TALLINNA TEHNIKAÜLIKOOL
MEHAANIKATEADUSKOND

Mehhatroonika instituut
Mehhatroonikasüsteemide õppetool

MHK70LT

Artur Peedimaa

TÖÖSTUSLIKU ROBOTPLATVORMI NAVIGATSIOONI SIMULEERIMINE

Autor taotleb
tehnikateaduse magistri
akadeemilist kraadi

Tallinn
2014

AUTORIDEKLARATSIOON

Deklareerin, et käesolev lõputöö on minu iseseisva töö tulemus.
Esitatud materjalide põhjal ei ole varem akadeemilist kraadi taotletud.
Töös kasutatud kõik teiste autorite materjalid on varustatud vastavate viidetega.

Töö valmis teaduri Robert Hudjakov ja professori Mart Tamre juhendamisel

“.....”.....201….a.

Töö autor

..... allkiri

Töö vastab magistritööle esitatavatele nõuetele.

“.....”.....201….a.

Juhendaja

..... allkiri

Lubatud kaitsmisele.

..... eriala/õppekava kaitsmiskomisjoni esimees

“.....”.....201… a.

..... allkiri

TTÜ Mehhatroonikainstituut

Mehhatroonikasüsteemide õppetool

MAGISTRITÖÖ ÜLESANNE

2014. aasta kevadsemester

Üliõpilane: Artur Peedimaa, 104738MAHM
Õppekava: MAHM02/09
Eriala: Mehhatroonika
Juhendaja: Robert Hudjakov (teadur)
Mart Tamre (professor)

MAGISTRITÖÖ TEEMA:
(eesti keeles) Tööstusliku robotplatvormi navigatsiooni simuleerimine
(inglise keeles) Industrial robot platform navigation simulation

Lõputöös lahendatavad ülesanded ja nende täitmise ajakava:

Nr	**Ülesande kirjeldus**	**Täitmise tähtaeg**
1	Kasutatavate tööstuslike robotplatvormide ja nende navigeerimis/juhtimisvõimaluste ülevaade.	**1.04**
2.	Närvivõrkude kasutamisvõimalused tööstuslike robotplatvormide simulatsioonirakendustes (ülevaade, analüüs) Robotino XT kasutamine simulatsioonirakenduse riistavaralise baasina.	**15.04**
3.	Võimalikud lahendusnäited. Näidete kirjeldus ja käitumise/kasutamise analüüs. Loodud/modifitseeritud tarkvara kirjeldus.	**1.05**
4.	Edasised võimalikud arendused/ideed tööstusliku robotplatvormi loomiseks. Võimalikud rakendused teistes valdkondades.	**10.05**
5.	Ohutuse analüüs robotplatvormi kasutamise kohta tulevikus tööstuses.	**16.05**

Lahendatavad insenertehnilised ja majanduslikud probleemid:
Objekte iseseisvalt tuvastava ja teekonda planeeriva tööstusliku robotplatvormi navigeerimis- ja juhtimissüsteemi loomine.

Töö keel: eesti

Kaitsmistaotlus esitada hiljemalt 12.05.14 **Töö esitamise tähtaeg 22.05.14**

Üliõpilane: Artur Peedimaa /allkiri/ …… kuupäev……

Juhendaja: Robert Hudjakov /allkiri/ ……. kuupäev……

Juhendaja: Mart Tamre /allkiri/ ……. kuupäev……

Konfidentsiaalsusnõuded ja muud ettevõttepoolsed tingimused formuleeritakse pöördel

SISUKORD

EESSÕNA

Antud lõputöö teema pakuti välja Tallinna Tehnikaülikooli mehhatroonikainstituudi direktori professori Mart Tamre poolt. Tööks vajalike andmete kogumine ja katsete tegemine toimus TTÜ Mehhatroonikainstituudis. Katsete tegemiseks kasutatud väljakut ja robotit aitasid üles panna ja esmaseadistada Ülari Lees ja Raul Kerme. Töö tegemisel abistasid konsultatsioonidega teadur Robert Hudjakov ja professor Mart Tamre. Töö autor tänab eespool mainitud isikuid.

LÜHENDITE LOETELU

ROS - Robotic Operating System

AGV - Automated Guided Vehicle

NFN - Natural Feature Navigation

OpenCV - Open Source Computer Vision

USB - Universal Serial Bus

IEEE - Institute of Electrical and Electronics Engineers

RGB - Red, Green, Blue

RGBA - Red, Green, Blue, Alpha

JPEG - Joint Photographic Experts Group

PNG - Portable Network Graphics

SISSEJUHATUS

Tänapäeva tööstuses on aina olulisem roll automatiseerimisel, mis aitab kokku hoida kulusid ning suurendada tootlikkust ja efektiivsust. Automatiseerimise eesmärgil kasutatakse esemete transportimisel – peamiselt tööstus- ja laohoonetes – autonoomseid robotplatvorme (ingl *Automated Guided Vehicle*). Taoliste robotplatvormide kasutamine toob aga endaga kaasa mitmeid väljakutseid. Robot peab oskama kindlaks teha oma asukohta ruumis ning vajadusel seda ka korrigeerida, kui tema liikumisel tekib kõrvalekalle planeeritud teekonnast. Lisaks tuleb arvestada ka takistustega, mis võivad roboti teekonnale tekkida, ja võimalusega teekonda ümber planeerida, et platvorm uude sihtpunkti liiguks. Tänapäeval kasutatakse platvormide juhtimiseks erinevaid, kuid enamasti siiski algelisi lahendusi, mis ei arvesta kõikide nimetatud kriteeriumitega ja lisaks võivad teatud lahendused seada spetsiifilisi tingimusi objektide (masinad, pingid, alused, riiulid jms) paiknemisele hoones. Nimetatud põhjused on ka antud töö kirjutamise ajendiks.

Töö peamine eesmärk on välja selgitada tööstusliku robotplatvormi navigeerimis- ja juhtimissüsteemi loomise võimalikkust, kus robotplatvorm on võimeline oma teekonnal takistusi tuvastama ning neid ka vältima või äärmuslikus olukorras täiesti uue teekonna sihtpunktini välja arvutama. Üheks loodava süsteemi tingimuseks on seatud, et süsteem peab olema kohanemisvõimeline, st lisaks takistuste vältimisele oskama ka ümber arvutada planeeritud teekonda või sihtpunkti asukohta. Olulisel kohal on ka kohanemiseks kuluv aeg, mis ei tohi ületada olemasolevate lahenduste ajakulu sarnases olukorras. Samuti ei tohiks süsteemi ülesseadmine olla olemasolevate lahendustega võrreldes keerukam. Vältimaks platvormi kõrvalekaldumist planeeritud teekonnast, mis võib tekkida liikumisandurite ebatäpsusest, peab olemas olema täiendav informatsiooni roboti asukoha kohta, mille abil vajadusel korrigeerida roboti teekonda.

Ühe võimaliku lahendusena taolise süsteemi loomiseks pakubki antud töö välja monokulaarse kaamera kasutamise pealtvaates pildi edastamiseks, st kaamera asetseb lae all ning jälgib maapinnal toimuvat. Kaamerat kasutades on võimalik kindlaks teha roboti positsioon, orientatsioon, tuvastada takistusi ja vaba põrandapinda ning antud andmeid kasutades arvutada platvormi läbimiseks kõige optimaalsem teekond. Takistuste vältimiseks, muutlikes tingimustes toimetulemiseks ning teekonna arvutamiseks peab süsteem aga teadma, millised on takistused ning millised on need objektid, mille juures platvorm peatuma peab, ning kui need samad objektid ruumis ümber tõsta, siis suutma neid objekte jätkuvalt tuvastada. See tähendab, et süsteem peab olema õppimisvõimeline. Süsteemi õppimisvõime tagatakse

tehisnärvivõrkude kasutamisega.

Tehisnärvivõrkude kasutamine eeldab siiski süsteemi eelnevat treenimist, st keskkonnas paiknevad objektid ja takistused tuleb kõigepealt ära klassifitseerida ning selgeks õpetada. Treenitud närvivõrku saab aga hiljem kasutada nende samade või sarnaste objektide tuvastamiseks, isegi kui nende asukoht keskkonnas (tööstushoones) on muutunud. See võimaldab väikese vaevaga planeerida robotplatvormile uut teekonda. Antud töös välja pakutud pealtvaatega kaameral, mis roboti positsiooni ja objektide kohta pidevalt infot registreerib, ning tehisnärvivõrkudel põhineval lahendusel võiks olla mitmeid eeliseid hetkel kasutatavate lahenduste ees.

Tehisnärvivõrkude sobivuse väljaselgitamiseks antud ülesande lahendamiseks on kasutatud objektide klassifikaatorit. Klassifikaator on kirjutatud programmeerimiskeeles C# ning selleks kasutatati arenduskeskkonda Microsoft Visual Studio.

Simulatsioonirakenduse riistvaralise baasina on kasutatud Festo mobiilset robotplatvormi Robotino XT. Robotino XT ongi loodud just arendus- ja õppe-eesmärgil ning toetab seetõttu erinevaid programmeerimiskeeli: C, C++, Java, Matlab, LabView, ROS, .Net, Microsoft Robotics Developer Studio. Kuna varasemaid kokkupuuteid ROS'i (Robotic Operating System) kasutamisel roboti juhtimise ja navigeerimise programmeerimiseks on instituudis olnud vähe, valitigi antud ülesande lahendamiseks just see tarkvararaamistik. ROS pakub erinevaid teeke ja tööriistu, mis lihtsustavad rakenduste loomist. ROS ise toetab programmeerimiseks kaht erinevat keelt: Python ja C++. Kuna mainitud kahest keelest on autor varasemalt kasutanud ainult C++'i, kasutatigi just seda keelt. Kuigi ROS toetab teoreetiliselt mitmeid erinevaid operatsioonisüsteeme, siis ametlik tugi on ainult Unix-põhisel operatsioonisüsteemil Ubuntu. ROS'i kasutamiseks installeeriti arvutisse 32-bitine operatsioonisüsteem Ubuntu 12.04 LTS ning C++ keeles programmeerimiseks kasutati arenduskeskkonda Qt Creator.

Magistritöö jaguneb viieks suuremaks osaks. Kõigepealt antakse põhjalikum ülevaade olemasolevatest tööstuslike robotplatvormide juhtimislahendustest (ptk 1). Teises peatükis tutvustatakse alternatiivset lahendust tööstuslike robotite juhtimiseks ning tuuakse välja selle eelised olemasolevate lahenduste ees (ptk 2.1). Kirjeldatakse süsteemi ülesehitust (ptk 2.2) ning räägitakse töö teoreetilistest, täpsemalt tehisnärvivõrkude kasutamisest pilditöötluseks (ptk 2.3), ROS-i kasutamisest robotplatvormi juhtimiseks (ptk 2.4) ning roboti asukoha jälgimisest (ptk 2.5). Kolmandas peatükis kirjutatakse katseteks kasutatud süsteemist (ptk 3.1) ning erinevate katsete tulemustest: uuritakse klassifikaatori võimekust (ptk 3.2), teekonna

planeerimise toimivust ROS-is (ptk 3.3) ning roboti asukoha tuvastamist pealtvaates kaamera kaudu (ptk 3.4). Neljandas peatükis on hinnatud lahenduse kasutamisega kaasnevaid võimalikke riske (ptk 4) ja lõpuks on antud ülevaade edasistest tegevustest ja arendusideedest (ptk 5).

1. OLEMASOLEVAD LAHENDUSED

Autonoomsete robotplatvormide juhtimiseks kasutatakse erinevaid lahendusi, mis oma keerukuse poolest varieeruvad suhteliselt palju, kuid põhimõtteliselt võib need lahendused jagada kahte gruppi: muutumatu ja muutuva teekonnaga lahendusteks. Levinum lahendus muutumatu teekonnaga robotplatvormide juhtimiseks põhineb juhtme- või joonejälgimisel. Levinumad muutuva teekonnaga lahendused kasutavad platvormi juhtimiseks laserit, güroskoopi, kaameraid. Järgnevalt antakse nendest lahendustest põhjalikum ülevaade ning tuuakse välja nende eelised ja puudused.

1.1. Joonejälgimine

Joonejälgimisel põhinevate robotplatvormide alla võib liigitada kolm põhimõttelt sarnast, kuid tehniliselt erinevat lahendust. Joonejälgimisel põhinevate lahenduste puhul kasutatakse platvormi teekonna planeerimiseks põrandale kleebitud värvilist linti, magnetlinti või põrandasse monteeritud voolujuhet ning platvorm saab ainult seda teekonda liikumiseks kasutada.

Värvilise kleeplindi või magnetlindi kasutamise põhiline eelis peitub lahenduse lihtsuses ja odavuses [1]. Uue teekonna andmiseks robotplatvormile ei pea tegema rohkemat, kui vana rada (kleeplint) põrandalt üles võtta ning kleepida maha uus teekond. Samas ei ole selline lahendus paindlik ning sõltuvalt, kui tihti on vaja robotile uus teekond planeerida, võib uue teekonna kleepimine ajaliselt kulukas tegevus olla. Lisaks ei ole antud lahenduse korral platvormil kuidagi võimalik teekonnale tekkinud takistusi vältida. Kuigi platvormidel võivad olla andurid, millega hoitakse ära kokkupõrge, ei saa robot enne edasi liikuda, kui takistus on tema teekonnalt kõrvaldatud. Samuti on tiheda liiklusega pindadel oht, et kleeplint kulub ära või läheb katki ning seeläbi jällegi seiskab platvormi.

Platvormide teekonda on võimalik planeerida ka põrandasse monteeritud voolujuhtme abil. Juhtmes on madalapingeline ning madala sagedusega vool, mida robotil paiknev vastuvõtja tuvastab. Kasutades erinevaid sagedusi, on võimalik planeerida erinevaid teekondi. Kuna juhe paigaldatakse üldjuhul põranda sisse, peab kõikvõimalikud platvormi teekonnad eelnevalt läbi mõtlema, sest hiljem ei ole neid enam võimalik muuta [1]. Nagu ka lindi kasutamisel, saab robot sõita ainult mööda üht kindlat teekonda ning takistuste sattumisel teekonnale, on roboti ainus võimalus peatuda, kuni objekt tema teekonnalt kõrvaldatakse.

1.2. Güroskoopnavigeerimine

Üks robotplatvormide juhtimise lahendusi on inertsiaalne navigeerimine (ingl *inertial navigation*). Taolisi süsteeme kasutatakse väga palju sõjaväetehnikas (laevad, lennukid, juhitavad raketid). Roboti hetkeasukoha ja kiiruse arvutab välja arvuti, kasutades selleks güroskoopi ja kiirendusmõõturit. Esialgne positsioon tuleb süsteemi jaoks siiski defineerida, kuid edaspidisel liikumisel tugineb platvorm ainult anduritele. Paraku on taolisel süsteemil üks puudus: iga järgmine roboti positsioon arvutatakse eelmisest integreerimise teel. Kuna andurid ei ole üldjuhul absoluutselt täpsed, tekib mõõteviga, mis ajapikku akumuleerub [2]. Selle probleemi lahendamiseks on hoonete põrandale paigaldatud saatjad, mille abil on võimalik kontrollida platvormi kõrvalekaldumist teekonnast ning seeläbi korrigeerida liikumist. Selliste saatjate kasutamine aga tähendab, et platvormi liikumiseks kasutatakse radasid, sarnaselt joonejälgimisel põhinevate lahendustega, ning seega ei ole süsteem iseseisvalt võimeline takistustega teekonnal toime tulema. Staatilise objekti korral on küll eelnevalt võimalik määrata robotile teekond, kust robot liigub ümber takistuste, kuid lõpuks peab robot ikkagi rajale naasma, et anduri kaudu oma positsiooni õigsust kontrollida.

1.3. Lasernavigeerimine

Lasernavigeerimiseks (ingl *Laser Target Navigation*) paigaldatakse seintele, postidele või masinatele peegeldavad lindid. Platvormile on aga monteeritud lasersaatja ning vastuvõtja, mis tavaliselt asub pöörleva torni otsas. Saatjast tulev kiir peegeldub lindilt vastuvõtjasse ning seeläbi mõõdetakse nurk ja kaugus ehk tehakse kindlaks roboti positsioon [3]. Mõõdetud andmeid võrreldakse ennustatud andmetega (positsioon) ning vea korral on võimalik roboti asukohta korrigeerida. Lasernavigeerimisel põhinevale platvormile on ka võimalik lihtsasti ette anda uus teekond. Võrreldes joonejälgimisel põhinevate või güroskoopi kasutatavate lahendustega, on lasernavigeerimist kasutades platvormi liikumisvabadus suurem. Paraku ei adresseeri antud lahendus takistuste tuvastamise ja vältimise küsimust. See on siiski lahendatav täiendavate andurite paigaldamisega robotile.

1.4. Loomulike objektide tuvastamine

Loomulike objektide tuvastamisel (ingl *Natural Feature Navigation*) põhinev lahendus kasutab üht või enamat kaugusmõõdikut (laser), güroskoopi ning Monte-Carlo lokaliseerimise meetodit oma asukoha määramiseks ning lühima teekonna planeerimiseks sihtpunktini. Antud lahenduse eeliseks on kindlasti tema paindlikkus, st süsteemi on võimalik suhteliselt kiiresti üles seada ning takistuste tekkimisel teekonnale on võimalik planeerida uus teekond ümber

takistuse. Võrreldes eespool mainitud meetoditega, ei vaja süsteem toimimiseks seintele või põrandale paigaldatud lisaelemente (kleeplindid, juhtmed). Töötamiseks peab robot siiski olema teadlik oma keskkonnast. See saavutatakse mällu salvestatud kaardi kasutamisega. Tuginedes kaardile ja liikumisel andurite kaudu saadud andmete hindamisel, üritab platvorm oma asukohta kaardil lokaliseerida [4].

1.5. Geonavigeerimine

Geonavigeerimisel (ingl *Geoguidance*) põhinev platvorm kasutab samuti mällu salvestatud kaarti oma asukoha tuvastamiseks keskkonnas. Liikumiseks kasutab robot eelnevalt defineeritud teekondi. Laserskänneri abiga on platvorm võimeline oma asukohta ruumis välja arvutama. Lisaks kasutab lahendus oma asukoha kontrollimiseks ja vajadusel korrigeerimiseks kindlaid punkte ruumis, näiteks seinu, poste, riiuleid [5]. Takistuste vältimiseks peab platvorm siiski tuginema oma anduritele, st platvorm ei suuda teekonnal paiknevaid takistusi kindlaks teha ning nendele reageerida, enne kui takistus on andurite ulatuses. Lisaks, hoonesisese plaani muutmine tähendab uue kaardi konstrueerimist ning uute teekondade defineerimist.

1.6. Masinnägemine

Masinnägemisel (ingl *Vision-Guidance*) põhinevad lahendused kasutavad tavaliselt platvormi juhtimiseks kaameraid (stereokaamerad) [6]. Oma asukoha määratlemiseks ruumis kasutatakse kaarti, mis luuakse roboti kaamerate abil, sõites kogu ala kõigepealt läbi. Sellist kaardistamise meetodit nimetatakse SLAM-iks (*Simultaneous localization and mapping*). Ebatäpsuste vältimiseks loodud kaardis on SLAM puhul ülimalt oluline roboti andurite täpsus ning müra puudumine keskkonnas. Kaartide perioodilisel täiendamisel võib mõõteviga kaarte moonutada ning seeläbi nad kasutamiskõlbmatuks muuta. Ebatäpsuseid on siiski täiendavaid meetmeid kasutades võimalik kõrvaldada (Kalmani filter, Monte-Carlo meetod). Sõltuvalt kaardistamiseks kasutavate vahendite täpsusest ei pruugi kaart olla täiuslik ning kaardil tekivad kohad, mille läbitavuse kohta puudub info. Selle probleemi lahendamiseks arvutatakse kaardi iga punkti jaoks tema hõivatuse tõenäosus (ingl *Evidence Grid technology*) ning sellest lähtuvalt saab platvormile planeerida teekonna. Kuna masinnägemisel põhinev lahendus kasutab navigeerimiseks kaameraid, puudub platvormil ülevaade tervest alast ning oma liikumisel ja takistuste vältimiseks peab robot tuginema ainult oma anduritele, millel on piiratud ulatus. Oma positsiooni arvutamiseks kaardil toetub robot samamoodi ainult oma anduritele ning väline tagasiside tema asukoha kohta puudub.

