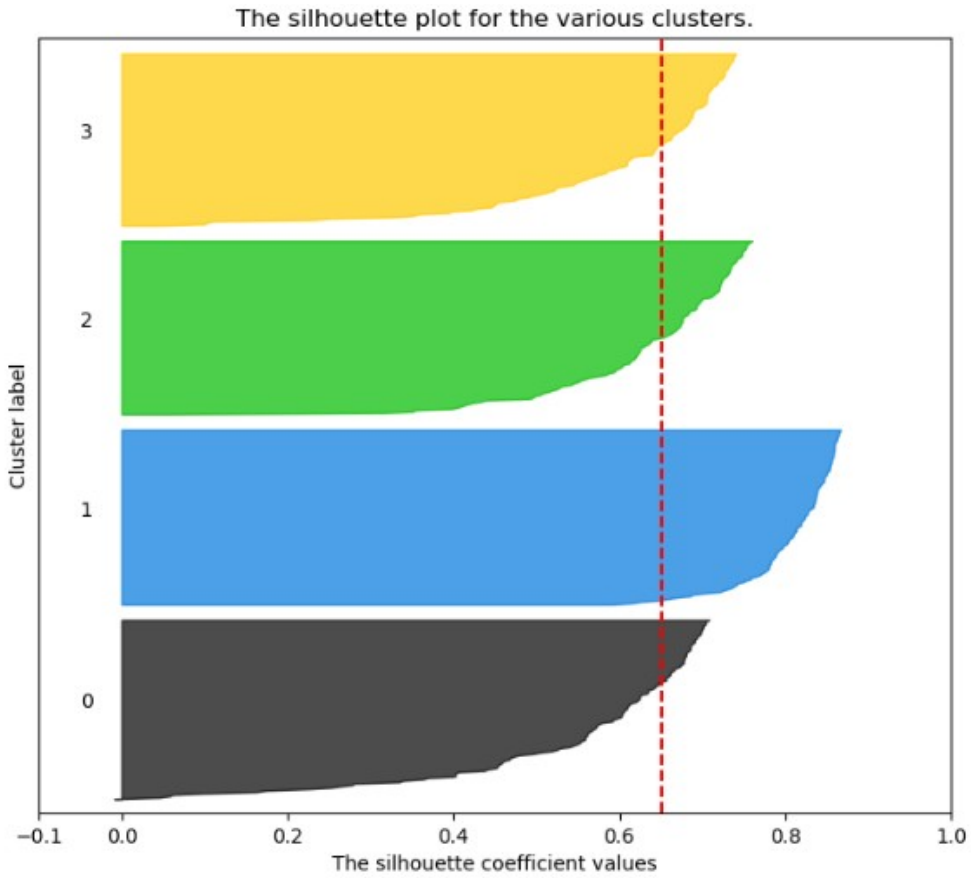
Ekvivalentsiklasside loomisel arvestatakse ainult käske, mida robot suutis täita. Sünkroniseeritud käskudel loetakse käsk täidetuks kui vähemalt üks neist õnnestus. Oletame, et on kaks käsku, mida saab robotile edastada, ning teise käsu õnnestumine sõltub esimese käsu väärtusest ning õnnestub ainult kindlates kohtades. Kui lugeda

sünkroniseeritud käske ainult siis õnnestunuks, kui kõik üksikud käsud õnnestuvad, siis läheks suur osa infost kaduma.

Ekvivalentsiklasside leidmine toimub kasutades juhendamata masinõpet, täpsemalt klasterdamist, kuna robotsüsteem on must kast ning pole teada midagi, mis aitaks meil otsuseid teha selle suhtes, millisesse ekvivalentsiklassi peaks mingi punkt kuuluma, peale saadetud käskude ise. Klasterdamiseks kasutatakse K-keskmiste algoritmi varianti. K-keskmiste algoritm on sobilik selleks ülesandeks, kuna selle hüperparameetriks on klastrite arv, mis võimaldab kasutajal määrata hüperparameetri vahemiku olekuruumi vähendamise kordades. Implementatsioonina kasutatakse *SciKit* teegi algoritmi *MiniBatchKMeans*. Hüperparameetri optimisatsiooniks kasutatakse kasutaja määratud klastrite arvu vahemikus siluettanalüüsi. Siluettanalüüs võimaldab antud kontekstis paremini kui teised hüperparameetri optimisatsiooni variandid leida avastaja tulemustest üles sobiliku suurusega klastrid. Teised optimisatsioonimeetodid kaldusid alati suuremate klastrite arvu poole ning ilma piiranguteta tekitasid kas ühe või kahe punkti suurused klastrid testitud olukordades.

K-keskmiste algoritm on klasterdamise algoritm, mis proovib minimiseerida punkti kaugust klastri tsentroidist. Teisisõnu, punkt kuulub klastrisse, kui see on kõige lähemal just selle klastri tsentroidile [11] . *MiniBatchKMeans* algoritm kasutab miniseeriaid, et vähendada K-keskmiste algoritmi tööaega, minimiseerides sama funktsiooni. Miniseeriad on juhuslikult valitud alamhulgad sisendandmetest, mis valitakse igal treenimise iteratsioonil [13] .

Siluettanalüüs võimaldab mõõta, kui kaugel on iga punkt klastris naaberklastrite punktidest ning seeläbi pakub viisi hinnata klasterdamise kvaliteeti [14] . Punkti siluettkordaja arvutatakse valemiga $\frac{b-a}{max(a,b)}$, kus a on keskmine klastrisisene kaugus ning b on keskmine lähima klastri kaugus [15] . Klasterdamise headuse hindamiseks arvutatakse kõigi klastrite siluettkordajate keskmine, ning mida lähemal väärtusele 1 see on, seda parem on klasterdamise tulemus [14] . Lisaks sellele võimaldab siluettanalüüsi graafik ka hästi inimesel hinnata klasterdamise kvaliteeti. Joonisel 12 [14] on näha suhteliselt hea tulemusega klasterdamise siluettanalüüsi graafikut.



Joonis 12. Näide siluettanalüüsist [14] .

Pärast sünkroniseeritud sõnumitele ekvivalentsiklasside leidmist analüüsitakse iga sünkroniseeritud käskude komplekti üksikuid käsu väärtusi eraldi. Kui esialgses leitud ekvivalentsiklassis on kõik käsud robotil õnnestunud täita, siis ei muudeta midagi. Muidu tõstetakse kõik õnnestunud käsud välja ülejäänud ekvivalentsiklassist eraldi ekvivalentsiklassi. See tagab kindluse, et samasse ekvivalentsiklassi ei satuks käsk, mis on ükskord õnnestunud ja teinekord pole õnnestunud, kui peaks juhtuma, et klasterdamise käigus pannakse need samasse klastrisse.

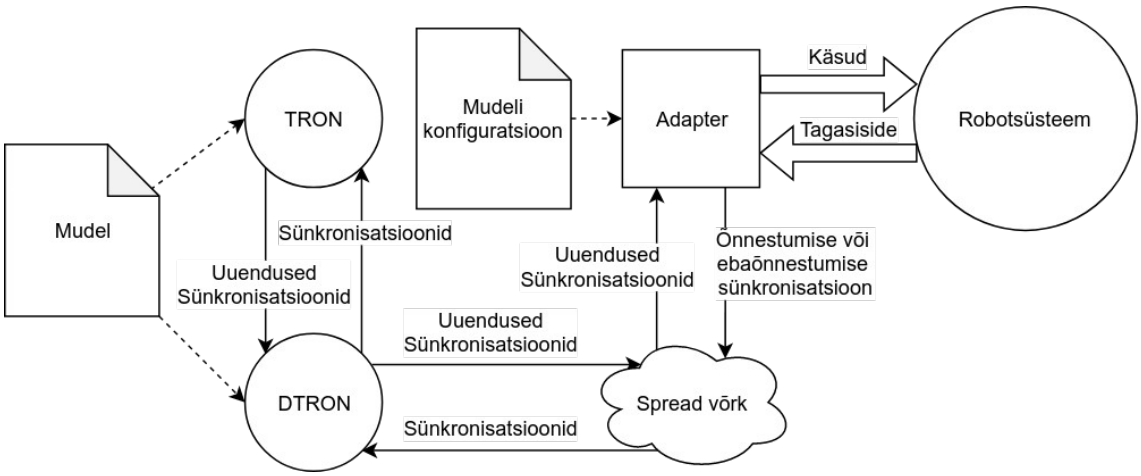
5.2.3 Uppaal mudeli loomine ja selle kasutamine testimiseks

Uppaal mudeli loomisel klastritest teisendatakse kõigepealt klastrid olekumasinaks, kus olekute väärtusteks saavad klastrite tsentroidid, ja seejärel teisendatakse see Uppaal automaadi kujule.

Ajapiirangud ühest olekust teise liikumisele ajastatud sisendite korral pannakse paika vastavalt tegelikule aset leidnud liikumisele vastavate klastrite punktide vahel. Kui selliseid liikumisi on rohkem kui üks, valitakse ajavahemikest maksimaalne. Muidugi ei

vasta aset leidnud liikumise ajavahemik klastrite tsentroidide vahelise liikumise ajavahemikule täielikult, kuid see annab hinnangu, mida kasutaja saab vastavalt korrigeerida, määrates konfiguratsioonis ajavaru, mis liidetakse ajavahemikule juurde. Ajavarust võib siin kontekstis mõelda kui kahekordsest keskmisest ajast klastri tsentrist äärde liikumiseks, kuna tõenäoliselt tegelik aset leidnud liikumine toimus klastri ääres olevate punktide vahel.