Kokkuvõtteks, kuigi joonejälgimisel põhinevad tööstuslike robotplatvormide navigeerimissüsteemid on odavamad ning tehniliselt lihtsad, ei ole rajad muudetavad või kaasneb radade ümbertõstmisega ajakulu ning raja ümbertõstmise ajal platvorm suure tõenäosusega seisab. Lisaks ei ole antud lahenduste puhul võimalik takistustest mööda sõita, mis samuti platvormi seiskab. Keerukamad lahendused ei vaja lisaseadmete (lindid ja andurid põrandal või põrandas, peegeldavad elemendid seinal) paigaldamist ning oma asukoha kontrollimisel ja teekonna planeerimisel toetutakse mällu salvestatud kaartidele ning robotil paiknevatele anduritele. Kuna kaardid on staatilised, siis hoonesisese planeeringu muutumisel võib tekkida vajadus uue kaardi ehitamisele ning ka platvormile uute teekondade defineerimisele.

2. PEALTVAATES KAAMERA KASUTAMINE TÖÖSTUSLIKU ROBOTPLATVORMI NAVIGEERIMISSÜSTEEMI ALUSENA

2.1. Pealtvaates kaamera kasutamise analüüs

Ühe võimaliku alternatiivlahendusena tööstuslike robotplatvormide juhtimiseks on antud töös välja pakutud pealtvaates, näiteks laohoone lae külge kinnitatud kaamera kaudu pildi edastamine. Sellise süsteemi kasutamine võiks ära lahendada mitmed olemasolevate süsteemide puudujäägid. Eelistena saab välja tuua:

- kaameralt pidev tagasiside roboti tegeliku asukoha kohta,
- terviklik ning reaalajas ülevaade alast, mis lubab takistusi planeeritud teekonnal juba enne tuvastada, kui robot reaalselt takistuseni on jõudnud, ning vastavalt vajadusele uus teekond arvutada,
- kaob vajadus andurite (laser kaugusmõõdikud, güroskoobid jne) kasutamiseks robotil.

Töös on objektide (takistuste, laudade, pinkide, sihtobjektide jms) ja vaba põrandapinda, tuvastamiseks välja pakutud tehisnärvivõrkude kasutamist. Sarnaselt olemasolevate lahendustega on roboti juhtimiseks vajalik kaart (pilt alast), mille saab hõlpsasti kaamera abil salvestada. Antud kaarti koos objektidega, mida soovitakse antud alal tuvastada, kasutatakse tehisnärvivõrkude treenimiseks. Kui neid olulisi objekte hoones hiljem ringi tõstetakse või kui muudetakse siseplaani, on närvivõrkude kasutamisel võimalik objekte ka uues positsioonis tuvastada ning selleks ei pea uut kaarti ehitama, mis annab olemasolevate lahenduste ees veel ühe eelise.

Pealtvaates kaamera kasutamise puudustena võib välja tuua:

- kaamera vaatevälja ei tohi miski segada, kõik objektid ja robot peavad koguaeg nähtaval olema,
- mida kõrgemal kaamera asetseb, seda suurem ala mahub pildile, kuid objektide detailsuse säilitamiseks tuleks kasutada suurema eraldusvõimega ja seega kallimaid kaameraid,
- suurte hoonete puhul peab terve ala katmiseks kasutama mitut kaamerat.

Kuna antud töö eesmärk on välja selgitada, kas välja pakutud robotplatvormi navigeerimissüsteemi on üldse võimalik luua, on katsete tegemisel kasutatud üht ainsat kaamerat ning katsed ise on tehtud suhteliselt lihtsates tingimustes.

2.2. Süsteemi ülesehitus

Navigeerimissüsteem koosneb kolmest suuremast komponendist. Pildi saamiseks on lakke monteeritud kaamera. Kaamera edastab pildi arvutile, kus toimub andmete (pildi) töötlus. Töödeldud informatsiooni põhjal ning samal ajal ka platvormilt saadud andmete põhjal, edastab arvuti käsud, näiteks uue teekonna määramine takistuse vältimiseks, platvormile. Suhtlus kõigi kolme komponendi vahel peab suure tõenäosusega toimuma juhtmevabalt. Kuna platvorm on pidevas liikumises, siis igasuguste juhtmete kasutamine kommunikatsiooniks arvuti ja platvormi vahel oluliselt piiraks platvormi liikumisvabadust ning tooks endaga kaasa ka muid ebamugavusi: juhtmed segaks teiste masinate liikumist põrandal ning tekib oht, et juhe takerdub kuhugi ning puruneb, seisates platvormi. Taoliste piirangute ja ebamugavuste vältimiseks oleks suhtluseks arvuti ning platvormi vahel ainus lahendus seega juhtmevaba tehnoloogia kasutamine.

Juhtmevaba kommunikatsiooni võiks kasutada ka arvuti ja kaamera vahel, kuid sõltuvalt oludest saab andmete edastamiseks kasutada ka juhtmeid. Juhtmete kasutamisel oleks üks võimalikust lahendusest olemasoleva taristu kasutamine. Kaameratena saab kasutada etherneti teel kommunikeerivaid kaameraid, mis võivad kasutada suure tõenäosusega juba olemasolevat IT taristut (kaabeldus, ruuterid). Etherneti teel suhtlevad kaamerad saavad sageli oma toite samast kaablist (ingl *power over ethernet*), eeldades et ruuter toetab seda. Määravaks on siiski, kas selline IT taristu on olemas, kas seda on võimalik antud süsteemi jaoks rakendada ning kui palju lisatööd sellega kaasneb (lisakaabeldus). Juhtmevaba tehnoloogia kasutamine on küll mitmes mõttes mugavam - ei pea lisakaabeldust vedama, hilisem seadmete teisaldamine lihtsam, kuid siinkohal peab kindlasti arvestama juhtmevaba signaali kvaliteediga (signaali tugevus, ulatus) ning seda mõjutavate teguritega (müra, häired). Kuna antud töö eesmärk pole juhtmevaba kommunikatsiooni uurimine, siis nendele küsimustele rohkem ei keskenduta.

Teoorias võiks suhtlus toimida otse kaamera ja platvormi vahel arvutit kasutamata, kuid siinkohal peab arvestama tehisnärvivõrkude treenimiseks vajaliku ressursiga. Hetkel robotplatvormides kasutatav riistvara ei ole selliste ülesannete teostamiseks mõeldud ning seega oleks mainitud ülesande lahendamiseks realistlikum parema jõudlusega eraldi arvuti kasutamine. Töös välja pakutud süsteemi ülesehitus on toodud selel 2.1.

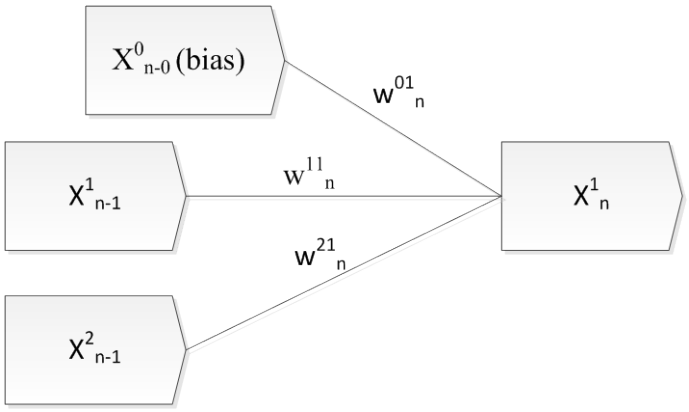
Sele 2.1 Navigeerimissüsteemi ülesehitus.

2.3. Tehisnärvivõrkude kasutamine objektide klassifitseerimiseks

Roboti juhtimiseks ja jälgimiseks, st teekonna planeerimiseks on oluline teada, kus paiknevad takistused, mida robot alguspunktist sihtpunkti liikumisel vältima peab. Takistuste ja vaba põrandapinna tuvastamiseks kasutatakse antud töös tehisnärvivõrke. Peatükid 2.3.1 – 2.3.3. põhinevad tööl [7].

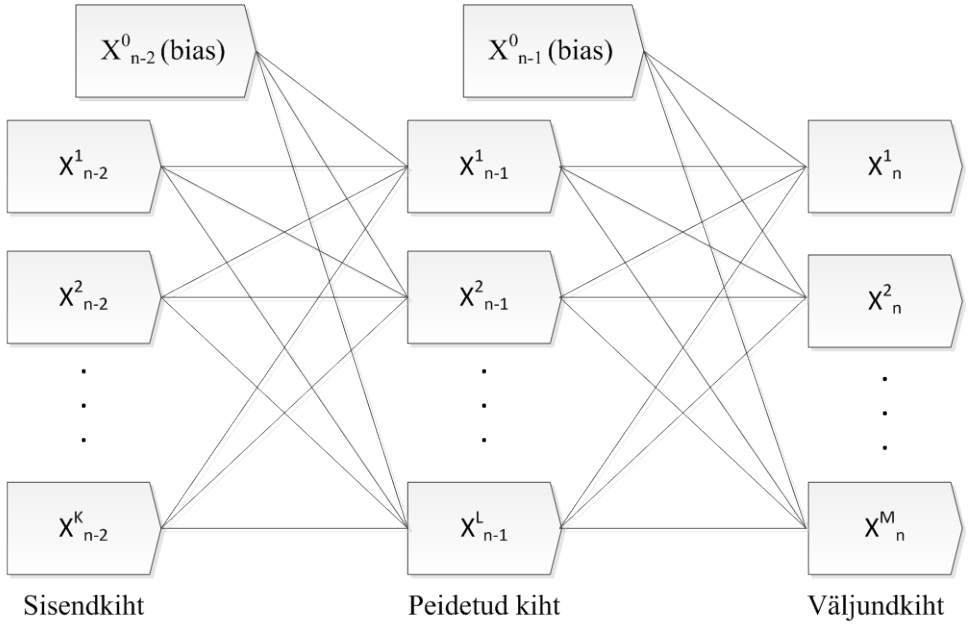
2.3.1. Tehisnärvivõrgud

Tehisnärvivõrk on lihtsustatud bioloogilise närvivõrgu mudel. Tehisnärvivõrk koosneb neuronitest, mis on omavahel kaalukoefitsientide kaudu ühendatud, st iga neuroni sisend korrutatakse läbi kaalukoefitsiendiga. Igal neuronil on väljund, mille väärtus arvutatakse tavaliselt sisendite kaalutud summa funktsioonist (sele 2.2) [8].



Sele 2.2 Lihtne tehisnärvivõrk.

Närvivõrkude struktuurid on erinevad, kuid reeglina paiknevad neuronid kihiti (sele 2.3). Esimest kihti nimetatakse sisendkihiks ning viimast kihti väljundkihiks, vahepeale jäävad peidetud kihid. Sisendkihis informatsioonitöötlust ei toimu, vaid jagatakse sisendneuronite väärtused esimesse peidetud kihti.



Sele 2.3 Tehisnärvivõrk ühe peidetud kihiga.

Tehisnärvivõrgud jagunevad kaheks: otsesuunatud ja rekurentsed (tagasisidega) tehisnärvivõrgud. Otsesuunatud võrgu neuroni väljund võib olla seotud ainult järgmises kihis oleva neuroni sisendiga. Tagasisidega võrkudes võib neuroni väljund olla ühendatud nii järgmise kui ka eelmise kihi neuronite sisendiga.

Tehisnärvivõrgu väljundkihi neuronite väärtuste arvutamiseks tuleb määrata sisendneuronite väärtused ning seejärel tuleb kihthaaval arvutada iga neuroni väljundväärtus. Väljundväärtus sõltub seejuures ainult neuroni sisenditest ja kaalukoefitsientidest.

Tehisnärvivõrkude kasutamise mõte seisneb selles, et võrku on võimalik mingi kindla ülesande täitmiseks treenida. Tehisnärvivõrgu treenimiseks kasutatakse teadaolevate sisendite ja väljunditega andmetekogumit. Treenimise käigus võrreldakse võrgu väljundeid ja teadaolevaid väljundeid ning modifitseeritakse kaalukoefitsiente nii, et teadaolevate väljundväärtuste ja tehisnärvivõrgu väljundväärtuste erinevus üle kogu andmehulga oleks minimaalne.

Näiteks võib võtta lihtsa kahekihilise otsesuunatud närvivõrgu, kus sisendkihis on kaks neuronit ning väljundkihis üks neuron (sele 2.2). Väljundkihi neuron on kaalukoefitsientide kaudu ühendatud mõlema sisendkihi neuroniga ning ühe täiendava neuroniga, mida kasutatakse nihutamiseks (ingl *bias neuron*).

Väljundneuroni väärtus arvutatakse valemiga

$$X_n^i = F\left(Y_n^i\right) = F\left(\sum_j X_{n-1}^j w_n^{ji}\right), \tag{2.1}$$

kus

X_n^i – väljundkihi neuroni väljundväärtus,

Y_n^i – kaalutud summa enne aktiveerimisfunktsiooni,

F – aktiveerimisfunktsioon (*tanh*).

Põhimõtteliselt käivitab närvivõrk mittelineaarse funktsiooni, mis arvutab kaalutud sisendsignaalide summast neuroni väljundi. Seda funktsiooni nimetatakse aktiveerimisfunktsiooniks.

Antud töös kasutatakse *tanh* aktiveerimisfunktsiooni, kuna see on sigmoid funktsioon, mis on koordinaatsüsteemi teljestike suhtes sümmeetriline. *Tanh* funktsiooni ja ka tema tuletist on lihtne arvutada:

$$y = \frac{d\tanh(x)}{dx} = 1 - \tanh^2(x) \tag{2.2}$$

Tavaliselt on tehisnärvivõrgud siiski keerukamad, kui antud näites kasutatud võrk, sisaldades mitut sisendit, mitut väljundit ning peidetud kihti. Selel 2.3 on ühtlasi ka toodud kõige levinumalt kasutatud närvivõrgu mudel (pertseptron), millel on K sisendit, L neuronit peidetud kihis ja M neuronit väljundkihis. Antud näites on tehisnärvivõrk täielikult ühendatud, st iga kihi iga neuroni väljund on ühendatud järgmise kihi iga neuroni ühe sisendiga. Sellise ehitusega närvivõrk on võimeline aproksimeerima suvalist pidevat funktsiooni, kui tema peidetud kihis on sobiv arv neuroneid [7].

Tehisnärvivõrgu väljundkihi neuronite väärtused arvutatakse sisendkihi neuronite väärtuste ja kaalukoefitsientide kaudu (valem 2.1). Levinud viis kaalukoefitsientide leidmiseks on nende õpetamine vea pöördlevi algoritmiga (ingl *backpropagation*). Antud algoritm seisneb selles, et etteantud sisendväärtustele lisatakse esialgsed kaalud. Summeeritud sisendväärtuste ja kaalude korrutisele rakendatakse aktiveerimisfunktsioon. Teostades seda igal kihil, saame lõpuks väljundvektori, mida võrreldakse teadaolevate etalonväärtustega ning saadakse vahe. Saadud vahet (viga) kasutatakse kaalude muutmiseks, liikudes närvivõrgus väljundkihist sisendkihi poole (sellest tuleneb ka algoritmi nimetus).

Nagu eelnevalt juba mainitud, on närvivõrgu treenimiseks vajalik andmekogumit, mille

väljundid on teada. Treenimist alustatakse kõikidele kaaludele suvalise väärtuse andmisega, valimi võtmisega kogumist ning seejärel valimi sisendväärtuste kaudu arvutatakse võrgu väljundid (valem 2.1). Teades arvutatud väljundi väärtust, võrreldakse seda teadaoleva väljundiga ning leitakse iga väljundneuroni jaoks viga:

$$\frac{\partial E_n^i}{\partial X_n^i} = X_n^i - T_n^i\,, \tag{2.3}$$

kus

X_n^i – arvutatud närvivõrgu väljund,

T_n^i – oodatud närvivõrgu väljund,

$\frac{\partial E_n^i}{\partial X_n^i}$ – Väljundneuroni viga.

Väljundi viga enne aktiveerimisfunktsiooni on:

$$\frac{\partial E_n^i}{\partial Y_n^i} = F'(X_n^i)\frac{\partial E_n^i}{\partial X_n^i} \tag{2.4}$$

Eelmise kihi neuroni viga on proportsionaalne neuroni kaaluga, millega ta ühendatud on:

$$\partial E_{n-1}^j = \frac{\partial E_n^i}{\partial Y_{n-1}^j} = \sum_i w_n^{ji}\,\frac{\partial E_n^i}{\partial Y_n^i} \tag{2.5}$$

Kaalu vea osakaal on võrdeline neuroni sisendväärtusega:

$$\frac{\partial E_n^i}{\partial w_n^{ji}} = X_{n-1}^j \frac{\partial E_n^i}{\partial Y_n^i} \tag{2.6}$$

Treenimise ajal antakse närvivõrgule ette mitmeid valimeid, pöördlevi algoritmi kasutades leitakse väljundite viga ning vastavalt sellele muudetakse kaalukoefitsienti (valem 2.6). Stabiilse paranemise tagamiseks närvivõrgu treenimisel tuleks kaalukoefitsiente korrigeerida teatud õppimiskiirust kasutades (ingl *learning rate*):

$$w(t) = w(t-1) - \eta\frac{\partial E}{\partial w'}\,, \tag{2.7}$$

kus

$w(t)$ – kaalukoefitsiendi väärtus pärast korrigeerimist

$w(t-1)$ – kaalukoefitsiendi väärtus enne korrigeerimist

η – õppimiskiirus

$\frac{\partial E}{\partial w'}$ - korrigeerimistegur (valemist 2.6)

Parima tulemuse saamiseks tuleks närvivõrku treenida mitmes tsüklis (epohh). Õpetamisperioode tuleks korrata niikaua, kuni saavutatakse väljundi vajalik täpsus või närvivõrgu väljundi täpsus lakkab paranemast.

Kokkuvõtteks, tehisnärvivõrke saab kasutada keerukate funktsioonide aproksimeerimiseks, kasutades selleks suurt hulka lihtsaid, kuid treenitavaid funktsioone. Närvivõrk koosneb neuronitest, mis käivitavad funktsiooni treenitavaid parameetreid kasutades. Neuronid omakorda paiknevad võrgus kihiti ning iga neuroni väljund on ühendatud ainult järgmise kihi iga neuroni sisendiga.

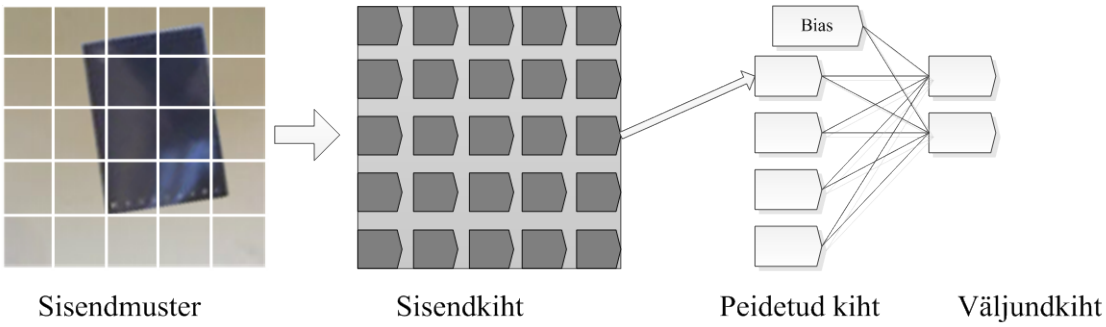
2.3.2. Konvolutsioonilised närvivõrgud

Kuigi piisavalt suuri tehisnärvivõrke on võimalik õpetada aproksimeerima suvalist funktsiooni, ei pruugi see olla parim lahendus: suured närvivõrgud ei ole tõhusad ning samamoodi võivad ülesande lahendamise juures olla teatud piirangud, mis muudavad suurte võrkude kasutamise ebapraktiliseks. Suured täielikult ühendatud närvivõrgud vajavad treenimiseks suurt hulka andmeid. See aga muudab õpetamisprotsessi ajaliselt kulukaks.

Konvolutsioonilistel närvivõrkudel on mitu spetsiifilist omadust, nagu lokaalsed vastuvõtuväljad (ingl *local receptive fields*), kaalude jagamine (ingl *weight sharing*), alamvalimid (ingl subsampling), mis on kasulikud just tunnuste (kujutiste) leidmiseks piltidelt. Järgnevalt on toodud näide konvolutsiooniliste närvivõrkude kasutamisest.

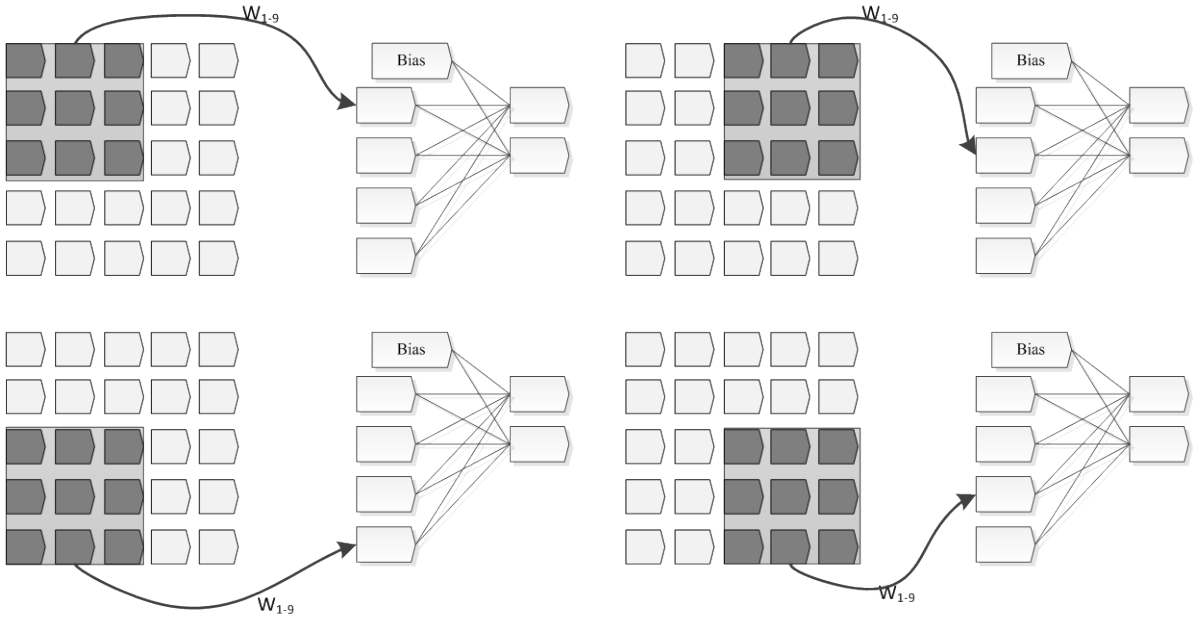
Halltoonis pildi elementide (mustrite) sisestamiseks närvivõrku kasutatakse sisendkihi, ühe peidetud kihi ning väljundkihiga närvivõrku. Sisendkihis peab olema neuroneid sama palju, kui on mustris piksleid. Paremaks näitlikustamiseks reastatakse neuronid kahemõõtmelisse massiivi ning seejärel kopeeritakse pikslite intensiivsuse väärtused sisendkihi neuronitesse. Värviliste piltide puhul on võimalik pilt jagada erinevatesse värvikanalitesse ja kasutada iga kanali jaoks eraldi tunnuskaarti.

Konvolutsioonilise närvivõrgu keskseks osaks on tuum (ingl *kernel*). Konvolutsioonilises närvivõrgus moodustab tuuma kaalude kogum, mis ühendab mitut eelmise kihi neuronit sellele järgneva kihi ühe neuroniga. Selel 2.4 on tuumaga ühendatud neuronid hallil taustal ning nool viitab järgmise kihi neuronile, millega nad on ühendatud.



Sele 2.4 Konvolutsiooniline närvivõrk.

Selel 2.5 on toodud näide, kus ainult osa sisendkihi neuronitest on ühendatud peidetud kihi iga neuroniga. Kõik need sisendkihi alamosad on ühendatud järgmise kihiga läbi sama konvolutsioonilise tuuma, st sisendkiht on ühendatud peidetud kihiga 9 kaalukoefitsiendi ning ühe neuroniga nihutamiseks (*bias neuron*). Konvolutsioonilise närvivõrgu nimetus on tulnud sellest, et sisuliselt teostab ta konvolutsioonitehte, mis seab kahele funktsioonile vastavusse kolmanda funktsiooni.



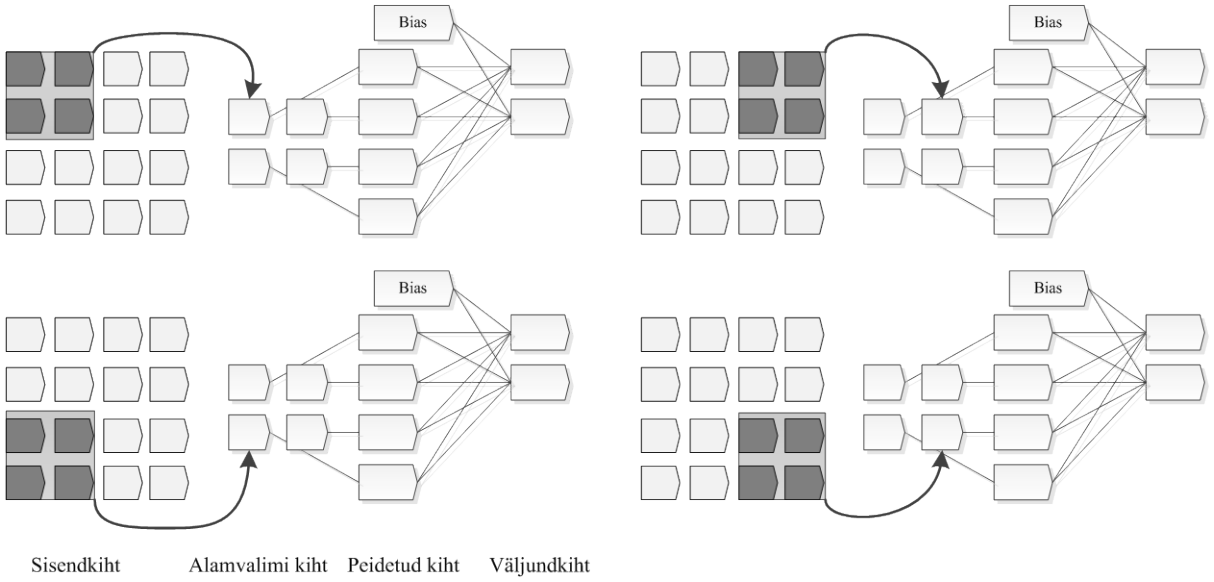
Sele 2.5 Konvolutsiooniline kiht.

Kaalude jagamisel on mitu kasulikku omadust. Kuna kaalukoefitsiente on taolises võrgus vähem, on ka võrgu muutujate väli väiksem ning seega saab võrgu treenimiseks väiksemat valimit kasutada, mis toob endaga kaasa ka aja kokkuhoiu närvivõrgu treenimisel.

Iga konvolutsiooniline tuum töötab sisendmustri alamosaga. Tuumad on võimelised eristama ja välja võtma sisendmustril lokaalseid tunnuseid. Närvivõrk on aga organiseeritud nii, et eristatud tunnuste kombinatsiooni põhjal saab võrk otsustada, kas sisendmustris on objekt olemas või mitte.

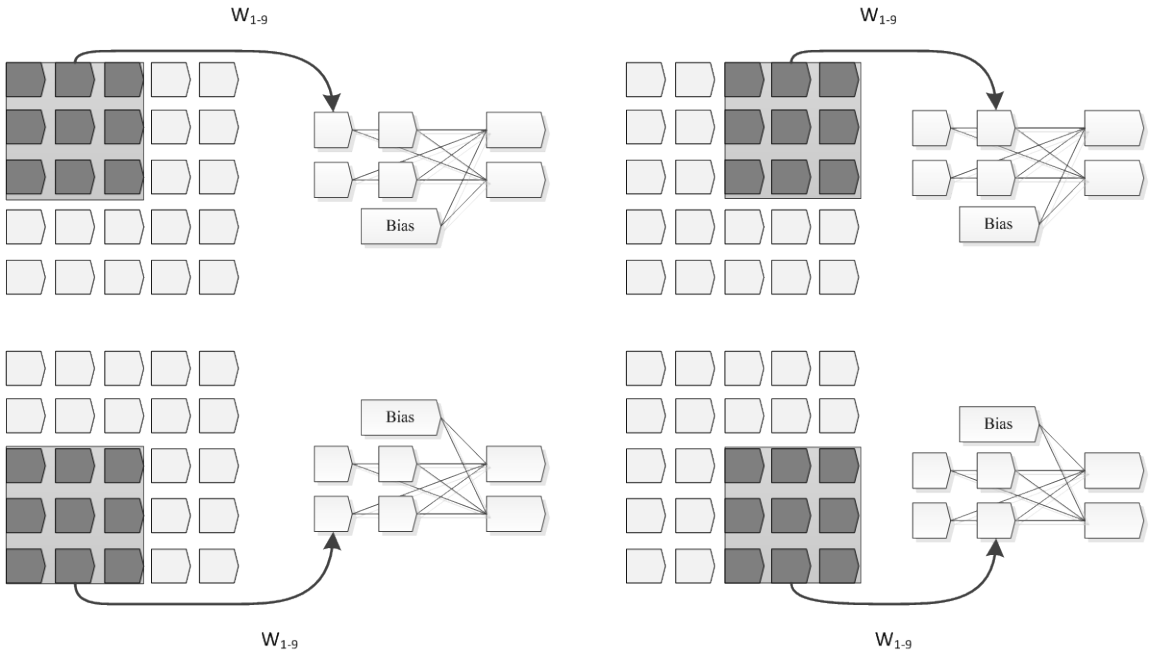
Kuna närvivõrk kasutab sisendkihi eri signaalide levitamiseks järgmisesse kihti sama konvolutsioonilist tuuma, on see võrk nihke suhtes invariantne. See tähendab, et sisendmustris olevaid tunnuseid on võimalik tuvastada, olenemata nende asukohast mustris. Nagu selgub, võib tunnuse täpse asukoha teadmine isegi vähendada klassifikaatori võimekust mustri identifitseerimisel [7].

Et veelgi vähendada täpsust tunnuse asukoha tuvastamiseks, kasutatakse alamvalimi kihte (ingl *subsampling*) kihte, mille abil on võimalik vähendada tunnuskaardi eraldusvõimet ning väljundi tundlikkust moonutuste ja nihete suhtes. Iga neuron alamvalimi kihis on ühendatud eelmise kihi nelja neuroniga ning seeläbi koondab nende nelja neuroni väljundid, kasutades selleks treenitavat tuuma.



Sele 2.6 Alamvalimi kiht.

Alamvalimi kihti ja konvolutsioonilist kihti on võimalik omavahel ühendada. Eraldusvõime vähendamiseks jätab konvolutsiooniline tuum sisendmustris vahele iga järgneva rea ja veeru (sele 2.7).

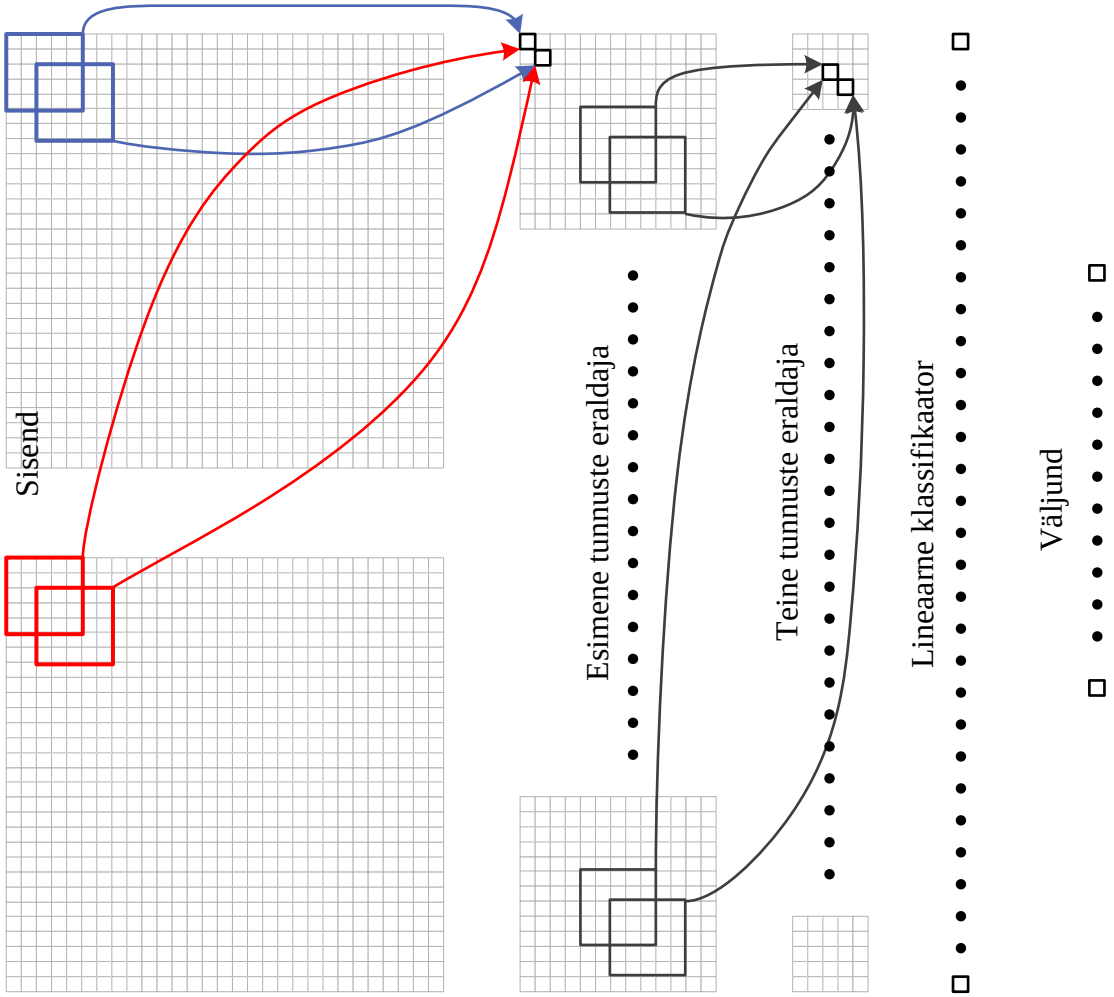


Sele 2.7 Integreeritud alamvalimi kihid.

Invariantsust geomeetriliste muutuste suhtes saab tõsta, kui võrgus kasutada mitut vahelduvat konvolutsioonilist ja alamvalimiga kihti. Alternatiivne variant on tunnuskaartide kuhjamine (ingl *stacking*) ning eraldusvõime vähenemist on võimalik kompenseerida tunnuskaartide suurendamisega igas kihis [7].

Mitme tunnuskaardi olemasolu on oluline, sest tunnuskaardid on eelmise kihiga ühendatud jagatud tuumade kaudu. Iga tuum on üldiselt võimeline eelmisest kihist tuvastama üht tunnust, st leitavate tunnuste arvu suurendamiseks on vaja suurendada ka tunnuskaartide arvu.

Antud töös kasutatud klassifikaatori sisendkihis on kolm või neli 29 × 29 piksli suurust mustrit RGB ja infrapuna värvikanalite jaoks (iga kanali jaoks üks muster). Vaikimisi on klassifikaator seadistatud nii, et eimene konvolutsiooniline kiht sisaldab kuut 13 × 13 piksli suurust tunnuskaarti, teises konvolutsiooniline kihis viiskümmend tunnuskaarti suurusega 5 × 5 pikslit ja kolmas peidetud kiht on lineaarne täielikult ühendatud kiht, mis koosneb 100 neuronist. Mõlemad konvolutsioonilised kihid kasutavad 5 × 5 piksli suuruseid tuumasid. Närvivõrkude võimekuse katsetamiseks tehti töös katseid erineva suurusega närvivõrkude peal.



Sele 2.8 Klassifikaatori struktuur.

Sisuliselt koosneb antud tehisnärvivõrk kahest omavahel ühendatud närvivõrgust, millel mõlemal on kolm kihti. Esimest kolme kihti võib vaadelda kui iseseisvat tunnuste eraldajat ja viimased kolm kihti moodustavad lineaarse klassifitseerija. Kolmas kiht on jagatu, st see on korraga nii tunnuste eraldaja väljundkiht kui klassifitseerija sisendkiht [7].

2.3.3 Õpetamine

Tehisnärvivõrke ja isegi konvolutsioonilisi võrke võib pidada üldiseks vahendiks, mille abil on võimalik lahendada pilditöötlusega seotud probleeme. Klassifikaatori konkreetsed omadused ja suutlikkuse määrab aga närvivõrgu struktuur ning selle treenimiseks kasutatavad andmed. Oluline on ka treenimiseks kasutatavate andmete kogumise meetod.

Antud töös kasutatav klassifikaator on võimeline andmete kogumiseks kasutama kaht erinevat meetodit: üks meetod tuleneb tähtede tuvastamisest käsitsi kirjutatud tekstis (meetod A), teine on aga innustatud mehitamata sõidukite navigatsioonisüsteemi poolt (meetod B).

Klassifikaatori õpetamiseks on võimalik kasutada mõlemat meetodit, kuid tulemuseks saab kaks väga erinevate omadustega klassifikaatorit [7].

Meetod A

Meetod A põhineb ideel, et konvolutsioonilised närvivõrgud on võimelised sisendmustris eristama erinevaid tunnuseid ja seejärel saab lineaarset klassifikaatorit õpetada leidma objekte sisendmustris. Põhimõtteliselt vastab klassifikaator oma töös küsimusel „kas sisendmustris on objekt olemas?“. Kõik klassifikaatori väljundid on sõltumatud ning korraga võib ühes sisendmustris olla mitu objekti klassi [7].

Treeningandmete koostamisel on aga oluline, et sisendmustri iga objekt on piisavalt suur või puudub täiesti. Klassifikaatoril peab olema piisavalt andmeid, mida on võimalik eristada, ning kui treeningandmetes on mustreid, kus objekt on kõigest mõne piksli suurune, tekitab see klassifikaatoris töös segadust ning halvendab tema võimekust.

Meetod B

Kasutades meetodit B, õpetatakse klassifikaatorit tuvastama sisendpildi ühe ainsa piksli kategooriat, täpsemalt aga mustri keskel oleva piksli kategooriat. Seejärel vastab klassifikaator küsimusele „mis kategooriasse antud piksel kuulub?“. Kategooriad on üksteist välistavad ja seega on klassifikaatori väljundid üksteisest sõltuvad [7].

Treeningandmete koostamisel heideti kõrvale mustrid, kust valitud piksel oli üle ülemääratletud, st piksel kuulus mitmesse kategooriasse korraga. Vaatamata sellele, veendutakse, et treeningandmetes on kõikides kategooriates võrdne arv mustreid.

Tundmatute mustrite hindamisel võib klassifikaator tuvastada mitu erinevat tunnust korraga. Tõenäosus sellise olukorra tekkimiseks on suurem just tunnuste piiride läheduses. Mustril oleva tunnuse klassi leidmiseks võib kasutada Bayes'i teoreemi ning otsustada millise klassi tunnus on suurem tõenäosusega ning ülejäänud tunnused alla suruda. Mõistlik on siiski edastada kõik väljundi väärtused, mida hiljem kasutatakse teekonna maksumuse arvutamiseks (ingl *costmap*). Seega kui piksel näiteks 50% tõenäosusega kuulub nii põrandale kui ka põrandal olev kastile, saab selle piksli maksumus (teekonna arvutamiseks vajalik) olema kuskil nende kahe klassi maksumuse vahel.

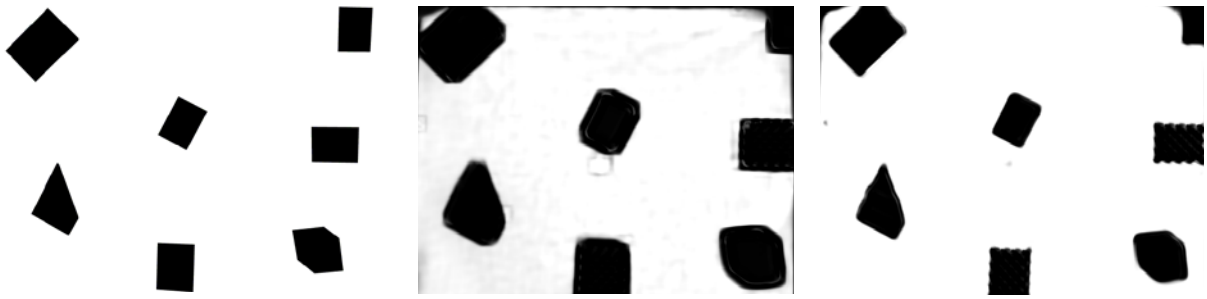
Meetod A vs Meetod B

Meetodi A üks eelistest on see, et antud meetodi kasutamisel ei ole nii oluline tunnuse

asukoht sisendmustris, st oluline on ainult tunnuse olemasolu. See on kasulik, kui tunnused sisendmustris ei ole väga täpselt defineeritud, näiteks ei pea tunnused olema defineeritud piksli täpsusega.

Meetodi A teine eelis peitub tema arvutusjõudluses: meetod A annab informatsiooni kogu 29 × 29 piksli suuruse sisendmustri kohta, samal ajal kui meetod B annab andmed ainult ühe piksli kohta. See tähendab, et meetod A puhul oleks võimalik kasutada esialgseid tulemusi, mis klassifitseerimisel saadakse, ning vajadusel neid tulemusi hiljem täiustada, kasutades selleks kattuvaid mustreid. Meetod B puhul peab aga pildi piksli haaval klassifitseerima. Pikslite vahelejätmisega küll võidetakse klassifitseerimise kiiruses, kuid sellega kaasneb ka oht, et olulised tunnused võivad jääda märkamata.