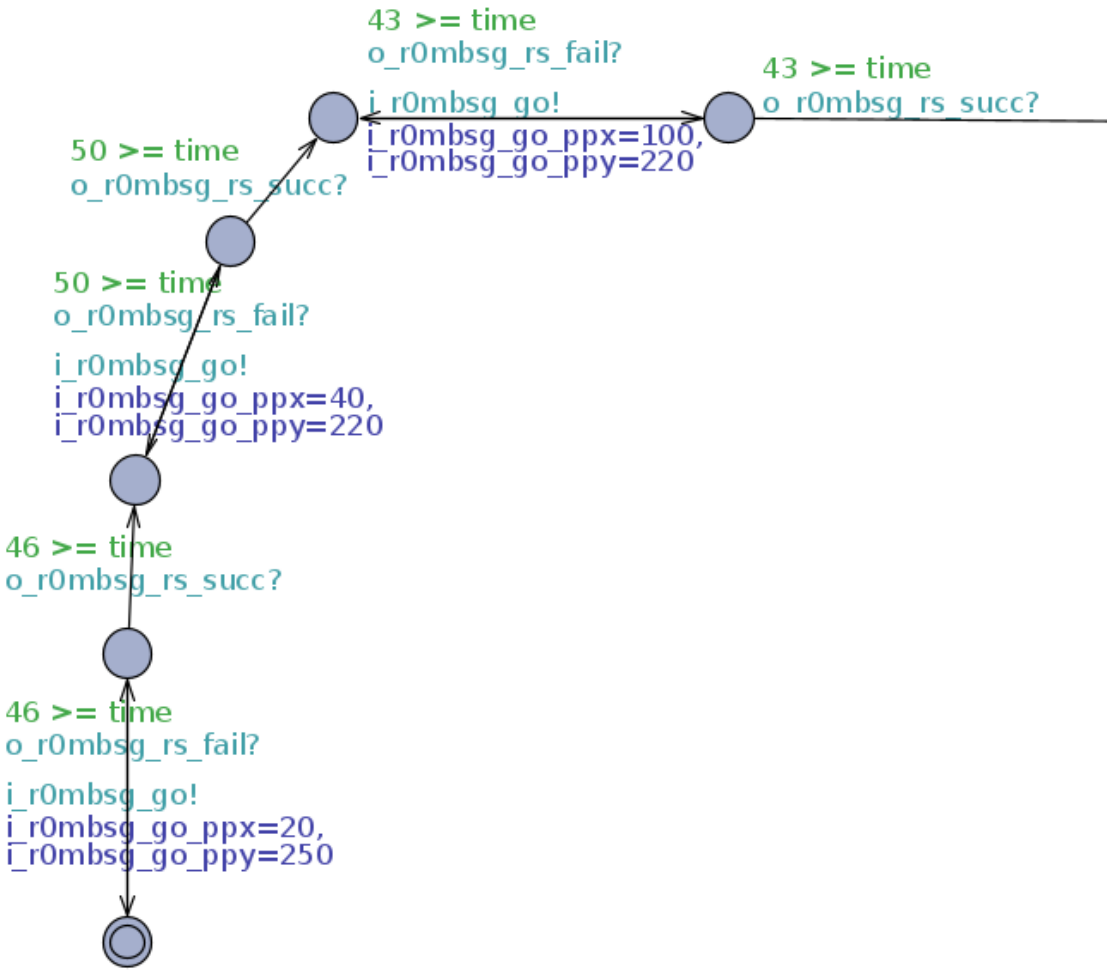
Igal Uppaal automaadi üleminekuga uude olekusse on seotud muutujate uuendused, mis saadetakse Uppaal TRON ja DTRON kaudu Spread võrgu kanalisse, mida kuulab loodud adapter, mis dekodeerib vastavalt mudeli looja väljastatud Uppaal automaadi konfiguratsioonile muutujad ning saadab muutujate väärtustega väärtustatud käsu edasi robotsüsteemile. Samuti kuulab adapter robotsüsteemi tagasisidet käsule ning edastab selle omakorda Spread võrgu kanalisse, mida kuulab DTRON ja mis edastab selle TRONile. Adapter saadab käsu õnnestumise korral sünkronisatsiooni, mis tähistab õnnestumist ja mille korral mudelis minnakse edasi järgmisesse olekusse. Ebaõnnestumise korral saadetakse sünkronisatsioon, mis tähistab ebaõnnestumist, ning mudelis ei minda sel korral uude olekusse. Nii õnnestumise kui ka ebaõnnestumise sünkronisatsioonil on peal ajapiirang, mis takistab edasi testimist, kui see ületatakse. Joonisel 13 on näha TRON, DTRON, Spread võrgu, adaptri ning robotsüsteemi omavahelist suhtluse skeemi.



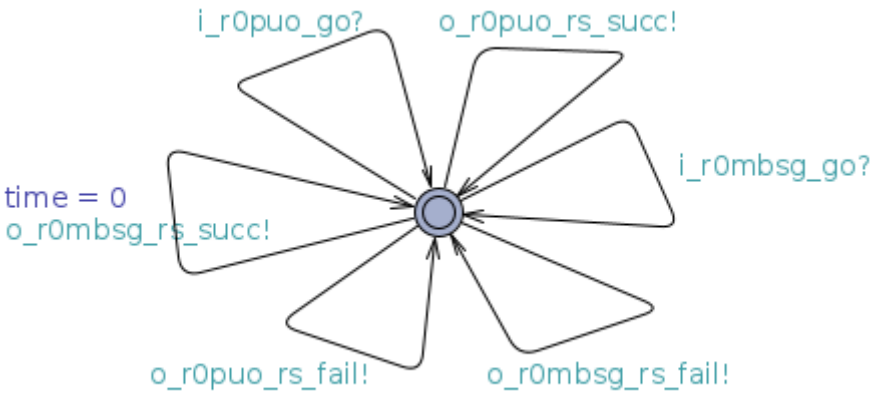
Joonis 13. TRON suhtlus robotsüsteemiga läbi vahekihtide ja adapteri.

Joonisel 14 oleval mudeli lõigul on näha, et õnnestumist tähistab sünkronisatsioon *o_r0mbsg_succ* ning ebaõnnestumist *o_r0mbsg_fail*. Lisaks on näha ajapiiranguid ning

muutujate uuendusi. Joonisel 15 on näha testitava süsteemi mudelit ning aja lähtestamist, mis on vajalik ajavahemike arvestamiseks.



Joonis 14. Näide mudeli looja poolt loodud olekumasina Uppaal mudeli lõigust.

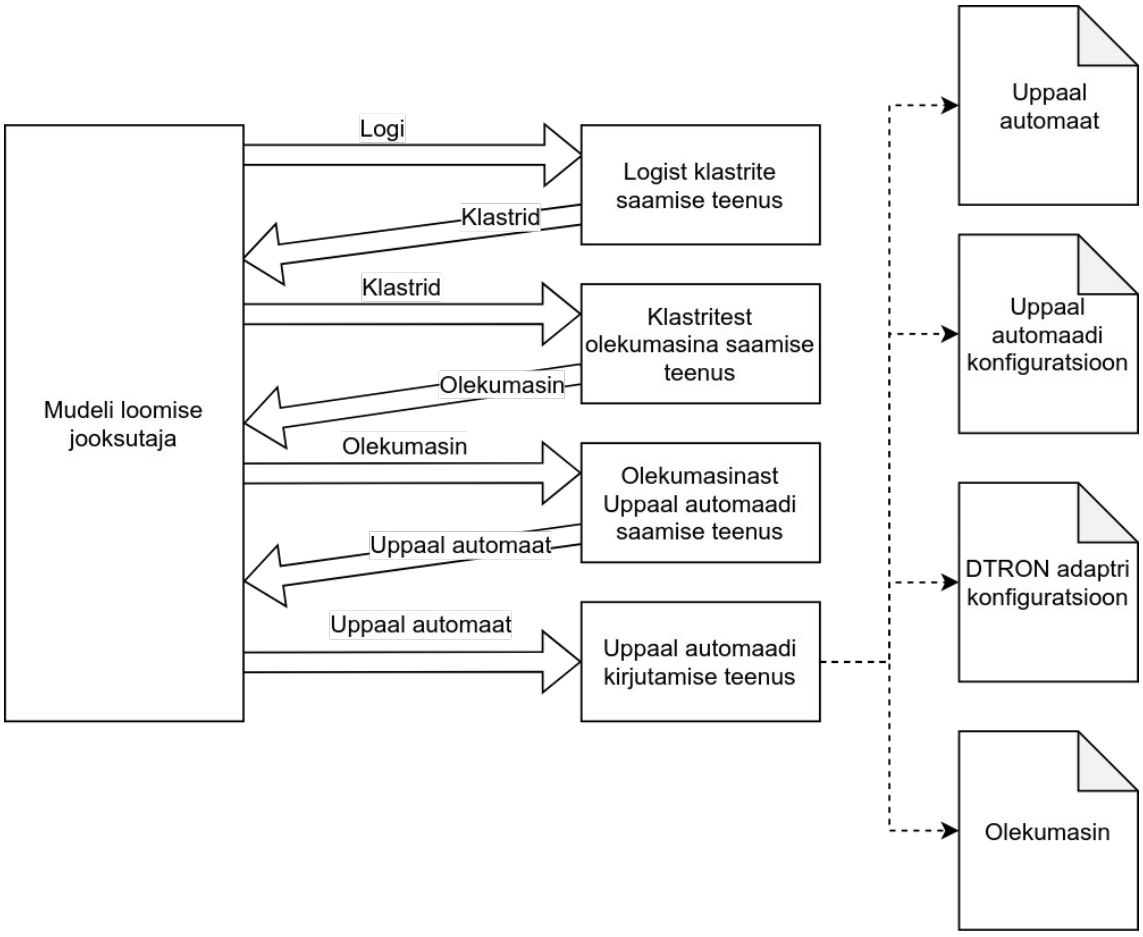


Joonis 15. Näide mudeli looja poolt loodud testitava süsteemi Uppaal mudelist.

Sünkroniseeritud käsud pannakse ühes Uppaal mudelis samasse malli, kuna need tuleb saata samal ajal. Iga sünkroniseerimata käsu ja sünkroniseeritud käskude komplekti kohta tehakse eraldi Uppaal mudel, kuna need käsud on järelikult sõltumatud teistest.

5.2.4 Konfigureeritavus

TestIt stsenaariumi konfiguratsioonis saab kergelt välja vahetada iga mudeli loomise komponendi osa. Mudeli loomise komponent on implementeeritud kahe osana: teenustena ja käivitatava programmina, mis kutsub neid teenuseid välja. ROS teenuseid on neli: logist klastrite saamise teenus, klastritest olekumasina saamise teenus, olekumasinast mudeli saamise teenus ning lõpuks ka mudeli kirjutamise teenus. Vaikimisi teenuse saab asendada lihtsalt stsenaariumi konfiguratsioonis spetsifitseerides teise teenuse aadressi (vaata Lisa 1). Joonisel 16 on näha, kuidas suhtlevad teenused ja mudeli loomise jooksutaja omavahel.