Meetodi B põhiliseks eeliseks on tema väljundi eraldusvõime (sele 2.9), st klassifitseeritakse iga pildi piksel. Meetodi A puhul piirab eraldusvõimet mustri suurus. Eraldusvõimet on küll võimalik tõsta, kasutades kattuvaid mustreid, kuid väljund jääb siiski hajutatuks, sest meetod A puhul ei defineerita tunnuse asukohta sisendmustris [7].



Sele 2.9 Klassifitseerimismeetodite võrdlus: käsitsi klassifitseeritud mask (vasakul), meetod A (keskel), meetod B (paremal).

Õpetamisprotsess

Nagu eelnevalt juba mainiti, kasutatakse närvivõrgu treenimiseks andmekogumit, mis sisaldab mustreid, mille väljundväärtused on teada. Enne treenimise alustamist antakse kaalukoefitsentidele suvalised väärtused ja määratakse õppimiskiirus. Õppimise käigus laetakse andmetest ükshaaval mustrid, eraldatakse värvikanalid ning sisendkihi neuronitele antakse pikslite intensiivsuse väärtused. Pärast sisendkihi käivitamist on võimalik arvutada esimese peidetud kihi väljundid (valem 1). Esimese peidetud kihi väljundeid kasutatakse teise peidetud kihi sisendites, mis samamoodi arvutatakse. Protsess kordub, kuni on välja arvutatud väljundkihi neuronite väljundid.

Pärast väljundite väärtuste saamist arvutatakse närvivõrgu viga, mis on teadaoleva

väljundväärtuse ja arvutatud väärtuse vahe. Järgnevalt kasutatakse vea pöördlevi meetodit, kus liigutakse väljundkihilt tagasi sisendkihi poole ning leitakse viga esimeses peidetud kihis. Viga jaotatakse kaalukoefitsientide vahel (valem 6). Jagatud kaalukoefitsientide puhul liidetakse vead kokku. Viimase sammuna kohandatakse kaalukoefitsiente (valem 7).

Närvivõrku treenitakse mitmes epohhis. Pärast igat epohhi vähendatakse õppimiskiirust, tasakaalustades niimoodi õppimisprotsessi. Vältimaks sõltuvust järjekorrast, kuidas mustreid närvivõrgule ette antakse, muudetakse treenimiseks kasutatav andmekogum juhuslikuks.

Õppimiskiiruse tõstmiseks kasutatakse iga kaalu puhul veel täiendavat parameetrit – hessiani diagonaal. Antud parameeter arvutatakse enne igat epohhi 500 mustri põhjal. Antud meetod tõstab närvivõrgu õppimiskiirust kolmekordselt ning seejuures ei nõua suuri arvutuslike kulutusi [7].

Nii meetodi A kui ka meetodi B jaoks on protsess sarnane, erinevus seisneb ainult selles, kuidas andmekogumit genereeritakse. Tulemuseks on siiski erinevate omadustega klassifikaatorid. Meetod A tuleb paremini toime sisendi vigadega, meetod B aga tagab parema väljundi eraldusvõime.

Närvivõrgu ehitamisel on ka oluline õige neuronite (ja kihtide) arvu valik. Kui närvivõrk on antud ülesande lahendamiseks liiga suur (liiga palju neuroneid), võib langeda võrgu töökiirus, kasvada mälukasutatavus, tekkida üleõppimise efekt (väljund hakkab kajastama mitteolulisi detaile, nt müra), võrk võib muutuda ebastabiilseks ja hakata väga tugevalt reageerima sisendvektori väärtuste muutustele või kaotab võrk oma üldistusvõime, st kaotab oma aproksimeerimisvõime [9].

Pärast klassifikaatori treenimist on teda võimalik kasutada ka objektide tuvastamiseks pealtvaates tehtud piltidel ning teekonna planeerimiseks.

2.4. Robotic operating system navigeerimissüsteemi baasina

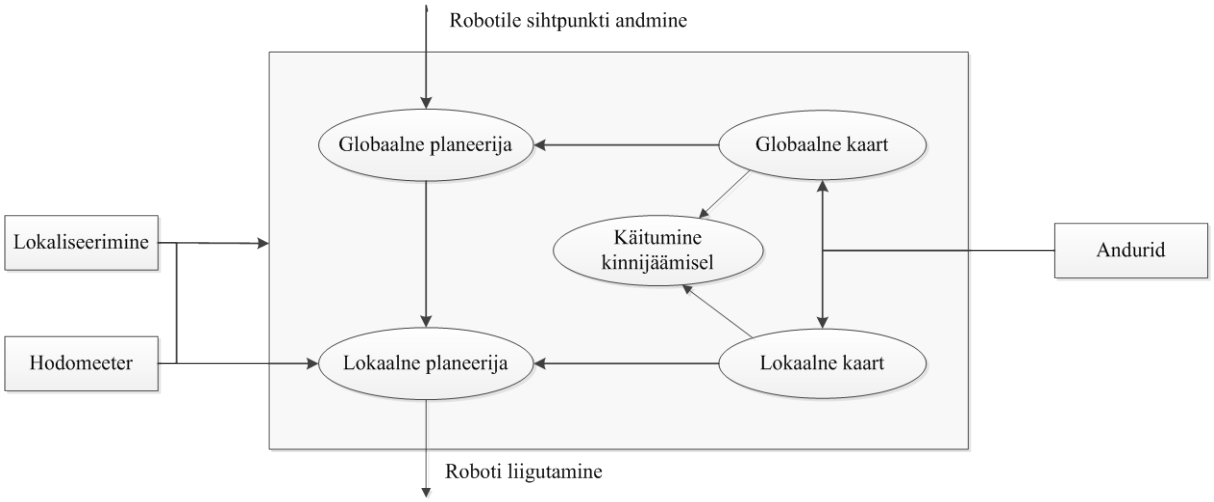
2.4.1. Robotic operating system

Klassifikaatori abil on võimalik tuvastada kaardil objekte ja arvutada ka teekonna maksumus roboti liikumisel alguspunktist sihtpunkti, kasutades selleks spetsiaalset kaarti (*costmap*), kus on iga punkti maksumus välja arvutatud. Kuna antud töös kasutatakse navigeerimissüsteemi simuleerimiseks ROS-i, siis uuritakse teekonna maksumuse arvutamist ja planeerimist just ROS-i baasil.

ROS on just mobiilsete robotite juhtimiseks mõeldud tarkvara raamistik, mis pakub erinevaid

teeke, tüüreleid, visualiseerimistööriistu, sõnumite ja pakettide haldamist ning tuge erinevatele robotitele. See oluliselt lihtsustab erinevate rakenduste loomist robotitele ja vähendab robotite seadistamiseks ja kasutuselevõtmiseks kuluvat aega.

ROS on peamiselt mõeldud aga masinnägemisel (ptk 1) põhinevate süsteemide jaoks. See tähendab, et robot kasutab keskkonnas liikumiseks eelnevalt kokku pandud kaarti ning oma andureid (LIDAR, stereokaamerad), mille abil teeb robot kindlaks läheduses olevad takistused ning oma asukoha kaardil. Selel 2.10 on toodud tüüpiline ROS-i baasil töötava navigeerimissüsteemi struktuur [10].



Sele 2.10 ROS-i navigeerimissüsteem.

Antud süsteemi tuumiku moodustavad teekonnaplaneerijad koos kaartidega teekonna maksumuse arvutamiseks. Nagu selelt 2.10 on näha, on nii kaarte kui ka planeerijaid kaht erinevat tüüpi: globaalsed ning lokaalsed. Globaalne planeerija ning kaart kasutavad eelnevalt ehitatud ja mällu salvestatud kaarti ning selle abil on võimalik planeerida roboti teekond terve kaardi ulatuses. Lokaalse teekonna planeerimiseks ja kaardi ehitamiseks kasutatakse aga anduritest tulevaid andmeid, st lokaalset planeerijat kasutatakse teekonna planeerimiseks ümber takistuse, kui andurid roboti teel selle avastavad.

Selel 2.10 toodud navigeerimissüsteemi toimimiseks peavad olema ära kirjeldatud kõik koordinaatsüsteemide vahelised seosed (kaart, hodomeeter, roboti alus, rattad, andurid robotil) ning süsteem peab sisaldama ka hodomeetrit. Koordinaatsüsteemide vahelisi seoseid kasutatakse roboti asukoha määramiseks kaardil, hodomeeter aga edastab meile andmeid roboti liikumise kohta (kiirus, läbitud vahemaa).

Täiendavalt on võimalik ka defineerida roboti käitumine olukorras, kus robot tajub, et ta on

kinni jäänud [9].

Näites toodud tüüpiline navigeerimissüsteem ei sobi täielikult antud töös välja pakutud ülesande lahendamiseks ning seega tuleb ROS-i antud lahenduse jaoks kohandada.

Välja pakutud lahenduses kasutatakse nii roboti asukoha kui ka takistuste tuvastamiseks pealtvaates kaamerat. Kui kaamera vaateväli hõlmab kogu ala ning jälgib seejuures roboti ümber toimuvat, kaob otsene vajadus robotil olevate andurite ja koos sellega ka lokaalse kaardi järele ära, st kaamera kaardistab pidevalt ja korraga kõike.

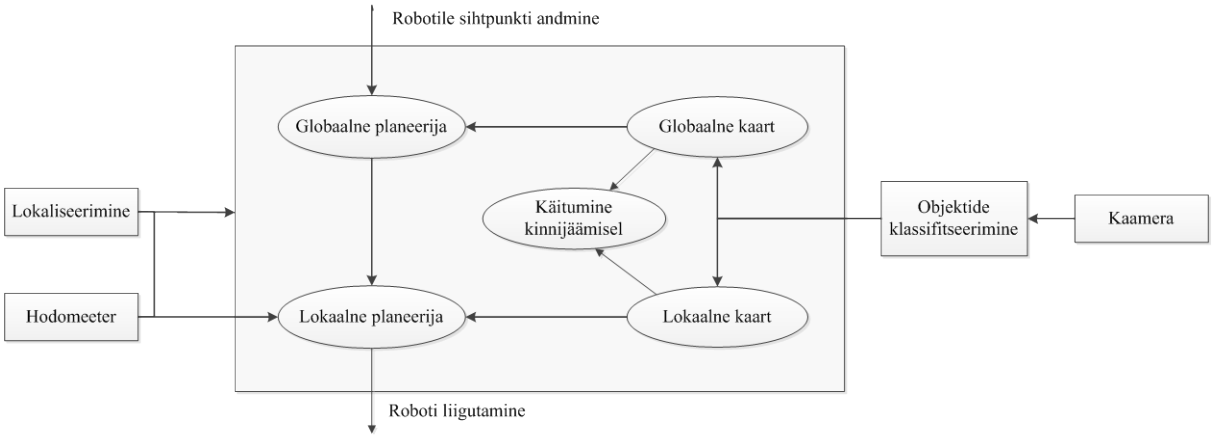
Samas peab arvestama, et tüüpilise masinnägemisel põhineva navigeerimissüsteemi globaalne kaart on mällu salvestatud ja muutumatu kaart. Sellest lähtuvalt sarnaneb pealtvaates kaamera rohkem andurile, mis lihtsalt ei paikne robotil, ja seega edastab andmed lokaalsele kaardile, mis pidevalt uueneb. Robotil paiknevad andurid on aga, erinevalt pealtvaates kaamerast, väiksema ulatusega ja seega moodustab pealtvaates kaamera pilt suure lokaalse kaardi.

Sõltuvalt esitatud vajadustest ja nõudmistest ei ole vaja pidevalt korraga tervet kaamera vaatevälja uuendada. Suure hoone pinna ja seega ka suure kaardi pidev uuendamine terves ulatuses võib nõuda võimsamat arvutit. Samas kui platvorm paikneb ainult ühes kaardi osas, ei pruugi olla tähtis teada, mis kaardi teises otsas toimub, st pidevalt jälgitavat ala oleks mõistlik piirata.

Navigeerimissüsteemi jaoks on siiski mõlemad, nii lokaalne kui ka globaalne kaart ja planeerija oluline. Pealtvaates kaamerat saab kasutada nii globaalse kaardi loomiseks, nt kaardistatakse ja salvestatakse andmed süsteemi käivitumisel, kui ka lokaalse kaardi tegemiseks. Kuna lokaalne kaart on mõeldud roboti lähiümbruse jälgimiseks, siis jälgitaksegi pidevalt kaamera abil roboti asukohta ning teatud ala selle raadiuses. Erinevalt traditsioonilisest lahendusest, mis kasutab robotil paiknevaid sensoreid ja on seega piiratud nende sensorite ulatusega, saab laes oleva kaamera kasutamisel jälgitava ala suurust vabalt muuta.

Sarnaselt teiste masinnägemisel põhinevate lahendustega on ka siin oluline, et kõik koordinaatsüsteemide vahelised seosed oleks ära kirjeldatud. Hodomeetri andmeid kasutatakse samamoodi andmete saamiseks roboti liikumise kohta. Neid andmeid kasutatakse ka roboti asukoha korrigeerimiseks, kui kaamerast tulevad andmed roboti asukoha kohta ja hodomeetri väärtused liialt erinevad. Olukordade jaoks, kus robot on kinni jäänud, kasutatakse eespool mainitud ROS'i sisseehitatud käitumismudelit. Olemasolevate lahendustega võrreldes jääb navigeerimissüsteemi struktuur põhimõttelt samaks (sele 2.11),

kuid erinevus seisneb viisis, kuidas andmeid ümbritseva keskkonna kohta kogutakse ja töödeldakse. Lisaks võib erinevusena märgata, et kaamerapilt lastakse läbi klassifikaatorist, mille käigus tuvastatakse objektid ja saadetakse vastavad andmed edasi.



Sele 2.11 Laes paikneval kaameral põhineva navigeerimissüsteemi struktuur.

2.4.2. Teekonna maksumuse arvutamine

Tehisnärvivõrgu abil tuvastatakse objektid ning koostatakse nende põhjal kaart. Antud kaart imporditakse ROS-i, kus seda kasutatakse omakorda maksumuskaardi (*costmap*) loomiseks ja teekonna planeerimiseks.

Teekonna planeerimiseks oleks mõistlik kasutada nii globaalset kui ka lokaalset kaarti. Globaalse kaardi abil on võimalik planeerida teekond terve ala ulatuses. Lokaalset kaarti aga kasutatakse roboti anduritest (antud juhul kaamera) saadud informatsiooni põhjal lähiümbruse jälgimiseks, ning erinevalt globaalsest kaardist, uuendatakse lokaalset kaarti teatud sagedusega. Maksumuse arvutamiseks kasutatakse mõlema kaardi puhul siiski sama põhimõtet, erinevus seisneb lihtsalt jälgitava ala suuruses ning uuendamise sageduses.

Teekonna planeerimiseks kasutatava kaardi loomiseks, ehitatakse kõigepealt võrestik, kus arvutatakse võrestiku iga lahtri jaoks tema läbimiseks kuluv maksumus (ingl *occupancy grid*), kusjuures lahter võib olla defineeritud piksli täpsusega. Sõltuvalt objektist määratakse lahtritele erinev maksumus, näiteks on takistusel suurem maksumus kui vabal pinnal. Iga lahtri jaoks välja arvutatud maksumust aga kasutatakse hiljem roboti teekonna välja arvutamisel.

Igal maksumuskaardi lahtril saab olla 255 erinevat väärtust ning vastavalt oma väärtustele jagunevad need lahtrid kolme erinevasse kategooriasse, st nad on kas vabad, hõivatud või tundmatud, mille puhul info lahtri hõivatuse kohta puudub. Lahtri väärtus arvutatakse pildi

pikslite intensiivsuse põhjal ning selleks kasutatakse valemit

$$occ = \frac{\left(255 - color_{avg}\right)}{255.0}, \tag{2.8}$$

kus

$color_{avg}$ – 8-bitine keskmine intensiivsus, mis üle kõikide värvikanalite on mõõdetud.

Nagu valemist järeldada võib, saab hõivatuse kasutamiseks kasutada ka värvipilti, mille puhul võrdub piksliintensiivsus värvikanalite keskmise intensiivsusega. Enamasti kasutatakse siiski must-valgeid pilte, kus heledamad pikslid tähistavad vaba ala ehk piirkonda, mida robot võib oma liikumiseks kasutada, ning tumedad pikslid hõivatud ala ehk takistusi, vahepeale aga jäävad tundmatud pikslid. Mis väärtustest alates loetakse piksleid aga vabaks või hõivatuks, sõltub täielikult seadistustest. Näiteks, kui piksli intensiivsus on 245 (praktiliselt valge), siis tema hõivatuse tõenäosus on 0,04. Kui aga piksli intensiivsus on 70, on tema hõivatuse tõenäosus 0,72. Kas sellise väärtuse puhul loetakse pikslit hõivatuks või tundmatuks, sõltub aga lävendist, millest alates loetakse pikslit hõivatuks, tundmatuks või vabaks.

Piksli intensiivsused 0 kuni 255 tõlgendab ROS omakorda veel ümber täisarvuks vahemikus 0 kuni 100, kus 0 tähistab vaba pikslit ja 100 absoluutselt hõivatud pikslit, täielikult tundmatu piksli väärtus on aga -1.

Objektide maksumuse arvutamisel ja teekonna planeerimisel peab arvestama ka roboti enda mõõtmetega. Mõõtmetega arvestamine on oluline, et vältida olukordi, kus näiteks planeeritakse robotile teekond läbi ava, kust robot tegelikkuses läbi ei mahu. Samuti ei tohi teekonda planeerida nii, et robot takistusest liiga lähedalt mööduks ja sellega kokku põrkaks. Mainitud probleemi lahendamiseks kasutatakse ROS-is inflatsiooni raadiust (ingl *inflation radius*).

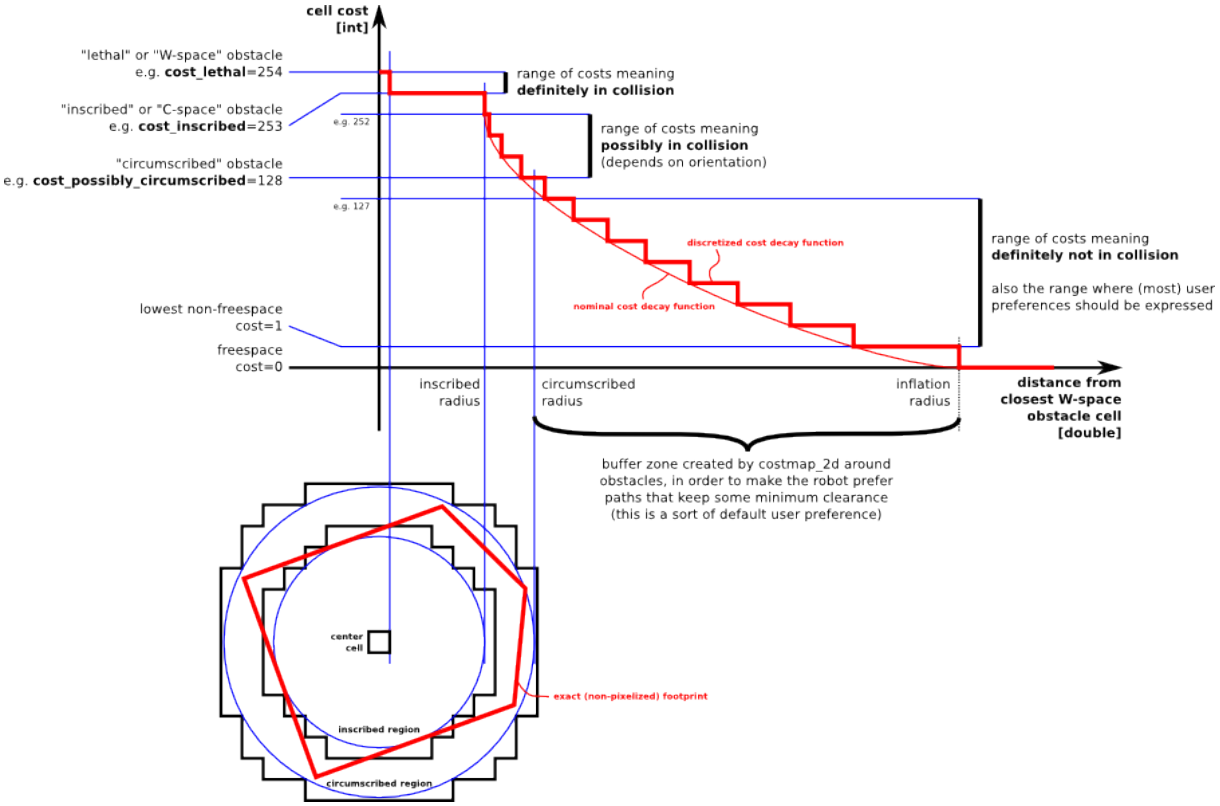
Inflatsiooni arvutamiseks leitakse kaardil kõigepealt need lahtrid, kus tuvastati takistused ning takistuse ümber olevatele lahtritele (pikslitele) arvutatakse uued väärtused (lahtri hind), mis takistusest eemaldudes inflatsiooni raadiuse ulatuses vähenevad. Väärtuste arvutamiseks kasutatakse viit erinevat taset [11].

- Takistusega lahtri maksumus on suurim (*lethal cost*). Kui roboti keskpunkt peaks asetsema antud lahtris, on robot takistusega kokku põrganud.
- Lahtrid, mis on takistusest roboti raadiuse kaugusel või lähemal (*inscribed cost*). Juhul kui roboti keskpunkt asetseb lahtris, mille väärtus on võrdne või üle antud vahemikus

kirjeldatud väärtuse, siis on robot suure tõenäosusega takistusega kontaktis.

- Lahtrite maksumuse arvutamisel peab arvestama ka roboti kujuga. Kui robot on näiteks ristkülikukujuline, siis sõltuvalt tema orientatsioonist võib robot olla takistusega kokku põrganud või mitte. Seetõttu oleks taolise kujuga robotite puhul mõistlik kasutada hoopis roboti välimist raadiust (*circumscribed cost*).
- Vaba ruumi (*freespace*) hind on null, st antud ruumis puuduvad igasugused takistused ning robot peaks võimalusel eelistama just neid lahtreid.
- Tundmatu väärtusega (*unknown cost*) lahtrite kohta informatsioon puudub ning neid lahtreid saab tõlgendada vastavalt vajadustele
- Kõikidele teistele lahtritele arvutatakse väärtused vaba ruumi väärtuse (null) ning välise raadiusega piiratud lahtrite väärtuse (*circumscribed cost*) vahel. Lahtrite väärtus sõltub nende kaugusest takistusega lahtrist ning ette antud hääbumisega (ingl *decay function*).

Antud töös kasutatakse navigeerimissüsteemi riistvaralise baasina Robotino XT mobiilset robotit. Roboti ristlõige on ringjoon raadiusega 0,185 m. See tähendab, et kõikidele lahtritele, mis asuvad takistustele mitte rohkem kui 0,185 m läheduses, arvutatakse kõrgem hind. Inflatsiooni raadiusena kasutatakse vaikeväärtust, mis on 0,55 m, st et hind arvutatakse ümber kõikidel lahtritel, mis on takistuse ümber oleva 0,55 m raadiusega ala sees. Inflatsiooni raadiust on võimalik muuta ning see tuleks valida vastavalt vajadustele. Inflatsiooni raadius tekitab sisuliselt puhvri takistuse ja roboti vahel. See tähendab, et robotile arvutatakse teekond, mis hoiab takistusest teatud vahemaad. Teatud vahemaa hoidmine on kindlasti oluline, kuid liiga suur puhver võib hakata piirama roboti liikumisvabadust. Liiga väikse raadiuse valimisel aga võib jällegi tekkida kokkupõrkeoht takistusega.



Sele 2.12 Inflatsiooni teel lahtrite hinna arvutamine [10].

Selel 2.13 on näiteks toodud kaart, mida teekonna planeerimiseks kasutada saab. Punased pikslid näitavad kaardil olevaid takistusi, sinise värviga on tähistatud alad, mis on takistusest roboti raadiuse kaugusel. Helehall ala tähistab vaba ruumi, mida robot saab oma teekonna planeerimisel kasutada. Tumehalli värviga on tähistatud tundmatu ala, mille koht meil informatsioon puudub. Kokkupõrke vältimiseks ei tohi roboti teekond kunagi lõikuda punase lahtriga ning roboti keskpunkt ei tohi sattuda sinisesse alasse.

Sele 2.13 **Roboti teekonna planeerimine maksumuskaardi abil.**

Kokkuvõtteks, roboti teekonna planeerimisel peab arvestama takistustega kaardil ning ka roboti enda mõõtmetega. Teades takistuste mõõtmeid ja asukohti kaardil ning roboti enda mõõtmeid, arvutatakse kaardi iga punkti jaoks välja teatud väärtus (lahtri hind). Takistuste väärtus on suurem (kõrgem hind), vaba ruumi maksumus aga väiksem. Antud lahtrite väärtusi kasutatakse edaspidi roboti optimaalseima teekonna planeerimiseks.

2.4.3. Teekonna planeerimine

Globaalne teekonna planeerimine

Teekonna planeerimiseks kasutab ROS Dijkstra algoritm [12]. Algoritm arvutab välja alguspunktist sihtpunktini kulgeva kõige väiksema maksumusega teekonna. Teekonna arvutamiseks on algoritmil vaja sõlmede (punktide) kogumit ja maksumust, mis kulub liikumiseks ühest sellisest sõlmest tema naabersõlme.

Kui eeldada, et robot asub alguspunktis ning ta peab liikuma sihtpunkti, siis kasutades Dijkstra algoritmi toimub roboti teekonna planeerimine järgnevalt:

1. Igale sõlmele määratakse esialgne maksumus. Alguspunkti algne väärtus on null, kõikide teiste punktide algne väärtus on lõpmata suur.
2. Kõik sõlmed märgitakse ära kui läbi vaatamata ning alguspunkt märgitakse aktiivseks

sõlmeks.

3. Vaadatakse läbi aktiivse sõlme kõik naabersõlmed ning arvutatakse igasse sellesse sõlme liikumise maksumus.
4. Pärast naabersõlmede läbivaatamist märgitakse aktiivne sõlm läbi vaadatud sõlmeks. Läbi vaadatud sõlme edaspidi enam ei kontrollita.
5. Kõige väiksema maksumusega naabersõlm märgitakse uueks aktiivseks sõlmeks ning omakorda vaadatakse läbi uue aktiivse sõlme naabersõlmed.
6. Antud protsessi korratakse, kuni sihtpunkt märgitakse läbi vaadatuks ning seejärel protsess lõpetatakse.

Kuna algoritm üritab leida kõige väiksema maksumusega teekonna sihtpunktini, siis algoritm peab kindlaks tegema iga sõlme naabersõlmede väärtused, kuni on lõpuks sihtpunkti jõutud. Põhimõtteliselt võib öelda, et algoritm laieneb pidevalt igas suunas alguspunktist väljapoole, kuni ta jõuab sihtpunktini, vaadates läbi iga sõlme, milleni jõudmise maksumus on väiksem kui sihtpunktini jõudmiseks. Lähtuvalt algoritmi tööpõhimõttest võib ta teatud topoloogia puhul olla väga aeglane. Kõige väiksema maksumusega teekonna leidmiseks peab Dijkstra algoritm läbi töötama suure koguse sõlmi, mis võib anda eelise algoritmidele, mis üritavad leida kiirelt teekonna sihtpunktini, samas ei tähenda see, et leitud teekond on kõige odavam (ei ole lühim).

Dijkstra algoritmi kasutatakse globaalse teekonna planeerimiseks, st teekonna planeerimiseks roboti alguspunktist sihtpunktini. Seejuures kasutatakse globaalse teekonna planeerimiseks tihtipeale täpsemaid, kuid staatilisi, st muutumatuid kaarte. Roboti liikumise ajal võivad tema teekonnale sattuda ootamatud takistused, nt inimene, ning robot peab oskama sellele reageerida ning suutma planeerida uue teekonna ümber takistuse. Teekonna planeerimiseks ümber takistuse kasutab ROS lokaalset planeerimist, mille eesmärgiks on uue teekonna planeerimine ümber trajektoorile tekkinud takistuse, jättes võimalusel ülejäänud eelnevalt planeeritud teekonna samaks.

Lokaalne teekonna planeerimine

Traditsiooniliste lahenduste puhul kasutatakse liikuvate takistuste tuvastamiseks ja vältimiseks robotile paigaldatud andureid. Nende ulatus on tihtipeale piiratud, st andurite abil on võimalik jälgida ainult teatud suuruses ala roboti ümber (lokaalne ala) ning teekonnale tekkinud takistusele on robot võimeline reageerima ainult juhul, kui ta on takistuse andurite

abiga tuvastanud. Takistuse tekkimisel teekonnale ei ole robot võimeline uut teekonda arvutama kaugemale, kui tema andurid „näevad“, st uut teekonda on robot võimeline planeerima ainult teatud ulatuses (lokaalselt).

Antud töös välja pakutud lahendus on selline, kus kogu teekond on kaamera vaateväljas ja seega oleks võimalik kõiki teekonnale tekkivaid takistusi pidevalt jälgida ja nende ümber uut teekonda arvutada. Samas ei pruugi selline lähenemine olla alati praktiline, sest see esitab riistvara suhtes oluliselt suuremaid nõudmisi. Efektiivsem oleks seega jälgida ainult roboti vahetus ümbruses olevat ala (lokaalset ala) ja kasutada lokaalset teekonna planeerimist. Erandiks võib olla olukord, kus teekonnale ilmunud takistus ei asu roboti lähiümbruses ning blokeerib tervenisti roboti teekonna ära. Antud juhul oleks kasulikum, kui roboti teekonda jälgitakse pidevalt kogu tema ulatuses ja seega oleks võimalik robotile uus teekond ennetavalt arvutada, mitte aga siis, kui takistus alles roboti vaatevälja jõuab.

Lokaalse planeerimise tööpõhimõte seisneb sobiva lokaalse teekonna perioodilises otsimises. Selleks arvutatakse mitu erinevat trajektoori ning kontrollitakse, kas trajektoor lõikub takistustega (sele 2.14). Kui arvutatud teekonnad ei lõiku takistusega, antakse neile hinnang ning valitakse selle põhjal parim teekond.

Lokaalse liikumise planeerimiseks saab ROS kasutada kaht lähenemisviisi: trajektoori levimise (ingl *Trajectory Rollout*) või dünaamilise akna lähenemise algoritmi (ingl *Dynamic Window Approach*). Mõlema meetodi tööpõhimõte on järgmine [12]:

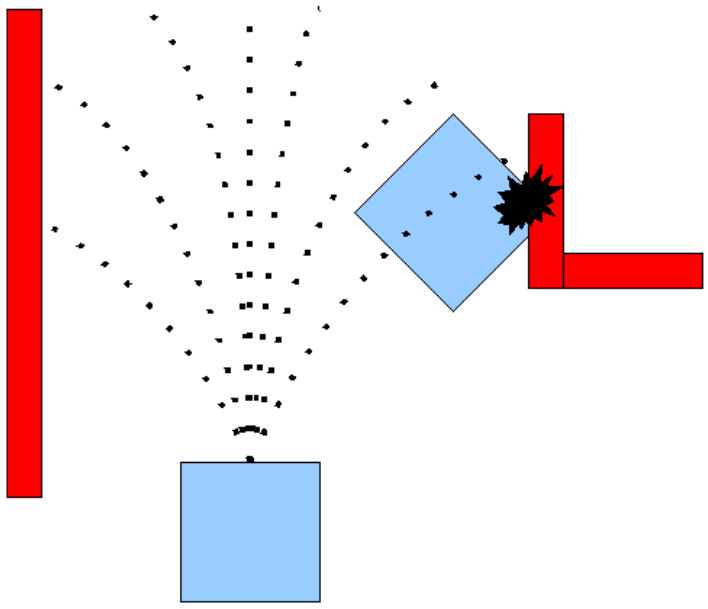
1. Diskreetselt mõõta robotit ümbritsevat ala (kiirus, suund)
2. Iga mõõdetud kiiruse kohta viia läbi edasiliikumise simulatsioon roboti hetkeseisust, et ennustada, mis võiks juhtuda, kui mõõdetud kiirust lühikest aega rakendatakse.
3. Hinnata edasiliikumise simuleerimise käigus saadud igat trajektoori, kasutades seejuures hindamiseks kaugust takistusest, kaugust sihtpunktist, kaugust globaalsest teekonnast ja kiirust. Kõrvaldada keelatud (takistustega lõikuvad) trajektoorid.
4. Valida kõige kõrgema hinde saanud trajektoor ja anda mobiilsele platvormile sellele vastav liikumissuund ja –kiirus.
5. Korrata tegevust.

Trajektoori levimise algoritm ja dünaamilise akna lähenemise algoritm mõõdavad roboti ümber olevat ala erinevalt. Trajektoori levimise algoritm mõõdab võimalike liikumissuundi ja –kiiruseid kogu edasiliikumise simuleerimise aja jooksul, samal ajal kui dünaamilise akna

lähenemise algoritm mõõdab suunda ja kiirust ainult ühe perioodi kaupa, mistõttu on viimane meetod oluliselt efektiivsem, kuid samas ei pruugi tagada roboti liikumisel nii sujuvat kiirendust kui trajektoori levimise algoritm.

Trajektoori efektiivsemaks arvutamiseks kasutatakse võrestikku (ingl *map grid*), mille suurus on võrdne lokaalse maksumuskaardiga. Võrestik luuakse roboti ümber ning sellele märgitakse ka roboti globaalne teekond. See tähendab, et lahtritele, mis asuvad planeeritud teekonnal, antakse väärtus null. Ülejäänud lahtrite väärtuste arvutamisel lähtutakse nende kaugusest neile kõige lähema nullväärtusega lahtri suhtes.

Globaalse teekonna sihtpunkt (teekonna lõpp-punkt) võib tihtipeale jääda väljapoole lokaalseks planeerimiseks kasutatavat võrestikku (roboti lokaalset ala). See tähendab, et trajektooridele arvutamisel peab arvestama lokaalse sihtpunktiga. Lokaalne sihtpunkt on esimene teekonna punkt lokaalses alas ning sellele järgnev punkt asub juba väljaspool lokaalset ala.



Sele 2.14 Lokaalne teekonna planeerimine.

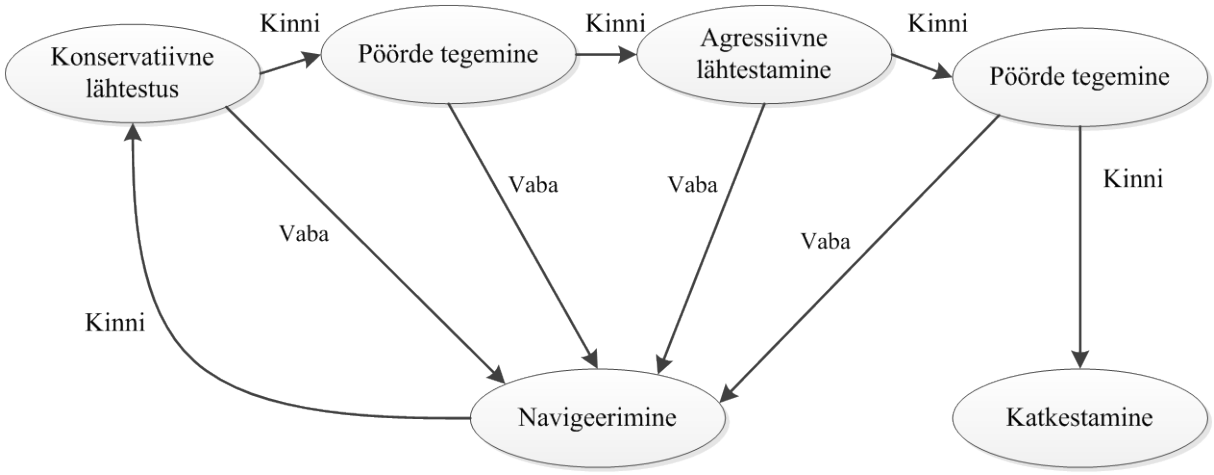
2.4.4. Roboti käitumine kinnijäämisel

Roboti liikumisel mööda planeeritud teekonda võib siiski tekkida ka olukord, kus robot jääb kinni. Sellises olukorras võib robot enda vabastamiseks läbi viia rida tegevusi (sele 2.15). Enda vabastamiseks üritab robot vaikimisi läbi teha järgmised tegevused:

1. Kõik takistused, mis asuvad väljaspool kasutaja poolt määratud ala, eemaldatakse kaardilt (konservatiivne lähenemine).
2. Kui võimalik, sooritab robot ühel kohal olles pöörde (ingl *in-place rotation*)

3. Kui pärast pööret ei ole robot vabaks pääsenud, eemaldatakse kaardilt kõik takistused, mis asuvad väljaspool ristkülikukujulist ala, milles robot saab ennast pöörata (agressiivne lähtestamine).
4. Järgmisena üritab robot sooritada veel ühe pöörde.
5. Kui robot pole jätkuvalt vabaks pääsenud, kuulutab ta oma sihtpunkti kättesaamatuks ning katkestab oma tegevuse.

Antud järjekorras viiakse tegevused läbi siiski vaikimisi seadistuse korral, st ROS-is on võimalik seadistada, kuidas robot peaks kinnijäämise korral käituma. Antud töö raames kasutati neid seadistusi siiski ei muudetud.



Sele 2.15 Roboti käitumine kinnijäämisel.

2.5. Roboti asukoha tuvastamine

Roboti liikumisel keskkonnas on väga oluline teada, missugune on roboti asukoht ja suund. Ebatäpsete andmete korral võib juhtuda, et robot kaldub teekonnast kõrvale, jääb kuhugi kinni, põrkub takistusega ning seetõttu ei jõua näiteks ka oma sihtkohta. Paljud robotid, sh ka antud töös kasutatav Robotino XT, on võimelised läbitud vahemaad mõõtma, kasutades selleks mootoritel asetsevaid koodreid. Koodrite abil on võimalik loendada, kui palju pöördeid on mootor teatud aja jooksul teinud, ning teades muid läbitud teekonna arvutamiseks vajalikke parameetreid, nt mootori ülekandesuhe, ratta läbimõõt, on võimalik läbitud vahemaa arvutada. See tähendab, et koodreid kasutades on võimalik roboti asukohta ruumis ainult kaudselt kindlaks teha. Paraku ei ole antud meetod roboti asukoha määramiseks piisavalt täpne, näiteks võivad roboti liikumisel rattad libiseda, mis tähendab, et roboti tegelik asukoht erineb välja arvutatud asukohast. Roboti tegeliku asukoha määramiseks oleks seega vaja kasutada täiendavat lahendust, mis annab robotile tagasisidet tema tegeliku asukohta kohta ja mille abil on robot võimeline kõrvalekallet planeeritud teekonnast korrigeerima.

Erinevaid lahendused, kuidas robot saab tagasisidet oma asukoha kohta ruumis, on välja toodud peatükis 1. Antud töös välja pakutud lahendus kasutab roboti asukoha pidevaks jälgimiseks kaamerat, mis edastab pealtvaates videosignaali, st kaamera on paigaldatud jälgitava ala kohale. Kaameralt saadud andmete abil planeeritakse robotile teekond, registreeritakse teekonnale sattuvaid takistusi, kuid jälgitakse ka roboti asukohta ning antakse robotile selle kohta tagasisidet.

Roboti asukoha tuvastamiseks kaamerapildis on oluline suuta robotit teda ümbritsevast keskkonnast igal ajahetkel selgelt eristada. Üks võimalike viise roboti enda leidmiseks kaardil oleks, sarnaselt teiste objektide tuvastamisega, tehisnärvivõrkude kasutamine. Sellisel lahendusel oleks siiski mõningaid puudusi:

- Sõltuvalt roboti kujust ei pruugi olla võimalik roboti suunda määrata, nt ringikujulise ristlõike puhul.
- Roboti kujuga sarnased objektid võivad näida kui robot ise. See tähendab, et vaateväljas näib olevat korraga rohkem kui üks robot, mis võib süsteemi segadusse ajada.

Üks viise, kuidas on võimalik nii roboti asukohta kui ka suunda tuvastada, oleks geomeetriliste kujundite ja mustrite kasutamine robotil. Antud kujundid peavad siiski täitma mitmeid tingimusi:

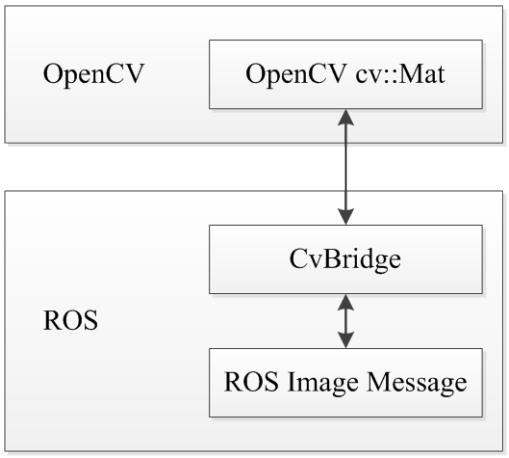
- Kujund või kujundid peavad olema sellise kujuga või paiknema robotil nii, et nende asendi põhjal on võimalik kindlaks teha roboti suund.
- Kujund peab oma unikaalse kuju või mustriga, et välistada selle korduvust. Sarnase kujundi ilmumisel kaamera vaatevälja võib näida, et vaateväljas on kaks robotit, ja sel viisil jällegi süsteemi segadusse ajada.