Joonis 16. Mudeli looja implementatsioon ROS teenustena.

5.3 Mudeli korrastaja

Mudeli korrastaja kasutab implementatsioonina olekuruumi avastajat ning asendab avastamise strateegia mudeli korrastamise strateegiaga. Mudeli korrastaja väljundiks on uus Uppaal automaat.

Mudeli korrastamise strateegia sisendiks on olekumasin, mida uuendatakse korrastamise käigus. Enne mudeli korrastamise alustamist leitakse olekute paarid, mille vahel võiks proovida uusi üleminekuid tekitada. Iga oleku kohta valitakse paariliseks talle lähim olek, millesse üleminekut ei ole.

Mudeli korrastamise käigus läbitakse mudel ning iga oleku juures proovitakse tekitada uus üleminek tema paariliseks olevasse olekusse ning seejärel tullakse tagasi ja jätkatakse mudeli läbimist. Kui mudel on läbitud, siis läbitakse mudel tagurpidi ehk suundutakse tagasi alguspunkti, et tekitada üleminekuid olekute vahel, kus muidu eksisteeris ainult ühesuunaline üleminek. Lisaks mõõdetakse iga ülemineku käigus üleminekuks kuluv aeg ja uuendatakse mudelis vastavalt ajapiirang. See võimaldab määrata ajapiiranguid täpsemalt, kui esialgse mudeli loomise käigus määrati. Ajapiirangute uuendamiseks võimaldatakse konfiguratsioonis spetsifitseerida eraldi ajavaru mudeli korrastamise käigus uuendatud ajapiirangute jaoks (vt Lisa 2).

Mudeli korrastaja strateegia saab kasutaja TestIt stsenaariumi konfiguratsioonis (vt Lisa 1) kergelt välja vahetada enda strateegia vastu.

5.4 TestIt integratsioon

TestIt käsurea tööriist võtab sisendina ühe režiimidest: test, avastaja, mudeli korrastamine või mudeli loomine (vastavalt *test*, *explore*, *refine-model*, *learn*) ning kutsub TestIt deemonis välja õige teenuse. TestIt deemoni poole peal käsitletakse testi, avastaja ja mudeli korrastamise režiime ühesugustena. Nende puhul pannakse käima SUTid (*System Under Test*) ning TestIt konteiner. Testimise režiimi puhul pannakse käima ainult kasutaja spetsifitseeritud käsk. Avastaja ja mudeli korrastaja puhul pannakse käima ka avastaja komponent. Mudeli korrastaja puhul pannakse käima ka mudeli loomise komponendi teenused, kuna avastaja komponent mudeli korrastaja režiimis pöördub lõpus olekumasinast mudeli loomise teenuse poole.

Mudeli loomise režiimi puhul pannakse käima ainult TestIt konteiner, kuna testitavaid süsteeme pole vaja selle jaoks käima panna. Samuti pannakse käima mudeli loomise komponendi teenused ning käivitaja, mis pöördub nende või kasutaja poolt spetsifitseeritud teenuste poole järjest.

Tööriist on integreeritud TestIt tööriistakomplekti nii, et toetab kõiki TestIt paralleelse jooksutamise võimalusi.

6 Tulemuste valideerimine

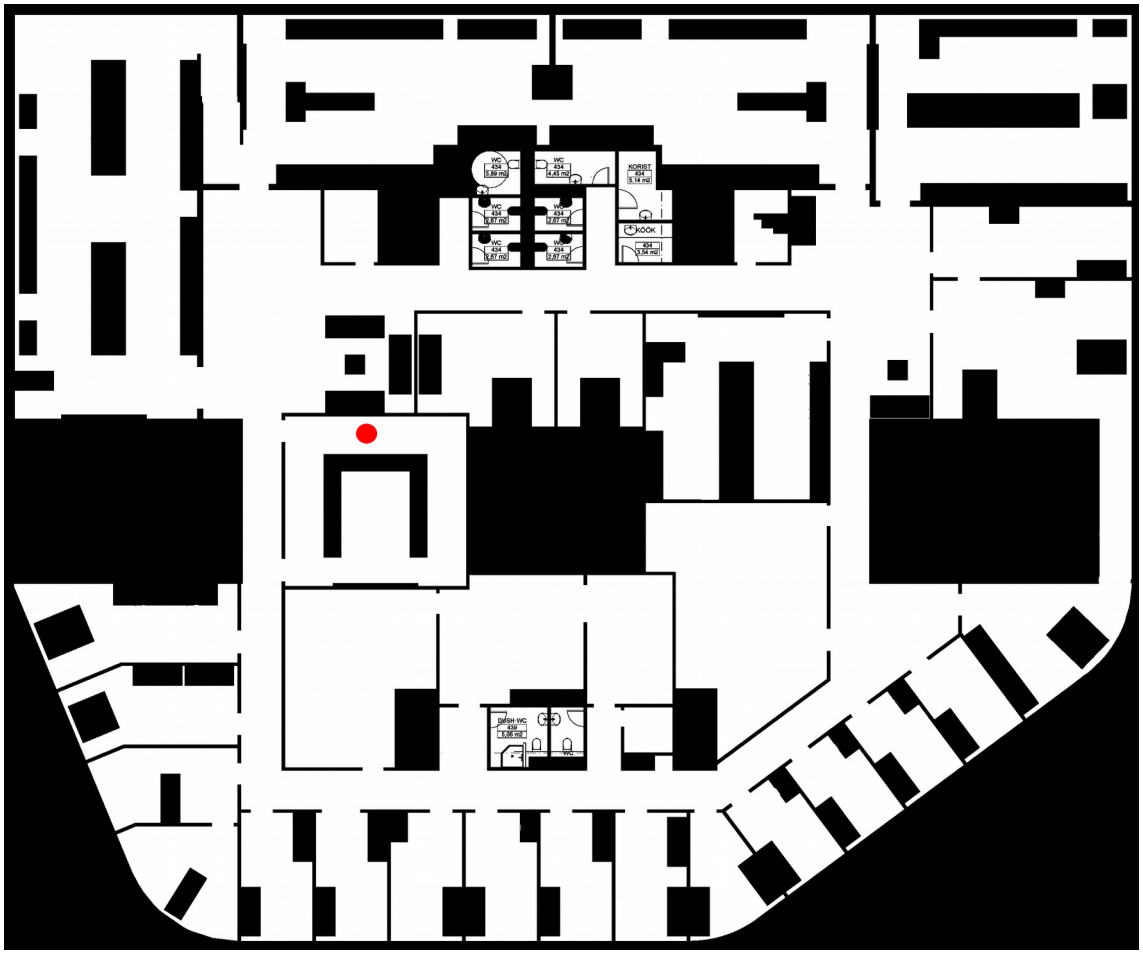
Valideerimiseks rakendatakse loodud süsteemi kõiki testimise faase robotsüsteemil simulatsioonis. Valideerimise eesmärk on tõestada, et implementeeritud süsteemi poolt loodud testmudeliga jõuab sama tulemuseni, mis käsitsi loodud mudeliga, ehk mõlema mudeliga leitakse üles koodikatet maksimeerivad testistsenaariumid.

Koodikatet maksimeeritakse, kuna see annab indikatsiooni sellest, kui suur osa funktsionaalsusest sai testitud. TestIt teek võimaldab ka teisi parameetreid mõõta ja optimeerida, mis võivad mõnes konkreetses kasutusmallis olulised olla.

6.1 Patrullroboti kasutusmall

Valideerimiseks kasutatav multirobotsüsteem koosneb kahest patrullrobotist ja ühest sissetungijast ning kaardist, mida on näha joonisel 17 [10] . Patrullrobotid sõidavad ettenähtud marsruuti pidi ning testmudel juhib sissetungijat. Kindlas kohas kaardil, mis on alloleval joonisel punasega märgendatud, saab sissetungija panna maha potentsiaalselt ohtliku tundmatu objekti. Patrullrobotitel on võimekus tuvastada tundmatut objekti. Tundmatu objekti tuvastamine on stsenaarium, mis tõstab koodikatet üle tavalise lävendi. Seetõttu on eesmärgiks mõlema mudeliga luua selline stsenaarium, kus tekib olukord, et patrullrobotil on võimalus tuvastada tundmatut objekti, testimaks seda funktsionaalsust.

Nagu eelnevalt mainitud, siis patrullrobotid sõidavad ettemääratud marsruuti pidi ja neid ei saa dünaamiliselt juhtida. Sissetungijal on kaks käsutüüpi, mida sellele saab saata: üks, millele saab ette anda positsiooni koordinaatides ning robot proovib liikuda kaardil sinna positsiooni, ning teine, mis on antud positsioonis tundmatu objekti maha panemise käsk. Tundmatu objekti saab ainult ühe korra maha panna ning seda ainult kindlas positsioonis kaardil. Joonisel 17 on tundmatu objekti maha panemise asukoht tähistatud punase täpiga.



Joonis 17. Simulatsioonis kasutatud kaart [10] koos tundmatu objekti asukohaga.

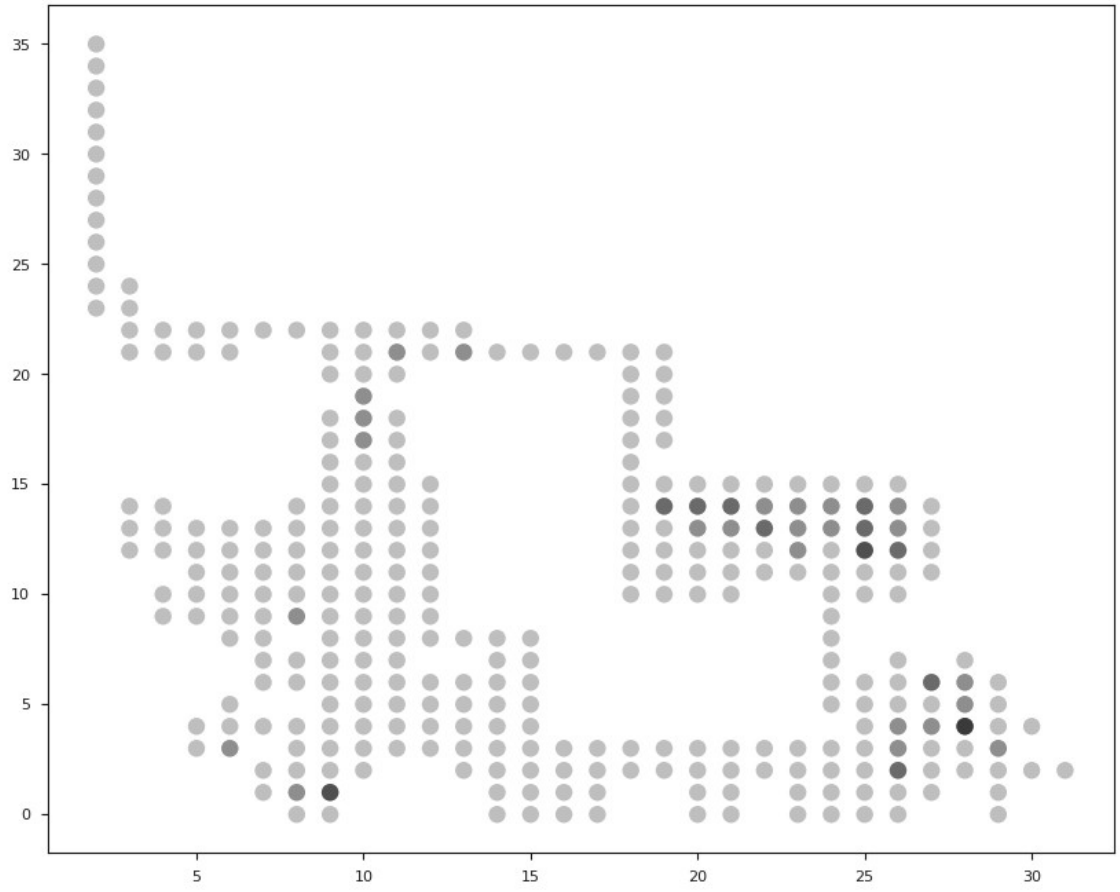
Tehniliselt koosneb süsteem neljast Dockeri konteinerist: TestIt konteiner, sissetungija roboti konteiner ning kaks patrullroboti konteinerit. Simulatsiooni jooksutab *Stage* simulaator [10] .

6.2 Olekuruumi avastamine

Olekuruumi kaardistamiseks antakse avastajale ette logi konfiguratsioon koos avastaja konfiguratsiooniga (vt Lisa 2), kus on määratletud viidatud kaks käsku: navigatsiooni käsk ja tundmatu objekti maha panemise käsk. Navigatsiooni käsu sammuks x ja y suunas on 1 punkt. Ülejäänud käsu väljad on määratud konstantideks. Objekti maha panemise käsus on määratletud ainuke väli tõese konstandina, mis annab teada sissetungijale, et panna maha tundmatu objekt. Samuti on määratletud tagasiside rubriigid mõlema käsu puhul, et kas käsu täitmine oli edukas või mitte. Konfiguratsioonis olid määratud sünkroniseeritud käskudeks mõlemad kasutatud käsud

ehk objekti prooviti maha panna igas erinevas positsioonis, kuhu robot jõudis, aga mitte vahepealsetes positsioonides.

Olekuruumi avastamine oli edukas antud ajaraamis. Avastaja kaardistas suhteliselt laia ala ning leidis üles koha, kus sai tundmatu objekti maha panna. Allolevalt jooniselt 18 on näha avastatud olekuruum ning ka tumedamad alad, kus käidi mitu korda.



Joonis 18. Avastatud olekuruum.

6.3 Mudeli loomine

Mudeli loomiseks kasutati kaardistamisest tekkinud logikirjeid. Logikirjetes on kirjas nii käsud, mis saadeti, kui ka tagasiside. Joonisel 19 on näha kahte logikirjet, millest esimeses on kirjas saadetud tundmatu objekti mahapanemise käsk ning teises selle käsu tagasiside vastavalt logija konfiguratsioonile (vt Lisa 2).

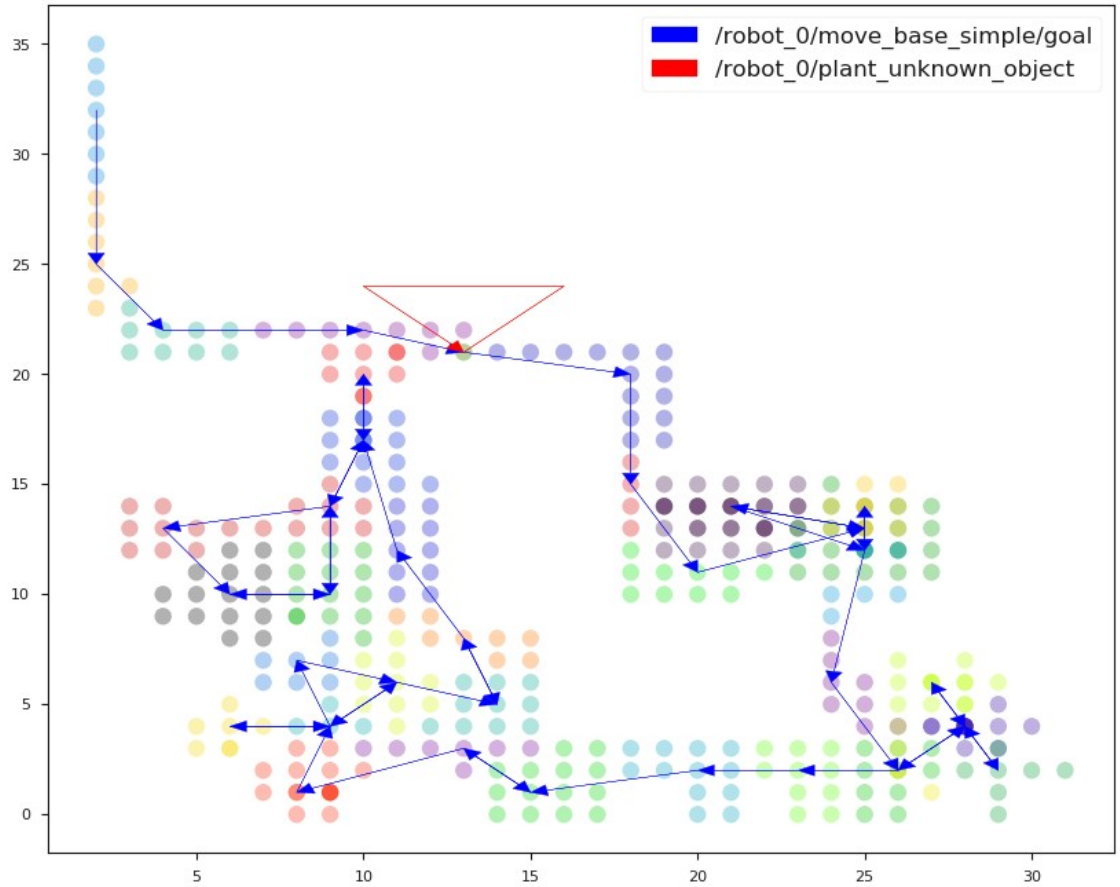
```
{
  "run_id": "...",
  "timestamp": 54.6,
  "coverage": {...},
  "test": "T1",
  "data": {
    "data": true
  },
  "event": "POST",
  "channel": {
    "identifier":
"/robot_0/plant_unknown_object",
    "type": "std_msgs.msg.Bool",
    "proxy": ""
  }
}

{
  "run_id": "...",
  "timestamp": 55.6,
  "coverage": {...},
  "test": "T1",
  "data": {
    "data": false
  },
  "event": "RESPONSE",
  "channel": {
    "identifier": "/unknown_object/planted",
    "type": "std_msgs.msg.Bool",
    "proxy": ""
  }
}
```


Joonis 19. Logikirjete näide.

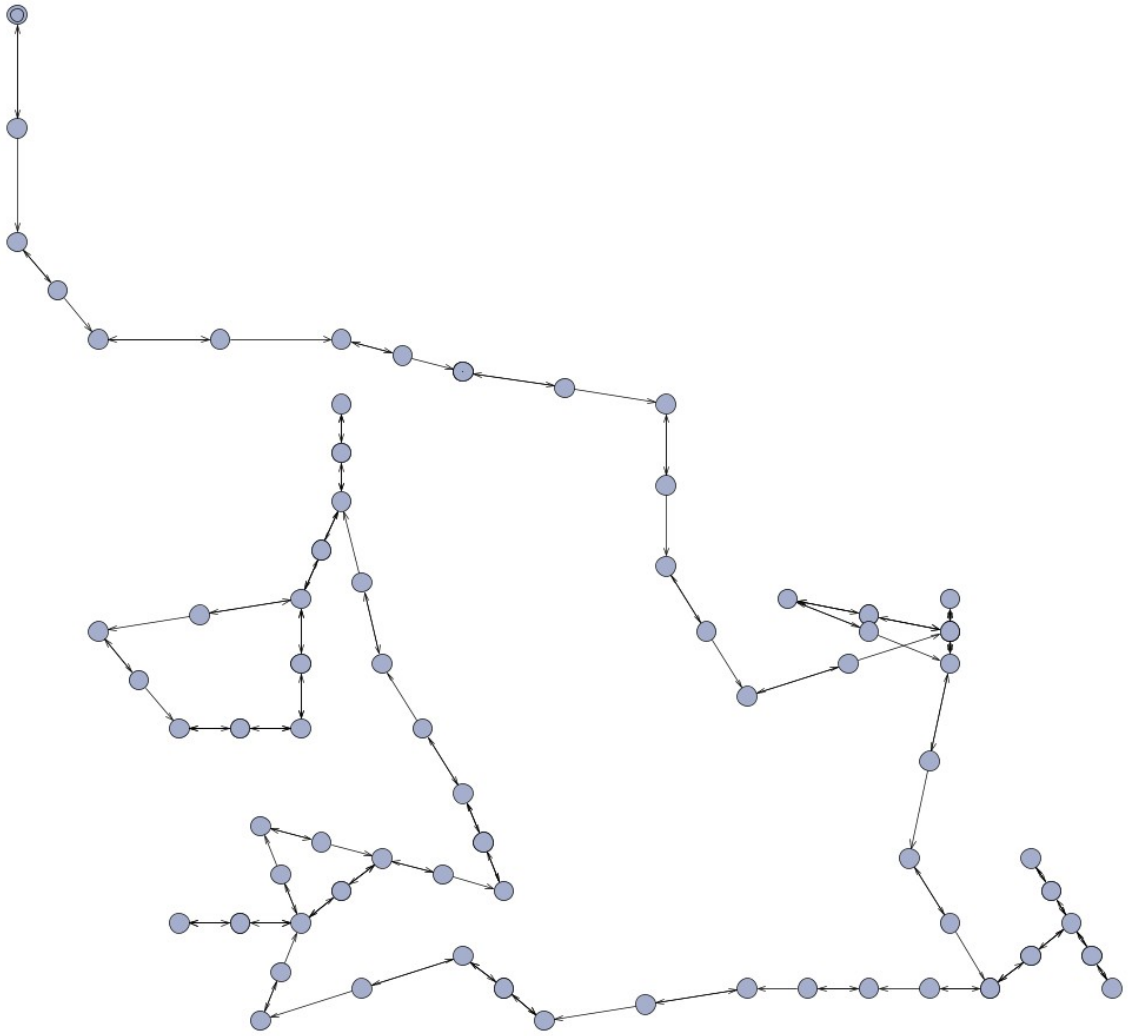
Mudeli loomise konfiguratsioonis oli määratud olekuruumi vähendamise kordadeks 5 kuni 10 korda. Navigatsiooni käsu puhul oli määratud, et klasterdatakse kõigi käskude muutujate järgi, s.t x ja y koordinaadi järgi. Objekti maha panemise käsu puhul oli määratud, et klasterdatakse tagasiside, kus õnnestumine kodeeritakse ühena ja ebaõnnestumine nullina, ning ajatempli järgi. Ajatempli lisamine teiseks argumendiks on vajalik, et klasterdada teise klastrisse need kohad, kuhu jõudmiseks läks kauem aega.