Antud töös on lähemalt uuritud lahendust, kus objektide, antud juhul roboti, asukoha tuvastamiseks kasutatakse geomeetrilisi kujundeid.

Lahenduse tööpõhimõte seisneb selles, et masinnägemissüsteemile antakse ette, missugust konkreetset kujundit kaamerapildist otsitakse. Antud kujund on märgitud robotile, st roboti asukoht tuvastatakse antud kujundi asukoha kaudu. Kuna roboti asukoht peab olema määratav igal ajahetkel, eeldab see, et robot on koguaeg kaamerapildis ning süsteem jälgib pidevalt tema asukohta ja suunda.

2.5.1 OpenCV kasutamine asukoha tuvastamiseks

Roboti asukoha tuvastamiseks ja jälgimiseks pealtvaates kaamerapildis kasutatakse OpenCV (Open Source Computer Vison) teeke, mis on mõeldud reaalaja masinnägemissüsteemides kasutamiseks. OpenCV teegid ühilduvad erinevate tarkvara platvormidega ja on seejuures ka üks ROS-i komponente, mistõttu on ka roboti asukoha tuvastamine hetkel tehtud ROS-i baasil, kasutades selleks ROS-i OpenCV teeke (sele 2.16).



Sele 2.16 Andmevahetus OpenCV ja ROS-i vahel.

Nagu eespool juba mainitud, antakse süsteemile ette kujund ja seda kujundit otsitakse kaamerapildist. Selleks kasutatakse funktsiooni matchTemplate. Funktsiooni abil on võimalik sisendpildilt otsida suvalist mustrit või objekti, mis on mallina ette antud [13]. Roboti asukoha leidmiseks on sisendiks määratud kaamera poolt edastatav videopilt ning malliks on geomeetriline kujund, mida kaamerapildilt otsitakse. Funktsioonil on kokku neli parameetrit:

- Sisendpilt, kust otsitakse mallile vastet (kaameralt tulev videopilt)
- Mall, mida sisendpildilt otsitakse (kujund, mida kaamerapildist otsitakse)
- Väljundmassiiv (võrdluse tulemused)
- Meetod, millega omavahel malli ja sisendpilti võrreldakse

Mallile vaste leidmiseks sisendpildis nihutatakse malli ühe piksli kaupa sisendpildis nii horisontaalses kui ka vertikaalses suunas. Kasutades eelnevalt määratletud meetodit, arvutatakse malliga kattuva pildiosa sarnasus malliga ning võrdlustulemused salvestatakse väljundmassiivis. Kokku on kuus erinevat meetodit, mille abil on võimalik malli ja sisendpilti omavahel võrrelda [13]:

- CV_TM_SQDIFF (*square difference*)

$$R(x,y) = \sum_{x',y'} \left(T(x',y') - I(x+x',y+y')\right)^2 \tag{2.9}$$

- CV_TM_SQDIFF_NORMED

$$R(x,y) = \frac{\sum_{x',y'} \left(T(x',y') - I(x+x',y+y')\right)^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}} \tag{2.10}$$

- CV_TM_CCORR (*correlation*)

$$R(x,y) = \sum_{x',y'} \left(T(x',y') \cdot I(x+x',y+y')\right) \tag{2.11}$$

- CV_TM_CCORR_NORMED

$$R(x,y) = \frac{\sum_{x',y'} \left(T(x',y') \cdot I(x+x',y+y')\right)}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}} \tag{2.12}$$

- CV_TM_CCOEFF (*correlation coefficient*)

$$R(x,y) = \sum_{x',y'} \left(T'(x',y') \cdot I'(x+x',y+y')\right), \tag{2.13}$$

kus

$$T'(x',y') = T(x',y') - \frac{1}{w \cdot h} \cdot \sum_{x'',y''} T(x'',y'') \tag{2.14}$$

$$I'(x+x',y+\ y') = I(x+x',y+y') - \frac{1}{w \cdot h} \cdot \sum_{x'',y''} I(x+x'',y+y'') \tag{2.15}$$

- CV_TM_CCOEFF_NORMED

$$R(x,y) = \frac{\sum_{x',y'} \left(T'(x',y') \cdot I'(x+x',y+y')\right)}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}} \tag{2.16}$$

Kasutades normeeritud (ingl *normalized*) meetodeid (valemid 2.10, 2.12, 2.16), avaldab valgustuse erinevus malli ja sisendpildi vahel väiksemat mõju [12]. See tähendab, et antud meetodid on sisendpildi valgustingimuste muutuste suhtes vähemtundlik ja võimaldab seega pildilt soovitud kujundit suurema tõenäosusega leida. Lisaks jäävad funktsiooni väljundväärtused normeeritud meetodite kasutamisel vahemikku 0 kuni 1 ja on täpsusega 10^{-6}. Sõltuvalt funktsioonis valitud meetodist võib mallile sisendpildis leitud parim vaste olla, kas

minimaalse või maksimaalse väärtusega, täieliku vaste leidmisel on funktsiooni väljundväärtused seega null või üks.

Näiteks arvutab funktsioon meetodite CV_TM_SQDIFF ja CV_TM_SQDIFF_NORMED kasutamisel malli ja sisendpildi erinevuste ruudu. See tähendab, et mida väiksem on väljundi väärtus seda tõenäolisemalt leiti mallile pildist õige vaste ning mallis antud kujundile ideaalse vaste leidmisel sisendpildis võrdub funktsiooni väljundväärtus nulliga.

Vaatamata sellele, et normeeritud meetodite kasutamine matchTemplate-i funktsioonis võimaldab soovitud kujundeid tuvastada ka keerulisemates valgustingimustes, jääb valgustus ja ka teised mõjutajad (varjud) sellegipoolest mingil määral videopilti mõjutama. See tähendab, et isegi kui malliks olev kujund, mida sisendpildilt otsitakse, on üles võetud ja välja lõigatud sellest samast sisendpildist, mida kaamera kuvab, ei pruugi matchTemplate-i funktsioon leida täielikku vastet kujundile (väljundväärtused üks või null). Seetõttu on kujundi tuvastamiseks kaamerapildis mõistlik kasutada lävendit, st teatud väljundväärtusest suuremate või väiksemate väärtuste korral loetakse kujund kaamerapildis leituks.

Funktsioon matchTemplate võrdleb omavahel malli ja sisendpilti ning salvestab tulemused väljundmassiivi. Kasutades ainuüksi matchTemplate-i funktsiooni aga ei ole võimalik kindlaks teha, kus asub parim vaste mallile sisendpildis ehk kus robot asub. Selleks on täiendavalt vaja kasutada funktsiooni minMaxLoc. Funktsiooni minMaxLoc sisendiks on funktsiooni matchTemplate väljundmassiiv, millest leitakse ekstreemväärtused, st väljundmassiivi väikseim ja suurim väärtus, ja nende asukohad pildil. Kokku on funktsioonil minMaxLoc viis parameetrit:

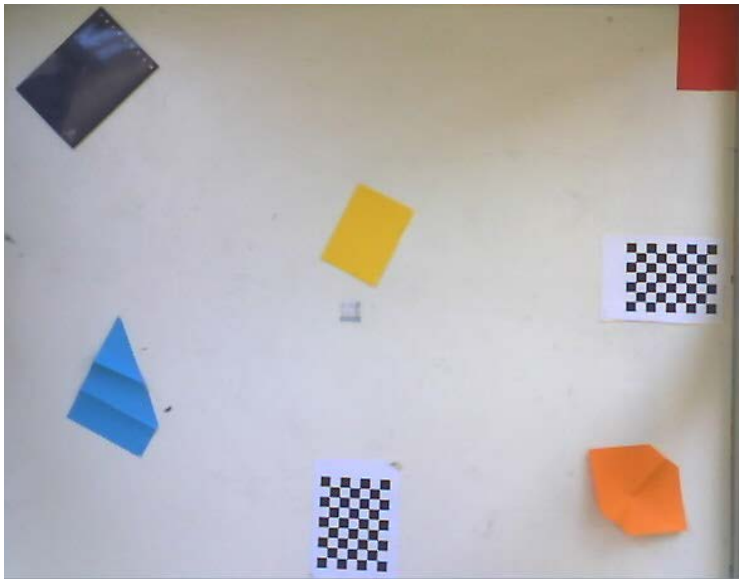
- Sisendmassiv (funktsiooni matchTemplate väljund)
- Sisendmassiivi minimaalne ja maksimaalne väärtus
- Minimaalse ja maksimaalse väärtuse asukoht pildil
- Mask alammassiivi valimiseks (valikuline)

Kokkuvõtvalt, kasutades OpenCV teegi funktsioone matchTemplate ja minMaxLoc on võimalik kaamera poolt edastatavas pildis tuvastada roboti asukoht ning jälgida roboti liikumist. Antud süsteemi eelduseks on, et robot peab olema ülejäänud keskkonnast selgesti eristuma. See saavutatakse roboti märgistamisega unikaalse kujundiga, mida süsteem pildilt otsib.

3. KATSED

3.1. Süsteemi seadistus

Erinevate katsete tegemiseks on antud töös kasutatud väljakut, mille pikkus on 1,92 m ja laius 1,63 m (sele 3.1). Antud väljakul simuleeritakse keskkonda, milles robot peab hakkama saama. Väljaku eri kohtadesse paigutatakse objekte ja takistusi, katsetatakse klassifikaatori tööd takistuste äratundmiseks, testitakse teekonna planeerimise toimimist ning roboti asukoha ja suuna tuvastamist. Väljaku kohale on paigaldatud kaamera, mis edastab oma videopilti arvutile. Antud arvuti omakorda suhtleb traadita ühenduse kaudu robotiga, st kaamerast saadetud andmete põhjal saab arvuti tagasisidet keskkonna ja roboti kohta ning juhib robotit (sele 3.2).



Sele 3.1 Väljak ja sellel paiknevad objektid.

3.1.1. Robotino XT

Riistavaralise baasina kasutatakse antud töös Robotino XT robotplatvormi, mille olulisemad tehnilised andmed on toodud tabelis 3.1.

Tabel 3.1 Robotino XT tehnilised parameetrid

Parameeter	Väärtus
Diameeter	370 mm
Kõrgus	260 mm
Kummist riba ümber roboti koos põrkeanduritega	

9 analoog infrapuna kaugusandurit
IEEE 802.11 standardil põhinev traadita ühendus (WiFi)

Antud töös siiski infrapuna kaugusandureid ei kasutatud, kuid tulevikus oleks kindlasti mõistlik need süsteemi integreerida, võimaldamaks mitmel erineval moel teele sattunud takistusi vältida. Üheks olulisemaks parameetriks, millega peab teekonna planeerimisel arvestama on roboti diameeter, st ei tohi olla võimalik planeerida sellist teekonda, kust robot oma mõõtmete tõttu läbi ei mahuks.

Nagu eelnevalt ka juba kirjutatud, on oluline, et teekonna planeerimisel arvestatakse roboti diameetriga: roboti jaoks ei tohi planeerida sellist teekonda, millest ta tegelikkuses läbi ei mahu.

3.1.2. Kaamera

Kaamerana on antud töös kasutatud Logitech C250 veebikaamerat. Kaamera tehnilised parameetrid on järgmised:

Tabel 3.2 Logitech C250 veebikaamera tehnilised andmed.

Parameeter	Väärtus
Lahutusvõime	640 × 480 pikslit
Vaatenurk (diagonaal)	63°
Suurim kaadrisagedus	30 kaadrit sekundis
Fookuskaugus	2 mm

Kaamera paigaldatakse väljaku kohale nii, et väljaku pikim külg (1,92 m) jääb täies ulatuses kaamerapilti. Kuna kaamerapildi külgede suhe on 4:3, siis väljaku lühim külg jääb pildile 1,44 m ulatuses. Võttes arvesse kaamera vaatenurka (63°) ning külgede suhet arvutatakse välja, kui kõrgele antud väljaku kohale tuleb kaamera paigutada. Selleks kasutatakse valemit

$$h = \frac{l}{2\tan\frac{\alpha}{2}}, \tag{3.1}$$

kus

l – väljaku diagonaali pikkus [m],

α – kaamera vaatenurk, [°].

Saadakse, et kaamera oleks vaja paigaldada ~1,96 m kõrgusele väljaku kohale.

Teades kaamera lahutusvõimet, saab arvutada kaamera resolutsiooni, st väikseima detaili suurust, mida kaamera suudab eristada. Tähistades sensori pikema külje resolutsiooni w_{sr}ja väljaku pikima külje$w_{väljak}$, arvutatakse lahutusvõime valemiga

$$R = \frac{w_{väljak}}{w_{sr}}, \tag{3.1}$$

seega

$$R = \frac{1920}{640} = 3\frac{mm}{piksel}$$

See tähendab, et üks piksel kaamera sensoris võrdub 3 mm väljakul ning sellest väiksemaid objekte ei suuda kaamera eristada. Töös läbi viidud katsete jaoks on antud resolutsioonist piisav, kuid resolutsiooniga peab siiski arvestama geomeetrilise kujundi, millega roboti asukohta määratakse, mõõtmete ja mustri valiku juures. Kaamera suurem lahutusvõime kindlasti võimaldab eristada väiksemaid objekte ja seeläbi pakkuda detailsemat pilti.

3.1.3. Arvuti

Töös kasutatud arvuti on DELL E6400 sülearvuti Ubuntu 12.04 32-bitise operatsioonisüsteemiga. Arvutile on ka paigaldatud ROS tarkvara raamistik (versioon Groovy Galapagos), mida kasutatakse Logitechi kaameralt pildi vastuvõtmiseks ning robotiga suhtlemiseks.

Nagu eespool juba mainitud, toimib andmevahetus arvuti ja Robotino XT vahel traadita ühenduse kaudu. Andmevahetuseks kasutatakse IEEE 802.11 standardit ehk WiFi-ühendust. WiFi ühenduspunkti (ingl *access point*) loob Robotino XT ning arvuti ühendatakse selle külge.

Veebikaamera on arvutiga ühendatud USB-kaabli kaudu. Pildiedastamiseks kaamerast ROS-i on kasutatud ROS-i uvc_camera paketti, mis sisaldab endas tüüreleid USB Video Klassi (UVC) seadmetele. Antud standard katab ühilduvuse suurema osa veebikaameratega ning võimaldab erinevaid kaamera parameetreid seadistada [14].

Erinevate töö käigus tehtud katsete ajal on valgustingimused erinenud – lisaks kunstlikule valgustusele (laevalgustus) mõjutas keskkonda ka looduslik valgus, sest katsetuste tegemiseks kasutatud väljaku läheduses asus aken, millest sissetulev päikesevalgus ruumi heledust oluliselt muutis. Parima pildi saamiseks, st seadistamaks kaamerat nii, et pilt ei üle- ega alavalgustatud oleks, oli enne katsete algust vaja kaamerat seadistada. Selleks kasutati GUVCView nimelist vabavaralist tarkvara, mis kasutab veebikaameraga ühildumiseks Linuxi UVC tüüreleid. Kuigi kaamera erinevaid parameetreid on võimalik ka ROS-i kaudu muuta, on kaamera seadistamine GUVCView kaudu hetkel oluliselt mugavam ja töö käigus eelistati viimast varianti.

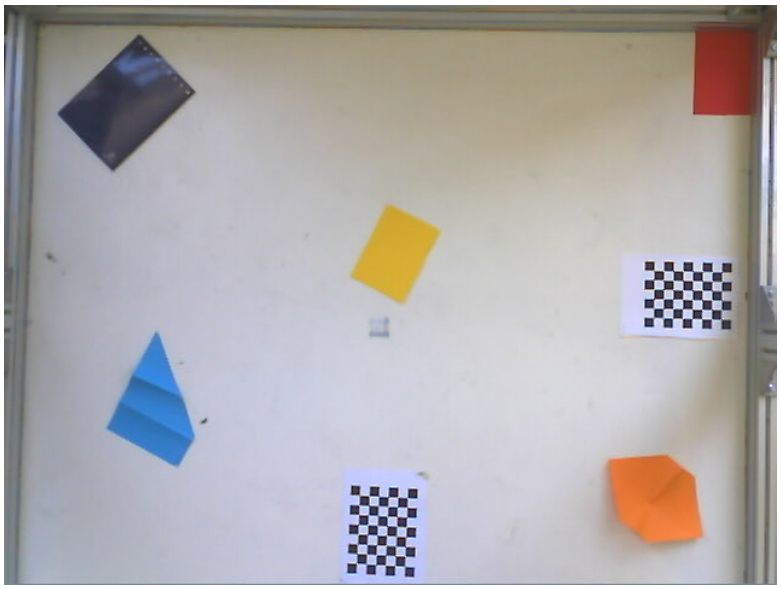
Sele 3.2 Süsteemi seadistus

3.2. Klassifikaatori katsed

Klassifikaatorit kasutatakse antud töös objektide – takistused, vaba põrandapind – tuvastamiseks väljaku pinnal. Täpsemalt põhineb klassifikaatori töö tehisnärvivõrkude treenimise kaudu objektide äratundmisel.

Tehisnärvivõrkude treenimisel antakse klassifikaatorile sisenditeks pilt, millelt soovitakse objekte leida, ignoreeritava ala mask ning tunnuste mask.

Sisendpilt peab kasutama RGB või RGBA värvimudelit. Tüüpilised RGB või RGBA värvimudelit kasutavad formaadid on vastavalt JPEG ja PNG. Pildi iga värvikanali kohta on närvivõrgu sisendis üks kaart (kokku kolm). Töös on sisendpildina kasutatud kaamera kaudu salvestatud üht kaadrit väljakust koos sellel paiknevate objektidega (sele 3.3).

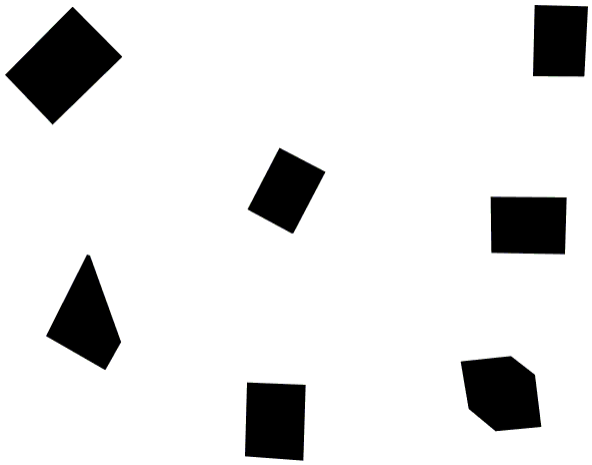


Sele 3.3 Väljakul paiknevad objektid.

Ignoreeritava ala maskiga saab pildil välistada tunnuseid, mis võivad osutuda närvivõrgu treenimisel segavaks või mida lihtsalt soovitakse välistada. Selleks võivad näiteks olla objektide varjud. Antud juhul on servades pildile jäänud väljaku ääred, kuid katsete käigus soovitakse neid ignoreerida (sele 3.4).

Sele 3.4 Ignoreeritava ala mask.

Tunnuste maski abil määratakse pildil need objektid, mida soovitakse pildil tuvastada, antud juhul takistused, mida robot liikumisel vältima peaks. Alternatiivina võiks takistuste asemel hoopis otsida vaba põrandapinda, kuid hetkel on siiski keskendutud objektide kui takistuste otsimisele. Iga tunnusmaski kohta on närvivõrgul üks väljund (sele 3.5).



Sele 3.5 Tunnuste mask.

Sisendpildist luuakse omakorda 29 x 29 piksli suurustest mustritest kaks andmebaasi. Üht andmebaasi kasutatakse hiljem klassifitseerimisvõimekuse katsetamiseks ning teise andmebaasi mustrid antakse ette närvivõrgule selle õpetamiseks. Närvivõrkude treenimiseks kasutatavate mustrite arv on seejuures muudetav ning mõjutab oluliselt närvivõrkude treenimiseks kuluvat aega. Lisaks on muudetav ka närvivõrgu esimese ja teise peidetud kihi ning lineaarse klassifikaatori neuronite arvu. Katsed erineva mustrihulga ja neuronite arvu juures on toodud tabelites 3.3 ja 3.4.

Tabel 3.3 Närvivõrkude õpetamine, kasutades meetodit A.