Oletame, et robot valib järgmiseks olekuks punkti, mis on teisel pool seina. Sellisel juhul peaks kuuluma see punkt, mis on teisel pool seina, teise ekvivalentsiklassi, kui see punkt, mis on samas ruumis. Normaliseerimise alla kuuluvateks väljadeks olid määratud tagasiside ja ajatempel. Ajastatud oli sisenditest ainult navigatsiooni käsk, kuna tundmatu objekti maha panemine toimib kohe käsu saatmisel ning seetõttu pole selle puhul aeg oluline.

Mudeli loomine toimis, eraldi klastrisse jäi punkt, kus sissetungija pani maha tundmatu objekti. Allolevalt jooniselt 20 on näha punktide klasterdamise tulemust värvide järgi ning olekumasinat. Joonisel 21 on näha olekumasinast loodud Uppaal automaati ilma siltideta.



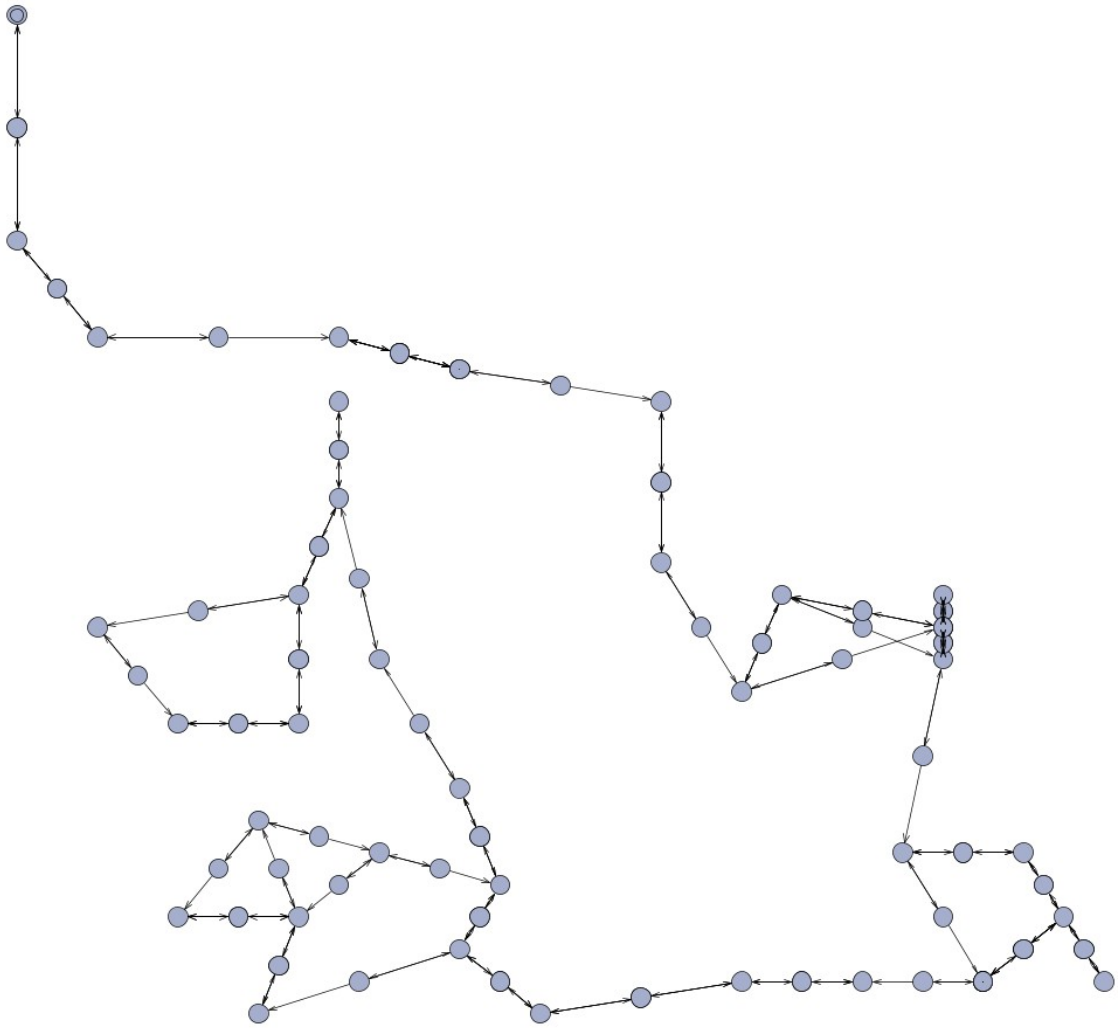
Joonis 20. Olekuruumi punktid, nendest loodud klastrid ning olekumasin.



Joonis 21. Olekumasinast loodud Uppaal mudel ilma siltideta.

6.4 Mudeli korrastamine

Mudeli korrastamise käigus leiti palju uusi üleminekuid lähedaste olekute vahel, mis lubab testijal rohkem valikuid teha ning ka vähendab tupikusse jäämise võimalusi. Uued üleminekud on ka tihti vastupidised esialgsetele üleminekutele, et saaks liikuda tagasi, kui jäädakse tupikusse. Alloleval joonisel 22 on näha uusi üleminekuid korrastatud mudelis.



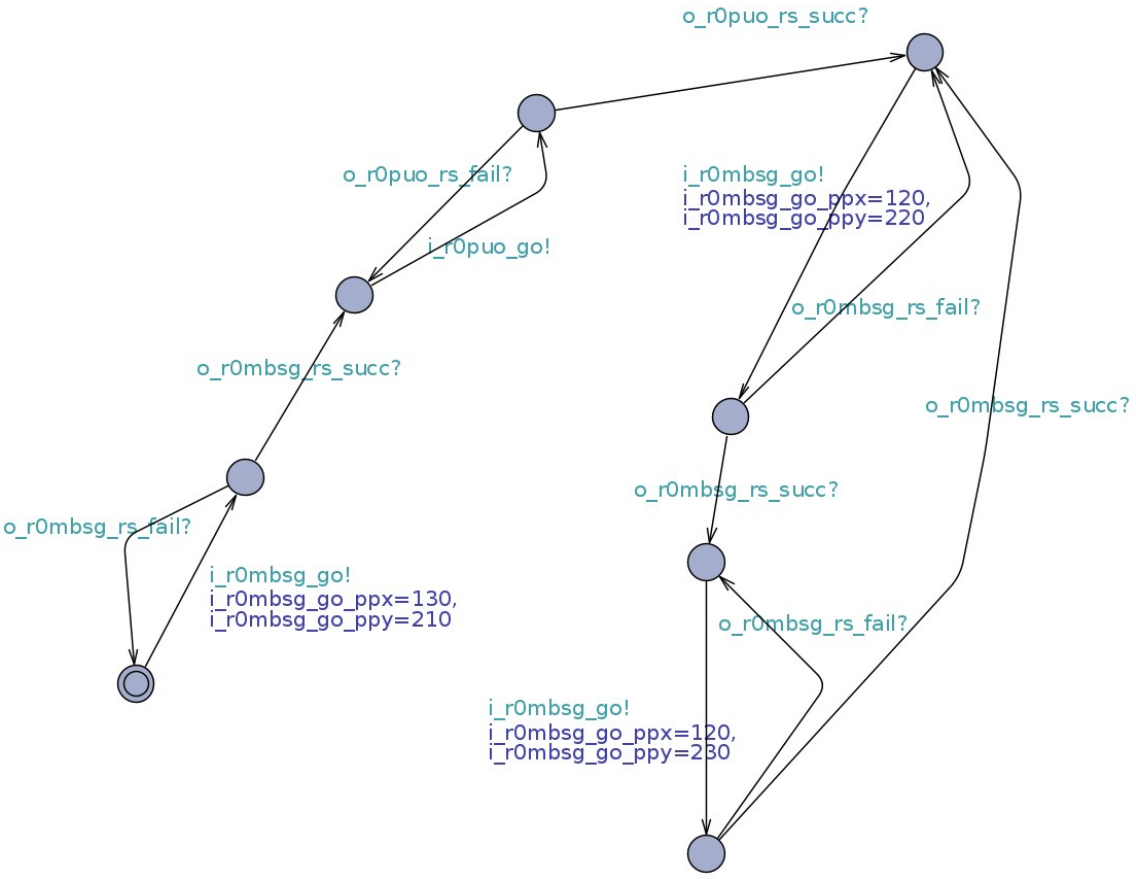
Joonis 22. Mudeli korrastamise käigus loodud mudel.

6.5 Testimine mudelite põhjal

Mudeli põhjal testimiseks jooksutati Uppaal TRONi ja DTRONi koos loodud adapteriga, mille tulemusel saadeti robotsüsteemile käske mudeli alusel.

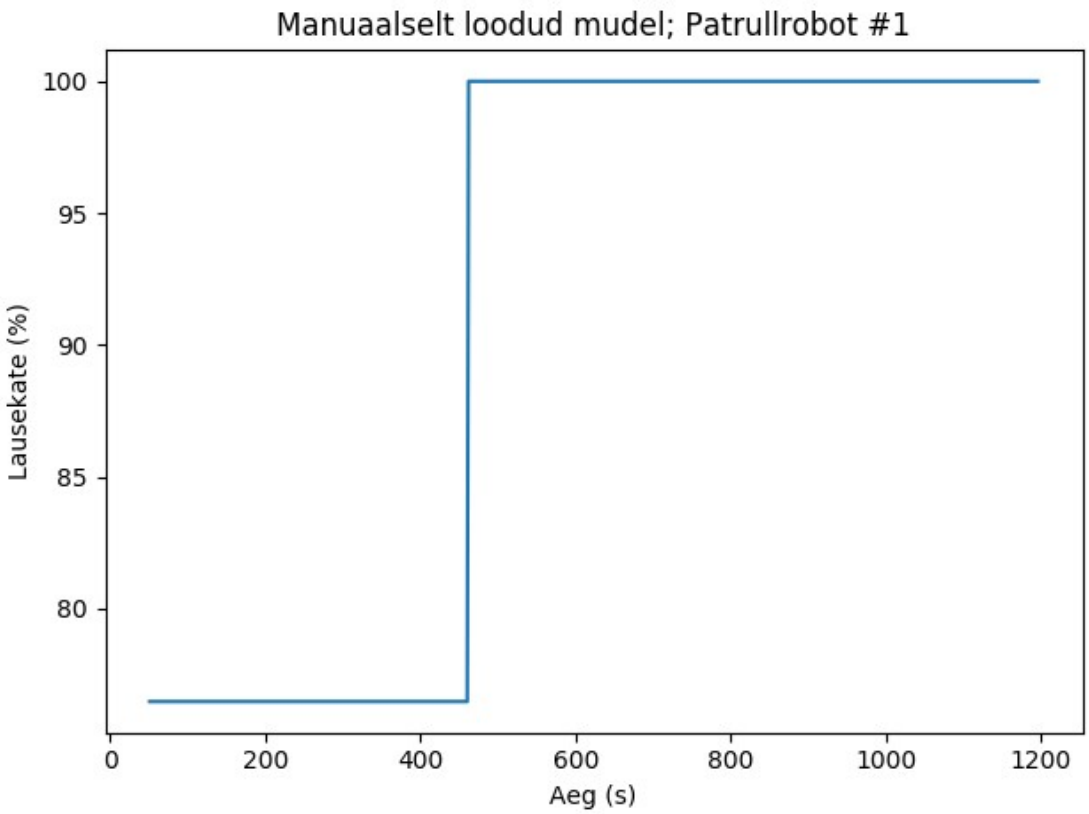
6.5.1 Käsitsi loodud mudeli põhjal testimine

Käsitsi loodud mudel, mida on näha joonisel 23, oli antud stsenaariumi testimiseks lihtne: sissetungija läheb otse tundmatu objekti maha panemise kohta ning paneb objekti seal maha ning jääb kõrval edasi tagasi liikuma. Eesmärgist lähtuvalt pole käsitsi loodud mudelil ajapiiranguid.

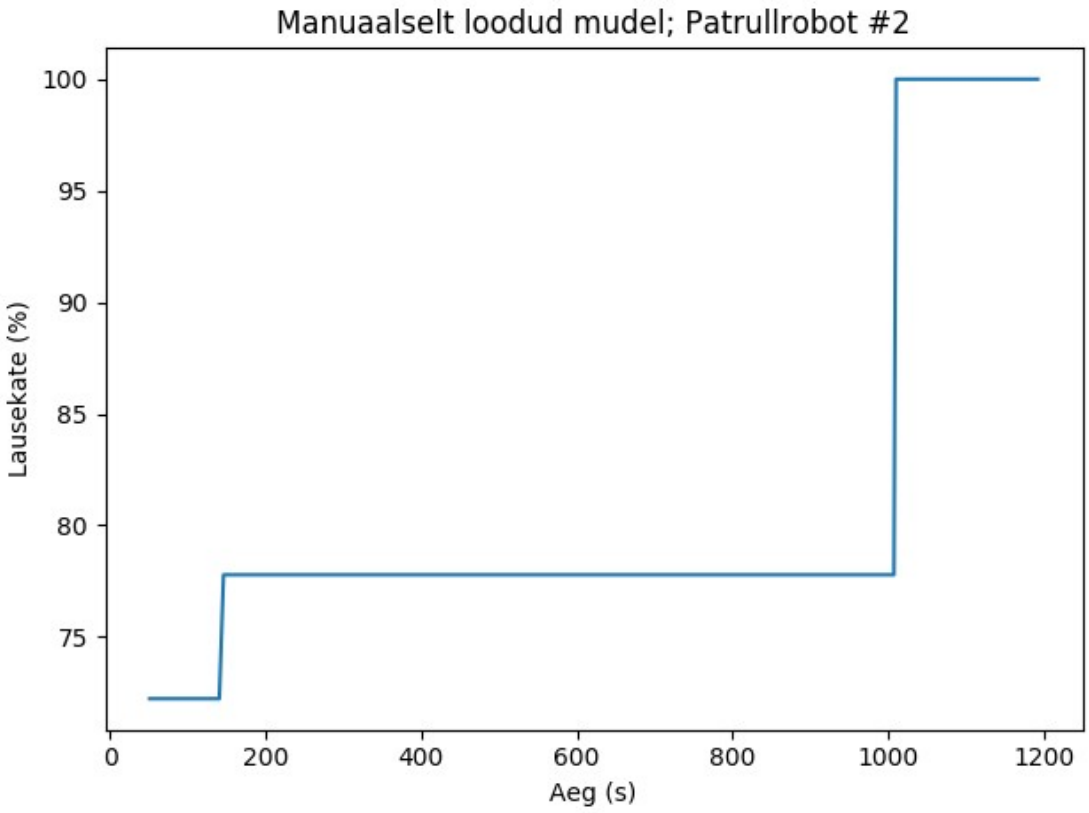


Joonis 23. Käsitsi loodud mudel robotsüsteemi testimiseks.

Allolevalt graafikutelt 24 ja 25 on *y*-teljel lausekate ning *x*-teljel aeg. Nagu graafikutelt näha, siis käsitsi loodud mudeliga saavutasid robotid testitavas failis sajaprotsendilise koodiridade katvuse ehk leidsid testi käigus üles tundmatu objekti.



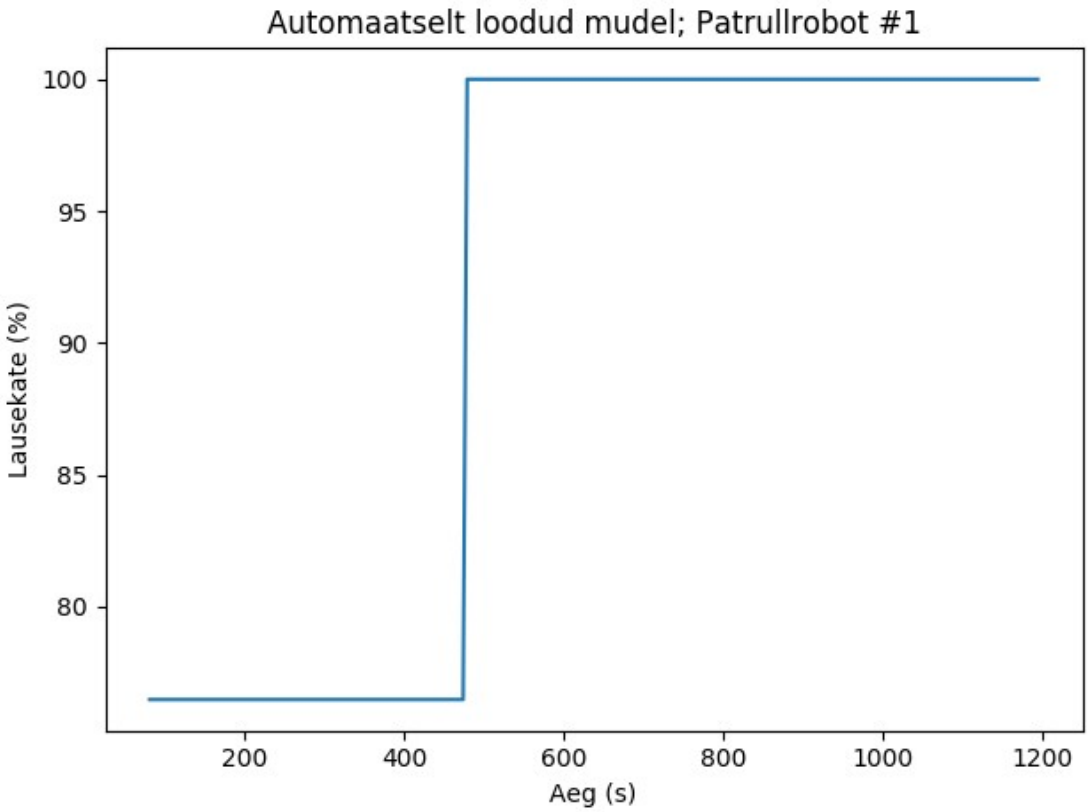
Joonis 24. Lausekate vastavalt ajale käsitsi loodud mudeliga esimese patrullroboti juhul.