Katse nr	1	2	3	4	5	6
Andmekogum:						
Treenimine	30000	10000	30000	10000	5000	5000
Testimine	3000	1000	3000	1000	500	500
Närvivõrk:						
Tunnuseid esimesed peidetud kihis	6	6	3	3	3	3
Tunnuseid teises peidetud kihis	50	50	25	25	20	10
Lineaarse klassifitseerija neuronite arv	100	100	50	50	40	20
Epohhe	1	1	1	1	1	1
Klassifitseerimise võimekus	**99,53%**	**99,4%**	**99,43%**	**99,10%**	**94,60%**	**94,00%**
Aeg, s	**150**	**50**	**38**	**13**	**5**	**2,5**

Tabel 3.4 Närvivõrkude õpetamine, kasutades meetodit B.

Katse nr	1	2	3	4	5	6
Andmekogum:						
Treenimine	30000	10000	30000	10000	5000	5000
Testimine	3000	1000	3000	1000	500	50
Närvivõrk:						
Tunnuseid esimesed peidetud kihis	6	6	3	3	3	3
Tunnuseid teises peidetud kihis	50	50	25	25	20	10
Lineaarse klassifitseerija neuronite arv	100	100	50	50	40	20
Epohhe	1	1	1	1	1	1
Klassifitseerimise võimekus	**99,30%**	**99,10%**	**99,03%**	**98,90%**	**97,60%**	**97,80%**
Aeg, s	**150**	**50**	**38**	**13**	**5**	**2,5**

Treenimistulemuste mõõtmiseks klassifitseeritakse kõik katsebaasis olevad mustrid.

Nagu näha, siis väiksema mustrite hulga ning neuronite arvu juures muutub närvivõrkude treenimine oluliselt kiiremaks (tabel 3.3; 3,4), samas aga ei lange klassifitseerimisvõimekus oluliselt. Alles mustrite arvu vähendamisel 5000-ni ja neuronite arvu vähendamisel esimeses peidetud kihis, teises peidetud kihis ja klassifikaatoris vastavalt 3-e, 20-e ja 40-ni langeb klassifikaatori võimekus rohkem.

See võib olla tingitud asjaolust, et andmehulk, st sisendpilt ei ole väga keerukas, sest sisaldab lihtsasti eristavaid objekte ning palju heledat põrandapinda. Seda tõestab ka asjaolu, et antud klassifitseerimisvõimekuse saavutamiseks piisas ühest treeningtsüklist.

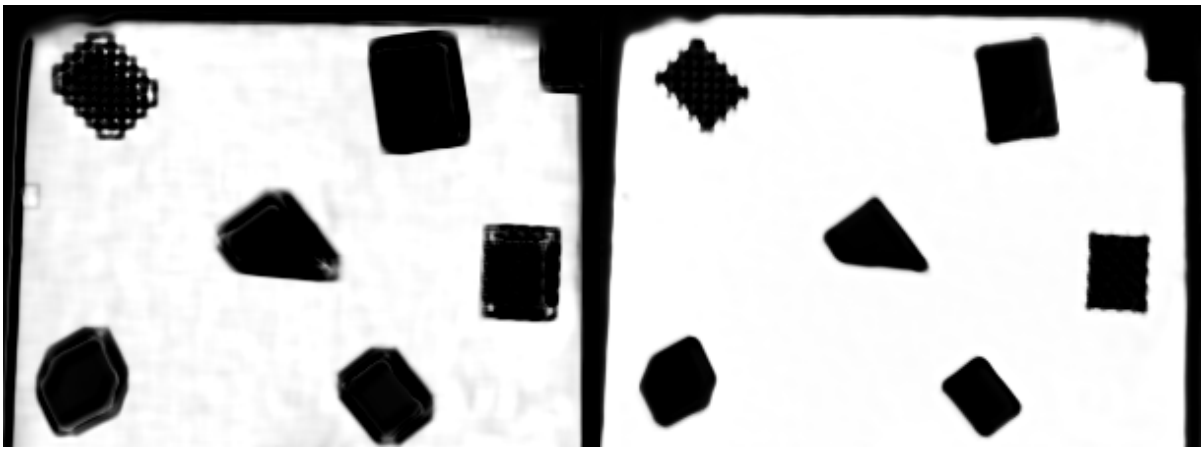
Lisaks ei ole ka sisendpildi lahutusvõime väga suur (640 × 480 pikslit) ning katab suhteliselt väikest ala. Reaalsetes oludes peaks süsteem katma oluliselt suurema maa-ala ja ilmselt kasutama ka parema lahutusvõimega seadmeid. See tähendab, et reaalsetes oludes tuleb treenimiseks kasutada rohkem mustreid ning suuremaid närvivõrke.

Kuigi meetod A ja meetod B erinevad oma lähenemise poolest andmete klassifitseerimisele, on kumbagi meetodiga tehtud katsete tulemused suhteliselt sarnased ning võisid samade parameetritega tehtud katse korral isegi varieeruda. Väikseid erinevusi siiski on märgata: suurema närvivõrgu ning treenimiseks kasutatavate mustrite arvu juures annab meetod A parema võimekuse, samas aga mustrite arvu ja närvivõrgu suuruse vähendamisel langeb

meetodi B võimekus vähem.

Seda on näha ka, kui võrrelda erinevate katsete klassifikaatori väljundeid (seled 3.7 .ja 3.8) Klassifikaatori toimivuse kontrollimiseks on sellesse laetud pilt samast väljakust, kuid objektid on ringi tõstetud (sele 3.6).

Sele 3.6 Väljakul ringi tõstetud objektid.



Sele 3.7 Meetodi A (vasakul) ja meetodi B väljundid, kasutades treenimiseks suurt võrku ja andmekogu (katse 1).

Sele 3.8 Meetodi A (vasakul) ja meetodi B väljundid, kasutades treenimiseks väikest võrku ja andmekogu (katse 6).

Meetodi B abil treenitud närvivõrgu väljundi tunnusmaskid on oluliselt detailsemad ning lähemad närvivõrgu treenimiseks kasutatud tunnusmaskile. Ka on mõistlikum kasutada pigem lihtsamaid närvivõrke ning vältida sel viisil närvivõrgu üleõpetamist. Väiksemate närvivõrkude õpetamine võtab ka oluliselt vähem aega (tabelid 3.3; 3,4), mis on süsteemi jõudluse kohalt väga oluline.

Arvestades, et mõlema meetodi puhul kulub närvivõrgu treenimiseks ühesuguste parameetrite korral sama palju aega, kuid väiksema mustrite arvu ja närvivõrgu suuruse juures on meetodi B abil treenitud võrgu väljund parem, on vähemalt antud katsetingimuste korral mõistlikum kasutada viimast meetodit.

Klassifikaatori väljundmaskis, nagu ka närvivõrgu sisendina kasutatud tunnustemaskis, on tuvastavad objektid tumedad (mustad) ning vaba põrandapind hele (valge). Antud klassifikaatori väljundmaski saab kasutada sisendina teekonna planeerimiseks ROS-is.

3.3. Teekonna planeerimise toimimine

Teekonna planeerimiseks ROS-is tuleb kõigepealt välja arvutada maksumuskaart, st leida iga piksli läbimise hind kaardil ning seejärel, andes robotile sihtpunkt, arvutada odavaim teekond sihtkohta jõudmiseks.

Kuigi maksumuskaardi saab arvutada ka klassifikaator ise, kasutatakse hetkel klassifikaatori väljundmaski maksumuskaardi arvutamiseks ROS-is. See on hetkel võimalik, kuna kõik objektid, mida soovitakse pildil tuvastada, on koondatud ühte klassi. Mitme tunnusmaski kasutamisel tuleks väljundina kasutada klassifikaatori loodud kaarti. Selline olukord oleks vajalik, kui pildil on vaja tuvastada enam kui kaks objektiklassi, millele on vaja määrata

erinevad hinnad, mida võetakse teekonna planeerimisel arvesse. Antud töös aga eristatakse ainult vaba põrandapinda, millel robot liikuda võib, ning objekte (takistusi), mida robot peab vältima.

Katsete tegemiseks on kasutatud üht staatilist pilti klassifikaatori väljundist, st mitte reaalajas kaamerast tulevat pilti. Lisaks on antud pilt hetkel ROS-is seadistatud kui staatiline kaart, st teekonna planeerimist katsetatakse muutumatu pildi põhjal. Kaamerast tuleva elava pildi põhjal objektide tuvastamist ja teekonna planeerimist ei jõutud antud töö käigus veel teostada ja omavahel ning oleks seega edasine arendustegevus.

Staatilist maksumuskaarti on aga võimalik mitmel moel veel seadistada:

- *resolution* – kaardi eraldusvõime. Eelnevalt arvutati välja, et 640 x 480 pikslise lahutusvõime korral vastab ühele pikslile 3 mm e. 0,003 m piksli kohta.
- *origin* – Vasaku alumise piksli koordinaadid kaardil (x-koordinaat, y-koordinaat, nurk). Vaikimisi on koordinaadid (0,0,0).
- *occupied_thresh* – pikslid, mille hõivatuse tõenäosus on lävendist kõrgem loetakse hõivatuks. Antud katsete puhul on selle väärtuseks seadistatud 0,75.
- *free_thresh* – pikslid, mis loetakse antud lävendist allapoole jääva väärtuse korral täiesti vabaks.

  Katsete käigus kasutati väärtust 0,1.
- *negate* – heleda ja tumeda värvi tõlgenduse vahetamine omavahel. Vaikeväärtus on null, mis jäeti muutmata. Antud parameetri väärtust „1“ on vaja kasutada, kui sisendiks on klassifikaatori loodud kaart, kus värvide tähendused on vastupidised – must on vaba pind ja hele on hõivatud ala.

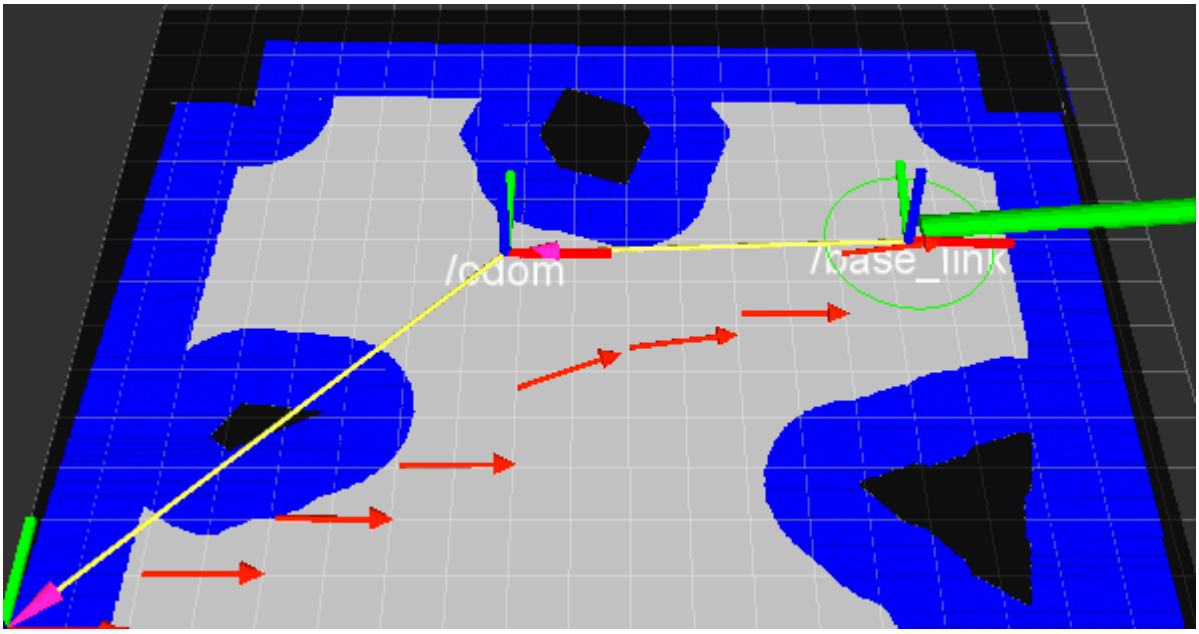
Resolutsiooni kaudu määratakse seega ära, kui suurt ala katab pilt reaalsuses, ning teisendatakse pikslid meetriteks. Antud parameetri kaudu seega määratakse, kui palju robot peab väljakul liikuma, kui koordinaadid antakse ette pikslites.

Staatilise kaardi sisendina võib kasutada suvalist pilti – see võib olla ka värvipilt, mille põhjal ROS ise, vastavalt ette antud parameetritele, maksumuskaardi arvutab. See siiski ei anna nii häid tulemusi, kui närvivõrkude kasutamine objektide identifitseerimiseks (sele 3.9).

Sele 3.9 Töötlemata pildi kasutamine kaardi loomiseks.

Robotile teekonna planeerimisel tuleb arvestada ka roboti mõõtmetega, st ei tohi olla võimalik planeerida sellist teekonda, kust mõnest lõigust ei mahu robot läbi, ega anda ette sellist sihtpunkti, kuhu robot oma mõõtmete tõttu ei suuda liikuda. Teades roboti mõõtmeid , on ROS-is võimalik märgistada objektide ümber ala, mis on võrdne roboti raadiusega. Teekonna planeerimisel nüüd jälgitakse, et roboti keskpunkt ei satuks antud ala sisse, vastasel korral tähendaks see kokkupõrget objektiga.

Roboti teekonna planeerimise visualiseerimiseks ning roboti juhtimiseks kasutatakse ROS-i Rviz tööriista. Rviz'i laetakse väljaku maksumuskaart koos vastavate parameetritega, roboti mudel (mõõtmed ja roboti erinevate detailid koordinaatsüsteemide vahelised seosed). Rviz'i kaudu on võimalik robotile anda ka sihtpunkt, kuhu robot peab liikuma. Sihtpunkti määramisel kontrollitakse, kas robotil on võimalik antud sihtpunkti liikuda. Sobiva sihtpunkti andmisel arvutatakse odavaim teekond ning antakse käsk roboti liikumiseks mööda arvutatud teekonda.

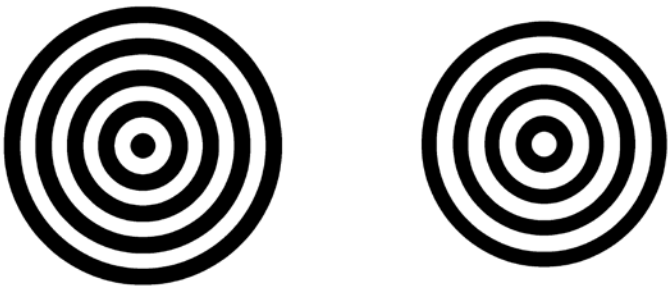


Sele 3.10 Roboti teekonna planeerimisel roboti mõõtmetega arvestamine ja teekonna planeerimine Rviz'is.

Antud töö käigus ei jõutud Robotino juhtimisele sisse ehitada tagasiside andmist tema asukoha kohta kaamerapildi põhjal. See tähendab, et roboti asukoht kaardil tuleb esmalt Rviz'is käsitsi määrata ning roboti poolt maha sõidetud teekonda mõõdetakse roboti hodomeetri andmete põhjal. Vaatamata sellele, oli robot siiski võimeline väljakul liikuma, kui talle ette anda sihtpunkt.

3.4. Roboti tuvastamine kaamera abil

Roboti tuvastamiseks kaamerapildis märgistatakse robot sellise kujundiga, mis ülejäänud keskkonnast selgelt eristub. Antud töös kasutatakse roboti tuvastamiseks kahest üksteisest erinevat kontsentrilistest ringjoontest koosnevat kujundit (sele 3.11).



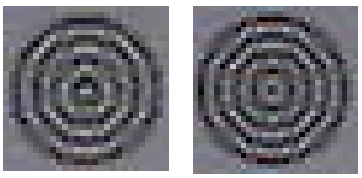
Sele 3.11 Kontsentrilised ringjooned, mida kasutatakse roboti asukoha tuvastamiseks.

Selline kujundi valik võimaldab lisaks asukohale määrata ka roboti asendit. Asendi leidmiseks käsitletakse ringjooni kui kaht eraldi kujundit, st iga kujundi leidmiseks on oma matchTemplate-i funktsioon ning minimaalsete ja maksimaalsete väärtuste asukohad, mis minMaxLoc funktsiooni abil leitakse.

Kuna matchTemplate funktsioon võrdleb talle ette antud malli sisendpildiga, nihutades malli piksli haaval vertikaalses ja horisontaalses suunas, ei ole funktsioon võimeline leidma otsitavat kujundit, kui seda on sisendpildis suvalise nurga võrra keeratud. Antud töös aga soovitakse just leida roboti asendit, st nurka, mille võrra on robot ennast pööranud. Siinkohal ongi lahenduseks just ringjoonte kasutamine otsitava mustrina – ringjoon ise on nurgainvariantne kujund, st ükskõik, kui palju robot ennast ka pöörab, ei muutu tema asend malli suhtes ning seetõttu peaks ringjoontest koosnev muster olema igal hetkel matchTemplate funktsiooni abil leitav.

Ühe ainsa ringjoontest koosneva kujundi abil ei ole võimalik roboti asendit (nurka) siiski tuvastada – selle leidmiseks vaadeldakse kahe mustri (kontsentrilised ringjooned) keskpunktide asendit üksteise suhtes. Selleks loetakse üht ringjoont liikumatuks ning teist ringjoont esimese ümber tiirlevaks mustriks. Mustrite keskpunktide vaheline kaugus moodustab täisnurkse kolmnurga hüpotenuusi ning täisnurga kaatetiteks on ringjoonte keskpunktidest horisontaal- ja vertikaalsihis väljuvad ristuvad sirged. Nurga leidmiseks kasutatakse arkustangensfunktsiooni, st see leitakse kaatetite pikkuste kaudu. Arvesse võetakse ka, millises sektoris tiirlev kujund asub, ning sellest lähtuvalt arvutatakse nurk vahemikus 0° - 360°.

Nagu eelnevalt juba kirjutati, saab matchTemplate funktsioonis valida kuue erineva meetodi vahel kuidas pildilt soovitud mustrit otsida. Allpool on nende meetodite toimivust katsetatud. Mallina on kasutatud pilte mustritest, mis on salvestatud kaamera kaudu (sele 3.12)

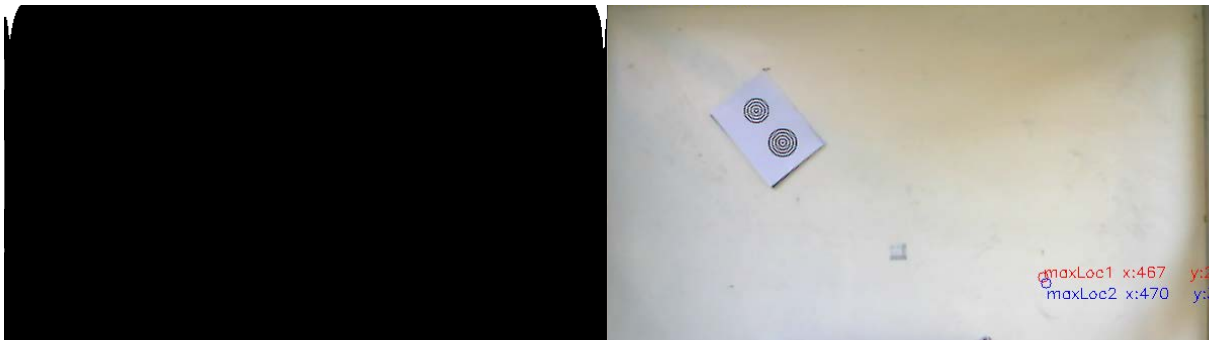


Sele 3.12 Mallidena kasutatud mustrid.

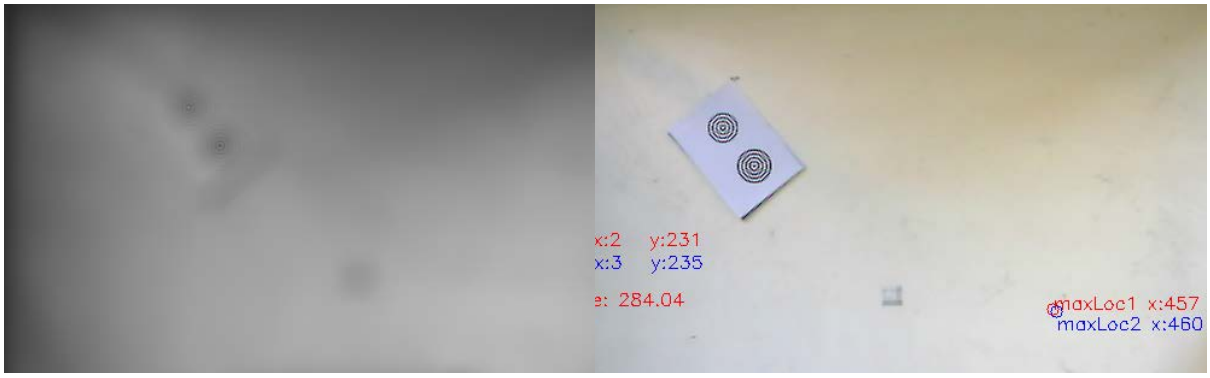
Sisendpildina, millelt mustrite otsitakse, on kasutatud kaamera poolt reaalajas edastatud pilti. Kõik katsed on seejuures tehtud praktiliselt ühel ajal, st katsetingimused olid konstantsed.