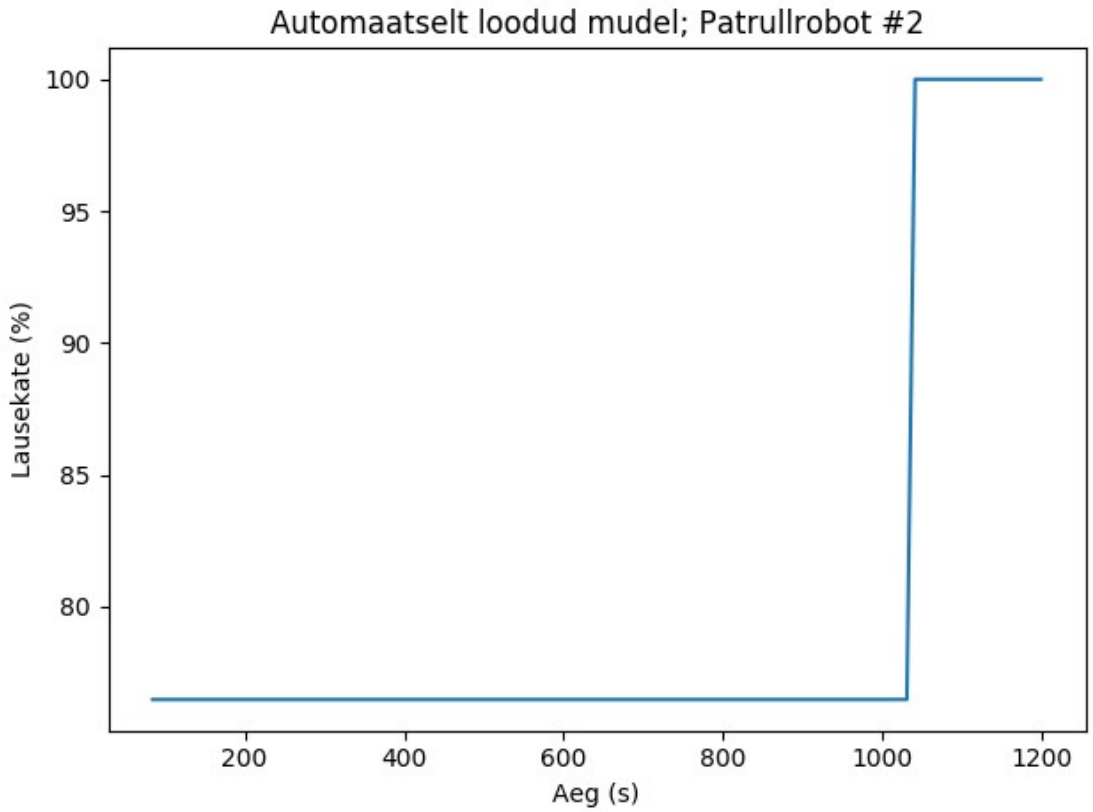
Joonis 25. Lausekate vastavalt ajale käsitsi loodud mudeliga teise patrullroboti juhul.

6.5.2 Automaatselt loodud mudeli põhjal testimine

Sarnaselt käsitsi loodud mudeliga testimisele, õnnestus ka automaatselt loodud mudeliga saavutada maksimaalne koodiridade katvus. Allolevatelt joonistelt 26 ja 27 on näha, et mõlemal patrullrobotil õnnestus leida üles tundmatu objekt ning see tuvastada.



Joonis 26. Lausekate vastavalt ajale automaatselt loodud mudeliga esimese patrullroboti juhul.



Joonis 27. Lausekate vastavalt ajale automaatselt loodud mudeliga teise patrullroboti juhul.

6.6 Tulemuste analüüs

Antud kasutusmallis õnnestus automaatselt loodud mudeliga sama edukalt leida üles koodikatet maksimeerivad kohad, kui käsitsi loodud mudeliga. Samuti on näha, et koodikate suurenes suhteliselt samal ajal nii automaatselt loodud mudeli kui ka käsitsi loodud mudeliga, mis tähendab seda, et nii käsitsi kui ka automaatselt loodud mudeli järgi juhitav sissetungija jõudis tundmatu objekti panna maha enne, kui patrullrobotid sinna jõudsid.

Saavutatud tulemused tõestavad, et leidub juhte, mille korral on võimalik testida robotsüsteeme, kasutades loodud süsteemi ehk õppides robotsüsteemi mudel automaatselt ja kasutades seda robotsüsteemi testimiseks. Kuna ehitatud süsteem on väga paindlik ja konfigureeritav ning võimaldab paljusid loogilisi süsteemi osi ROS vahendite abil lihtsalt välja vahetada, on loodud süsteemil potentsiaali leida kasutust paljude robotsüsteemide testimisel.

Kui automaatselt ehitatud mudel ei võimalda kõiki juhte testida, saab seda kohendada kas konfiguratsiooni muutes või käsitsi. Mõlemal juhul hoiab automaatne mudeli loomine palju aega kokku ja säästab arendusressurssi.

7 Kokkuvõte

TestIt on tööriistakomplekt, mis võimaldab automaatset mudelipõhist robotsüsteemide testimist. Töös tutvustatakse loodud laiendit TestIt tööriistakomplektile, mis võimaldab automaatselt õppida robotsüsteemide mudeleid Uppaal automaadi kujul. Mudeli automaatne loomine säästab testimis- ja arendusprotsessis palju aega. Tööriistad on implementeeritud kasutades ROSi ja võimaldavad testida seega robotsüsteeme, mis implementeerivad ROS suhtluse liideseid.

Tööriist võimaldab avastada robotsüsteemi olekuruumi vastavalt konfiguratsioonile, selle suurust vähendada läbi klasterdamise ning ehitada tulemustest Uppaal ajaga automaat. Avastamine toimub kasutades sügavuti otsingut ning valides võimalike punktide seast järgmiseks kaugeima läbitud teekonnast. Avastaja konfiguratsioon võimaldab määrata avastamise strateegia parameetrid ROS rubriikide abil ning spetsifitseerida avastamise käigu sammude pikkused iga rubriigi iga välja kohta. Avastaja konfiguratsioon võimaldab määrata ka sünkroniseeritud sisendid, kuhu saadetakse käsud sünkroniseeritult. Olekuruumi suuruse vähendamine toimub läbi olekute klasterdamise, kasutades K-keskmiste algoritmi varianti. Konfiguratsioonis saab määrata väljad, mille järgi klasterdatakse, sisendid, mis on ajastatud, olekuruumi suuruse vähendamise kordaja vahemiku minimaalse ja maksimaalse määra ning ajavaru, mis liidetakse tegelikule kulunud olekute ülemineku ajale juurde. Optimaalse klastrite arvu leidmiseks rakendatakse siluettanalüüsi konfiguratsioonis määratud klastrite arvu vahemikus. Sünkroniseeritud sisendid klasterdatakse koos ja pannakse ühte Uppaal mudelisse. Iga sünkroniseerimata sisendite komplekti kohta luuakse eraldi Uppaal mudel. On olemas ka mudeli korrastamise funktsionaalsus, mis leiab lähimate olekute vahel uusi üleminekuid ning lisaks uuendab ajapiiranguid mudelis. Lisaks konfigureeritavusele on võimalik kergelt asendada mudeli loomise, avastaja ning mudeli korrastaja komponendid spetsifitseerides konfiguratsioonis kasutaja enda loodud ROS teenused.

Tööriista valideerimiseks rakendati loodud funktsionaalsust kahest patrullrobotist ja ühest sissetungija robotist ning kaardist koosnevale simulatsioonile. Eesmärgiks oli testida patrullrobotite tundmatu objekti tuvastamise funktsionaalsust läbi sissetungija roboti juhtimise. Sissetungija robot pidi maha panema tundmatu objekti ning patrullrobotid pidid selle tuvastama. Loodud tööriist lõi mudeli, mis saavutas sama testitavuse ehk koodikatte taseme kui käsitsi loodud mudel. Tulemus tõestab, et loodud tööriist võimaldab testida robotsüsteeme või vähemalt anda genereeritud mudeli näol alguspunkt robotsüsteemide testimiseks.

Kasutatud kirjandus

[1] Abbas, H., O'Kelly, M., Rodionova, A., et al. Safe At Any Speed: A Simulation-Based Test Harness for Autonomous Vehicles. – *Cyber Physical Systems. Design, Modeling, and Evaluation, Lecture Notes in Computer Science*, 2018, vol 11267, lk 94–106.

[2] About ROS. [WWW] https://www.ros.org/about-ros/ (28.04.2020)

[3] Aichernig, B. K., Mostowski W., Mousavi M. R., et al. Model Learning and Model-Based Testing. – *Machine Learning for Dynamic Software Analysis: Potentials and Limits, Lecture Notes in Computer Science*, 2016, vol 11026, lk 74-100.

[4] Anier, A., Vain, J. Model based continual planning and control for assistive robots. – *HEALTHINF 2012: Proceedings of the International Conference on Health Informatics*, SciTePress, 2012.

[5] Behrmann G., David A., and Larsen K. A Tutorial on Uppaal 4.0. 2006. [WWW] https://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf (29.04.2020)

[6] Benveniste A., Caillaud B., et al. Contracts for Systems Design: Theory. – *Rennes Bretagne Atlantique*, *INRIA*, 2015, RR-8759. [WWW] https://hal.inria.fr/hal-01178467/document (27.04.2020)

[7] Deshmukh J., Horvat M., Jin X., et al. Testing cyber-physical systems through Bayesian optimization. – *ACM Transactions on Embedded Computing Systems*, 2017, 16(5s), lk 1–18. [WWW] https://doi.org/10.1145/3126521 (30.04.2020)

[8] Hessel, A. Larsen, K. G., Mikucionis M., et al. Testing Real-Time Systems Using UPPAAL. – *Formal Methods and Testing, Lecture Notes in Computer Science*, 2008, vol 4949, lk 77-117. [WWW] https://people.cs.aau.dk/~marius/tron/FMT2008.pdf (02.05.2020)

[9] Kalra, N., Paddock S. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? – *Transportation Research Part A: Policy and Practice*, 2016, vol 94, lk 182–193.

[10] Kanter, G., Vain, J. Model-based testing of autonomous robots using TestIt. – *J Reliable Intell Environ*, 2020, vol 6, lk 15–30. [WWW] https://doi.org/10.1007/s40860-019-00095-w (28.04.2020)

[11] Piech, C. Stanford CS221: K Means. 2013. [WWW] https://stanford.edu/~cpiech/cs221/handouts/kmeans.html (30.04.2020)

[12] ROS Concepts. [WWW] http://wiki.ros.org/ROS/Concepts (28.04.2020)

[13] SciKit learn: Mini Batch K-Means. [WWW] https://scikit-learn.org/stable/modules/clustering.html#mini-batch-kmeans (30.04.2020)

[14] SciKit learn: Selecting the number of clusters with silhouette analysis on Kmeans clustering. [WWW] https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html (30.04.2020)

[15] SciKit learn: Silhouette samples. [WWW] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_samples.html#sklearn.metrics.silhouette_samples (01.05.2020)

[16] Uppaal TRON: Introduction. [WWW] https://people.cs.aau.dk/~marius/tron/introduction.html (30.04.2020)

[17] Utting M, Pretschner A, Legeard B. A taxonomy of model based testing approaches. – *Software Testing, Verification & Reliability*, 2012, vol 22(5), lk 297–312. [WWW] https://doi.org/10.1002/stvr.456 (02.05.2020)

[18] Utting, M., Legeard, B. Practical Model-Based Testing: A Tools Approach. USA, California : Morgan-Kaufmann, 2007.

Lisa 1 – TestIt stsenaariumite konfiguratsioonid

```yaml
jobs:
  - tag: "T1"
    mode: "explore"
    concurrency: 1
    credits: 1
    pipeline: ""
    verbose: False
    moveStrategyService: ""
    moveStrategyInitTopic: ""
    loggerConfiguration:
"testit_tests/fixtures/logger.yaml"
    preLaunchCommand: "mv [[sharedDirectory]]
[[resultsDirectory]]/logger.log
[[sharedDirectory]][[resultsDirectory]]/pre-
[[tag]]-[[testUuid]]-logger.log || true"
    launch: "(rostopic pub -l /robot_1/enable
std_msgs/Bool \"data: true\" &); (rostopic pub
-l /robot_2/enable std_msgs/Bool \"data: true\"
&)"
    oracle: ""
    timeout: 86400
    timeoutVerdict: False
    postCommand: "echo 'Explore finished!'"
    bagEnabled: False
    bagMaxSplits: ""
    bagDuration: ""

  - tag: "T1"
    mode: "learn"
    concurrency: 1
    credits: 1
    pipeline: ""
    verbose: False
```

```
    preLaunchCommand: "mv [[sharedDirectory]]
[[resultsDirectory]]/logger.log
[[sharedDirectory]][[resultsDirectory]]/pre-
[[tag]]-[[testUuid]]-logger.log; cp
[[sharedDirectory]][[resultsDirectory]]/persist
ent/logger_finished.log [[sharedDirectory]]
[[resultsDirectory]]/logger.log; true"
    loggerConfiguration:
"testit_tests/fixtures/logger.yaml"
    stateMachineToUppaalService: ""
    logToClusterService: ""
    clusterToStateMachineService: ""
    writeUppaalService: ""
    launch: ""
    oracle: ""
    timeout: 86400
    timeoutVerdict: False
    postCommand: "echo 'Learn finished!'"
    bagEnabled: False
    bagMaxSplits: ""
    bagDuration: ""

  - tag: "R1"
    mode: "refine-model"
    continuousUpdate: true
    concurrency: 1
    credits: 1
    pipeline: ""
    verbose: False
    moveStrategyService: ""
    moveStrategyInitTopic: ""
    modelName:
"testit_tests/results/persistent/T1r0mbsgr0puo"
    preLaunchCommand: "(mv [[sharedDirectory]]
[[resultsDirectory]]/logger.log
[[sharedDirectory]][[resultsDirectory]]/pre-
[[tag]]-[[testUuid]]-logger.log); (cp
[[sharedDirectory]][[modelName]].yaml
[[sharedDirectory]][[modelName]]-
refined_model.yaml); (cp [[sharedDirectory]]
[[modelName]]-adapter_config.yaml
[[sharedDirectory]][[modelName]]-refined-
adapter_config.yaml); true"
    loggerConfiguration:
"testit_tests/fixtures/logger.yaml"
```

```
    stateMachine: "[[modelName]]-
state_machine.json"
    stateMachineToUppaalService: ""
    launch: ""
    oracle: ""
    timeout: 86400
    timeoutVerdict: False
    postCommand: "echo 'Refine model
finished!'"
    bagEnabled: False
    bagMaxSplits: ""
    bagDuration: ""

  - tag: "T2"
    mode: "test"
    concurrency: 1
    credits: 1
    pipeline: ""
    verbose: False
    preLaunchCommand: "mv [[sharedDirectory]]
[[resultsDirectory]]/logger.log
[[sharedDirectory]][[resultsDirectory]]/pre-
[[tag]]-[[testUuid]]-logger.log || true"
    uppaalModel:
"testit_tests/results/persistent/R1r0mbsgr0puo-
refined_model.xml"
    loggerConfiguration:
"testit_tests/fixtures/logger_detection.yaml"
    adapterConfiguration:
"testit_tests/results/persistent/R1r0mbsgr0puo-
refined-adapter_config.yaml"
    adapterLaunch:
"launch/testit_adapter.launch"
    modelConfiguration:
"testit_tests/results/persistent/R1r0mbsgr0puo-
refined_model.yaml"
```

```
    launch: "(roslaunch testit_dtron_adapter
testit_adapter.launch
configuration:=\"[[sharedDirectory]]
[[adapterConfiguration]]\"
adapter_launch_file:=\"[[sharedDirectory]]
[[adapterLaunch]]\"
model_configuration:=\"[[sharedDirectory]]
[[modelConfiguration]]\" &); (sleep 10 &&
rostopic pub -l /robot_1/enable
std_msgs/Bool \"data: true\" &); (sleep 10 &&
rostopic pub -l /robot_2/enable
std_msgs/Bool \"data: true\" &); (sleep 10 &&
rosrun testit_dtron run_dtron_test.sh
[[sharedDirectory]][[uppaalModel]])"
    oracle: ""
    timeout: 86400
    timeoutVerdict: False
    postCommand: "echo 'Test finished!'"
    bagEnabled: False
    bagMaxSplits: ""
    bagDuration: ""

  - tag: "T3"
    mode: "test"
    concurrency: 1
    credits: 1
    pipeline: ""
    verbose: False
    preLaunchCommand: "mv [[sharedDirectory]]
[[resultsDirectory]]/logger.log
[[sharedDirectory]][[resultsDirectory]]/pre-
[[tag]]-[[testUuid]]-logger.log || true"
    uppaalModel:
"testit_tests/results/persistent/handmade_model
.xml"
    loggerConfiguration:
"testit_tests/fixtures/logger_detection.yaml"
    adapterConfiguration:
"testit_tests/results/persistent/R1r0mbsgr0puo-
refined-adapter_config.yaml"
    adapterLaunch:
"launch/testit_adapter.launch"
    modelConfiguration:
"testit_tests/results/persistent/handmade_model
.yaml"
```

```
    launch: "(roslaunch testit_dtron_adapter
testit_adapter.launch
configuration:=\"[[sharedDirectory]]
[[adapterConfiguration]]\"
adapter_launch_file:=\"[[sharedDirectory]]
[[adapterLaunch]]\"
model_configuration:=\"[[sharedDirectory]]
[[modelConfiguration]]\" &); (sleep 10 &&
rostopic pub -l /robot_1/enable
std_msgs/Bool \"data: true\" &); (sleep 10 &&
rostopic pub -l /robot_2/enable
std_msgs/Bool \"data: true\" &); (sleep 10 &&
rosrun testit_dtron run_dtron_test.sh
[[sharedDirectory]][[uppaalModel]])"
    oracle: ""
    timeout: 86400
    timeoutVerdict: False
    postCommand: "echo 'Test finished!'"
    bagEnabled: False
    bagMaxSplits: ""
    bagDuration: ""

  - tag: "T4"
    mode: "test"
    concurrency: 1
    credits: 1
    pipeline: ""
    verbose: False
    loggerConfiguration:
"testit_tests/fixtures/logger.yaml"
    launch: "(rostopic pub -l /robot_1/enable
std_msgs/Bool \"data: true\" &); (rostopic pub
-l /robot_2/enable std_msgs/Bool \"data: true\"
&); rosrun testit testit_runner.py _filename:=/
testit/testit_tests/results/persistent/logger_t
ron_finished.log
_weights:=/testit/testit_tests/fixtures/weights
.yaml _test:=T2 _selection_mode:=0"
    oracle: ""
    timeout: 600
    timeoutVerdict: False
    postTestSuccessCommand: "echo 'postTest
success command'"
    postTestFailureCommand: "echo 'postTest
failure command'"
```

```
    postTestCommand: "echo 'postTest command;
UUID is [[testUuid]]'"
    postCommand: "echo 'Test finished!'"
    bagEnabled: False
    bagMaxSplits: ""
    bagDuration: ""
```

Lisa 2 – Logija, avastaja ja klasterdaja konfiguratsioon

```yaml
configuration:
  syncedExploreTopics:
    - [0, 1]
  clusterReductionFactor:
    max: 10.0
    min: 5.0
  modelTimeUnitInMicrosec: 1000000
  coverage:
    enable: True
    mode: "msg"
    reportingTimeLimit: 2.0
  inputs:
    - identifier:
"/robot_0/move_base_simple/goal"
      proxy: ""
      type: "geometry_msgs.msg.PoseStamped"
      timeout: 120
      explore:
        continue: true
        variables:
          - field: "pose.position.x"
            step: 1
            initial: 2
          - field: "pose.position.y"
            step: 1
            initial: 36
        constants:
          - field: "pose.orientation.z"
            value: 0
          - field: "pose.orientation.w"
            value: 1
          - field: "header.frame_id"
            value: "map"
      cluster:
```

```
        resolution: 0.1
      model:
        timed: True
        timeBuffer: 40
        timeBufferForRefineModel: 20
      feedback:
        topic: "/robot_0/move_base/result"
        type:
"move_base_msgs.msg.MoveBaseActionResult"
        field: "status.text"
        success: "Goal reached."
        failure: ""
    - identifier:
"/robot_0/plant_unknown_object"
      proxy: ""
      type: "std_msgs.msg.Bool"
      timeout: 120
      explore:
        continue: true
        goals:
          file: ""
        constants:
          - field: "data"
            value: true
      cluster:
        by:
          feedback:
            success: 1
            fail: 0
          timestamp: "feedback"
        normalise: ["feedback", "timestamp"]
      model:
        timed: False
      feedback:
        topic: "/unknown_object/planted"
        type: "std_msgs.msg.Bool"
        field: "data"
        success: true
```