Sõltuvalt kasutatud meetodist, on võimalik lävendi abil määratleda, missugusest minimaalsest

või maksimaalsest väljundväärtusest alates loetakse mustrit leituks. Sobivate väljundväärtuste korral leitakse järgmise sammuna juba kujundi asend. Katsete läbiviimiseks kirjutatud programmi leiab lisast 1 ning katsetulemused on näidatud seledel 3.13 – 3.18.



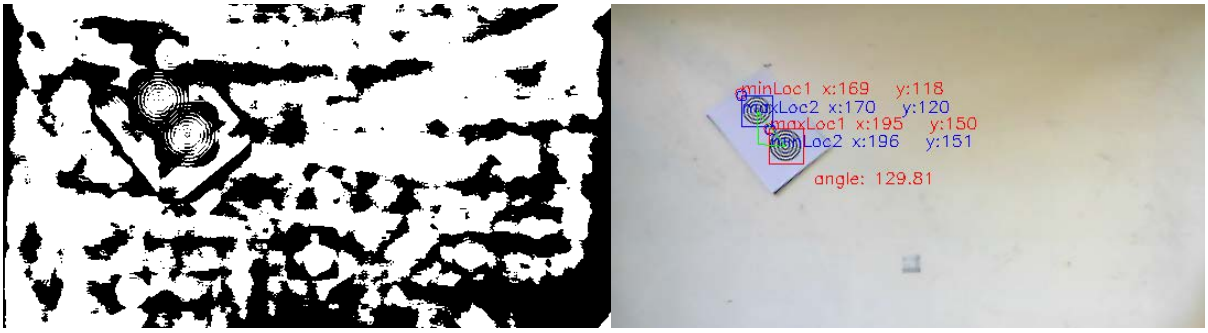
Sele 3.13 meetod CV_TM_SQDIFF



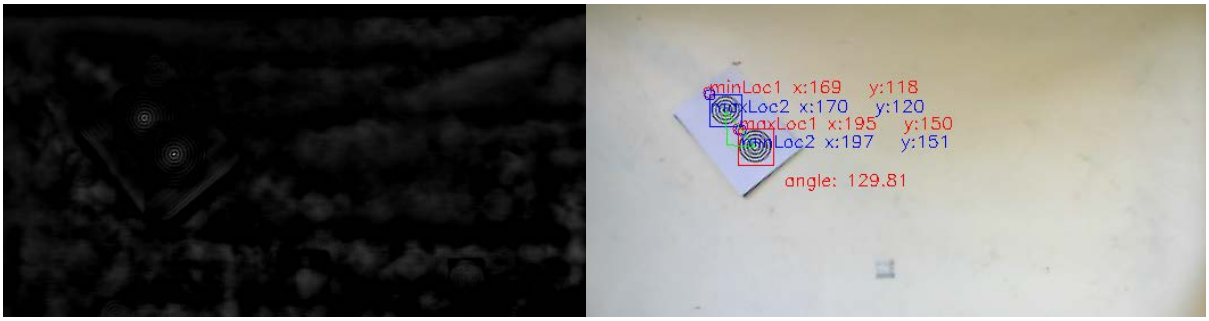
Sele 3.14 meetod CV_TM_SQDIFF_NORMED

Sele 3.15 meetod CV_TM_CCORR

Sele 3.16 meetod CV_TM_CCORR_NORMED



Sele 3.17 meetod CV_TM_CCOEFF



Sele 3.18 meetod CV_TM_CCOEFF_NORMED

Katsete käigus selgus, et kõige paremini suutsid antud katsetingimustes mustrit tuvastada meetodid CV_TM_CCOEFF ja CV_TM_CCOEFF_NORMED. Teised meetodid ei suutnud tihtipeale üldse mustrit pildis leida, st minimaalsed või maksimaalsed väljundväärtused ei kattunud mustri asukohaga pildil. Esmastes katsetes kasutati staatilist sisendpilti, millest suutsid ka teised meetodid edukalt mustri leida, kuid veebikaamera reaalajas edastatava videopildi kasutamisel sisendina, toimisid eespool nimetatud meetodid kõige paremini.

4. OHUTUSEANALÜÜS

Ohutuseanalüüsi eesmärgiks on leida ohud, mida antud lahendus võib töökeskkonnas töötaja tervisele või keskkonnale endale tekitada. Tööstuses kasutatavate lahenduste väljatöötamisel peab eelkõige arvestama lahenduste töökindluse ja ohutusega, eriti veel, kui samas keskkonnas liiguvad ringi ka inimesed.

Antud lahendus sisaldab ainult üht liikuvat komponenti – see on robot. Suurimaks ohuks inimesele kujutabki olukord, kus roboti juhtimises peaks tekkima häireid ning selle tagajärjel sõidab robot inimesele otsa. Antud töös on katsetatud lahendust väikese ala peal ja väikese robotiga, mis väga suurt ohtu ei kujuta. Vaatamata oma väikestele mõõtmetele (diameeter 370 mm, kõrgus 260 mm), on Robotino XT mass 11 kg, seega jalale otsa sõites, võib robot sellegipoolest haiget teha.

Antud töös välja pakutud lahenduse üks rakendusi võiks olla mobiilsete platvormide juhtimine. Mobiilseid platvorme saab tööstuses kasutada näiteks erinevate asjade transportimiseks. Taoliste platvormide mõõtmed ja mass on Robotinoga võrreldes oluliselt suuremad, põhjustades seega kokkupõrkel inimesega ka oluliselt tõsisemaid vigastusi.

Antud töös välja pakutud süsteem kasutab mitut komponenti, mille ülesütlemisel on kogu süsteemi töö häiritud või isegi peatatud:

- Süsteemi keskseks komponendiks on arvuti, mis võtab vastu andmeid kaameralt, töötleb neid andmeid ja suhtleb robotplatvormiga. Häired arvuti töös võivad seega mõjutada kogu süsteemi.
- Robotplatvormi ja arvuti vahel toimub andmevahetus traadita ühenduse kaudu (WiFi-ühendus), mille katkemisel pole enam võimalik robotile käske edastada. Lisaks on võimalik WiFi-ühenduse kvaliteeti suhteliselt lihtne mõjutada, seadistades näiteks samale sagedusele teine võrk.
- Süsteem kasutab kaamerat robotplatvormi asukoha määramiseks ning selle põhjal paranduste tegemiseks teekonnas, kui robot peaks liikumise käigus oma teekonnast kõrvale kalduma. Kaamera ülesütlemisel puuduks tagasiside roboti asukoha kohta ruumis, st robot liiguks pimesi ning ei näeks, kui tema teele satuks takistus, põhjustades niiviisi kokkupõrke.
- Võib ka juhtuda, et robot lihtsalt väljub mingil põhjusel kaamera vaateväljast, mis samuti põhjustab olukorra, kus puudub tagasiside roboti asukoha kohta.

Kuna kogu lahenduse ainsaks liikuvaks osaks on robotplatvorm, on vaja tagada just selle ohutus keskkonna ja inimese jaoks. Süsteem tuleks seega projekteerida nii, et igasuguste häirete tekkimisel andmevahetuses, jääks robot seisma.

Reaalne on ka olukord, kus andmevahetus erinevate komponentide vahel toimib, kuid kaamerapildis lihtsalt ei suudeta objekti tuvastada, mille tulemusena toimuks jällegi kokkupõrge. Selle välistamiseks on robotil võimalik kasutada ka täiendavaid kaugusandureid.

5. EDASISED TEGEVUSED JA ARENDUSIDEED

Antud töös ei õnnestunud täielikult kõiki süsteemi osasid omavahel liidestada, st terviklikku reaalajas toimivat lahendust ei jõutud valmis teha. Katsetati erinevaid komponente – närvivõrke objektide klassifitseerimiseks, roboti asukoha tuvastamist mustrite otsimise meetodil kaamerapildis, maksumuskaardi arvutamist ja teekonna planeerimist ROS-is ja liikumise simuleerimist, kasutades selleks klassifikaatori väljundit. Üks edaspidiseid tegevusi oleks kindlasti antud komponentide omavaheline liidestamine.

Kui aga objektide klassifitseerimiseks hakatakse kasutama reaalajas kaamerapilti, kerkib väga olulisele kohale süsteemi jõudlus – objektide tuvastamisel ei tohi olla suurt ajaviidet. Seega oleks üks edasistest tegevustest klassifikaatori jõudluse katsetamine kaamerapildi põhjal.

Katsete tegemiseks on töös kasutatud väikest väljakut ning kesise lahutusvõimega veebikaamerat, millest katsete tegemiseks on küll piisav, kuid ei kajasta reaalset olukorda. Parema ettekujutuse saamiseks välja pakutud lahenduse toimivuse kohta, tuleks kindlasti katsetada seda realistlikumas keskkonnas ja oluliselt suurema ala peal.

Sõltuvalt tööstushoone ehitusest ja plaanist ei pruugi üks kaamera kogu ala, millel robot liikuma peaks, ära katta. Terve ala katmiseks tuleks seega kasutada rohkem kui üht kaamerat.

Töös ei katsetatud ka roboti asukoha (mustri) tuvastamiseks hierarhilist mustriotsingut, mille abil on võimalik kiirendada mustri otsimist sisendpildis. Selle mõju avaldub, kui mustrit otsitakse suuremate piltide (suurem ala ja lahutusvõime) pealt. Antud töös olid pildid väikesed ning mustri otsing töötas piisavalt kiiresti. Reaalsetes oludes oleks kindlasti seda mõistlik kasutada.

Töös kasutatud kaamera on arvutiga ühendatud USB-kaabli kaudu. Vältimaks signaalikadu ei tohi kaamera arvutist väga kaugel asuda. USB 1.1 standardi kasutamisel ei ole soovituslik kvaliteedikao vältimiseks kasutada pikemat kaablit kui 5 meetrit. See tähendab, et suurte tööstuslike ruumide katmiseks konkreetne lahendus ei sobi. Alternatiivina saaks pildiedastuseks kasutada IP-kaameraid, mida saab arvutiga ühendada, kasutades ära juba olemasolevat IT taristut (Ethernet võrk) või, nagu ka robot, juhtmevabalt arvutiga suhelda.

Üks võimalikke tulevikulahendusi võiks olla ka kaamera ja roboti otsene suhtlemine arvuti vahenduseta ehk kogu andmetöötlus ja juhtimine teostatakse roboti riistvara peal. Siinkohal siiski mängib olulist rolli süsteemi tööks nõutud ressurss (närvivõrkude treenimine, objektide klassifitseerimine, mustriotsing suurelt alalt) ning kasutatav riistvara (arvutusvõimsus, energiatarve ja roboti akude mahtuvus).

KOKKUVÕTE

Käesoleva magistritöö eesmärgiks oli välja töötada alternatiivne lahendus tööstuslike robotplatvormide juhtimiseks ning katsetada selle lahenduse toimivust riistvaralise platvormi peal.

Esmalt uuriti olemasolevaid autonoomsete robotite juhtimiseks kasutatavad lahendusi. Need võib põhimõtteliselt jagada kahte gruppi: muutumatu ja muutuva teekonnaga lahendusteks. Muutumatu teekonnaga lahenduste kasutamisel antakse robotile ette üks kindel trajektoor, mis tavaliselt hoone põrandale monteeritakse (värviline või magnetlint, voolujuhe) ja mida mööda robot liikuma peab. Taolised lahendused on küll lihtsad, kuid selles seisneb ka nende nõrkus: robot ei ole võimeline mööda sõitma tema teekonnale sattunud takistusest.

Levinumad muutuva teekonnaga lahendused kasutavad platvormi juhtimiseks laserit, güroskoopi, kaameraid. See tähendab, et robot on teda ümbritsevast keskkonnast oluliselt teadlikum ning suudab andurite abil teda ümbritsevaid takistusi juba vältida. Vaatamata sellele, peab robot siiski ka teadma oma asukohta ruumis. Selleks kaardistatakse ja salvestatakse eelnevalt kogu ala, millel robot liigub. Hoone plaani muutumisel tähendab see, et kogu ala tuleb uuesti roboti poolt jälle kaardistada.

Alternatiivse lahendusena tööstuslike robotplatvormide juhtimiseks pakutakse välja pealtvaates kaamera kasutamise, mis reaalajas jälgib kaamera vaatevälja jäävat ala ning roboti liikumist sellel alal. Kasutuselolevate lahenduste ees omab see mitut eelist: saadakse pidevat tagasisidet roboti tegeliku asukoha kohta kaardil ja korraga jälgitakse oluliselt suuremat ala, kui robotil paiknevad andurid võimaldaksid.

Töös välja pakutud süsteem koosneb kolmest põhilisest komponendist: laes paiknev kaamera, mis igal ajahetkel jälgib maapinnal toimuvat; arvuti, mis võtab vastu kaamerapilti ja suhtleb robotplatvormiga; robotplatvorm, mis võtab arvutilt käske vastu ning samal ajal ka saadab arvutile infot. Andmevahetus arvuti ja roboti vahel toimub juhtmevabalt. Sõltuvalt aga keskkonnast ja hoones olevast taristust, võib andmevahetus kaamera ja arvuti vahel toimuda samuti juhtmevabalt või kasutada arvutivõrku. Katsete tegemiseks on antud töös kasutatud veidi lihtsustatud süsteemi: veebikaamera ühendamiseks arvutiga kasutati USB-kaablit.

Roboti teekonna planeerimiseks on oluline teada, kus paiknevad takistused, mida robot oma liikumisel alguspunktist sihtpunkti vältima peab. Takistuste ja vaba põrandapinna tuvastamiseks kasutatakse antud töös tehisnärvivõrke. Tehisnärvivõrkude kasutamise mõte

seisneb selles, et neid on võimalik mingi kindla ülesande täitmiseks treenida. Treenimiseks kasutatakse teadaolevate sisendite ja väljunditega andmetekogumit. Treenimise käigus võrreldakse võrgu väljundeid ja teadaolevaid väljundeid ning modifitseeritakse kaalukoefitsiente nii, et teadaolevate väljundväärtuste ja tehisnärvivõrgu väljundväärtuste erinevus oleks minimaalne.

Kuigi piisavalt suuri tehisnärvivõrke on võimalik õpetada aproksimeerima suvalist funktsiooni, võib see protsess olla väga ajakulukas. Reaalajas objektide tuvastamisel kaamerast on aga jõudlus oluline ning seetõttu on mõistlikum kasutada konvolutsioonilisi närvivõrke.

Antud töös on närvivõrgu treenimisel sisenditeks vaadeldava ala pilt, mis jupitatakse väiksemateks mustriteks ja koostatakse nendest andmekogu. Töös on võrreldud ka kaht erinevat meetodit antud andmekogu moodustamiseks ja nende mõju närvivõrkude võimekusele objektide tuvastamiseks.

Teades takistuste paiknemist hoone pinnal, tuleb robotile planeerida trajektoor alguspunktist sihtpunkti. Selleks kasutatakse spetsiaalset kaarti, kus on iga punkti läbimise hind välja arvutatud (maksumuskaart). Teades iga punkti hinda, leitakse kõige odavam teekond sihtpunkti jõudmiseks. Seejuures arvestatakse teekonna planeerimisel ka roboti mõõtmetega, st ei ole võimalik planeerida sellist teekonda, mida robot ei suudaks oma dimensioonide tõttu realiseerida.

Maksumuskaardi arvutamiseks ja teekonna planeerimiseks kasutatakse Robotic Operating System'i (ROS) tarkvararaamistikku, mis sisaldab erinevaid teeke, tüüreleid, visualiseerimistööriistu ning tuge erinevatele robotitele. See lihtsustab oluliselt robotite jaoks erinevate rakenduste loomist ning vähendab robotite seadistamiseks ja kasutuselevõtmiseks kuluvat aega.

Roboti liikumisel keskkonnas on oluline ka igal ajahetkel teada roboti asukohta. Kuigi robot mõõdab koodrite abil läbitud vahemaad ka ise ja edastab vastavaid andmeid arvutile, ei ole selline mõõteviis absoluutselt täpne ning ajapikku võib mõõteviga akumuleeruda, st robot ei pruugi enam asuda seal, kus ta arvab olevat. Laes paikneva kaamera kaudu on aga võimalik tuvastada roboti tegelik asukoht ning anda selle põhjal korrigeerida ka roboti asukohta.

Üks variante roboti tuvastamiseks hoone pinnalt on roboti märgistamine mingi konkreetse mustriga. See tähendab, et kaamerapildist ei otsita otseselt robotit, vaid roboti asukohta

tehakse kindlaks otsitava mustri leidmisel. Muster tuleks seejuures valida niimoodi, et selle kaudu oleks võimalik kindlaks teha nii roboti asukoht kui ka suund.

Erinevate katsete tegemiseks kasutati väikest väljakut mõõtmetega 1,92 m x 1,63 m. Pildiedastamiseks kasutati väljaku kohale paigaldatud veebikaamerat Logitech C250. Robotiks oli Robotino XT mobiilne platvorm.

Katsetuste käigus hinnati erineva suurusega närvivõrkude võimekust ja nende õpetamiseks kulunud aega. Võrreldi ka kaht erinevat andmehulkade moodustamiseks kasutatud meetodit. Selgus, et katsetes kasutatud objektide tuvastamiseks ei olnud väga suurt andmehulka ega ka keerulisi närvivõrke vaja kasutada. 30000 mustri kasutamisel andmehulgas ning 6, 50 ja 100 neuroni kasutamisel vastavalt esimeses, teises peidetud kihis ja lineaarses klassifikaatoris kulus närvivõrgu ühe perioodi treenimiseks ~150 sekundit. Samas 10000 mustri ning poole väiksema neuronite arvu kasutamisel mainitud kihtides vähenes õpetamiseks kulunud aeg 13 sekundile, samal ajal ei langenud võimekus praktiliselt üldse.

Katsetati ka matchTemplate-i funktsiooni ja selle erinevate meetodite võimekust roboti asukoha ja asendi leidmiseks. Selleks otsiti kaamerapildilt kontsentrilistest ringjoontest koosnevat kujundit. Leiti, et parema tulemuse andsid CV_TM_CCOEFF ja CV_TM_CCOEFF_NORMED meetodid, st antud meetodite abil suudeti mustri asukoht ja asend pildist täpselt üles leida.

ROS-i abil aga katsetati reaalset roboti juhtimist, kasutades selleks Rviz'i visuaalset tööriista, kus teekonna planeerimiseks kasutati maksumuskaarti ning arvesse võeti ka roboti mõõtmeid.

Antud töö raames katsetati kõikide süsteemi toimimiseks vajalikke komponente, kuid omavahel neid komponente liidestada ei jõutud. See oleks kindlasti üks tulevikutegevusi.

Lisaks olid erinevad katsed läbi viidud suhteliselt lihtsates tingimustes, mis ei peegelda täielikult reaalseid olusid. Parema ülevaate saamiseks oleks mõistlik läbi viia katseid suurema ala peal ning tööstuslikus keskkonnas, kasutades selleks ka suurema lahutusvõimega kaamerat.

Töös välja pakutud lahendus võiks olla reaalne viis robotplatvormide juhtimiseks. Kõige suuremat riski kujutab lahenduse toimivusele hetkel objektide klassifitseerimise kiirus. Süsteemi toimimiseks peab ta suutma objekte reaalajas klassifitseerida. Riistvara arvutusjõudlus pidevalt aga kasvab ning tulevikus ei tohiks see kindlasti probleemiks osutuda.

SUMMARY

The purpose of this thesis is to develop an alternative solution for controlling industrial robotic platforms and test the solution using a real platform.

Firstly, current solutions for controlling autonomous robots were explored. Basically, these can be divided into two groups: solutions with a fixed path and solutions with an open path. Robots that are guided using a fixed path follow a certain trajectory that is marked on the floor. The trajectory may be a marked using a coloured or a magnetic tape but an electric wire as well. Solutions that use a fixed path for robot guiding are simple but there lies their weakness as well: robot cannot avoid obstacles that lie on their path.

Commonly used solutions which are based on an open path, use lasers, gyroscopes or cameras to guide robots. This means that robots are more aware of their surroundings and can avoid obstacles that are recognised by the sensors. However, robots still need to know their location in the area. A map of the area is constructed for that purpose. The downside is that whenever the layout of the area changes, a new map has to be constructed.

As an alternative solution an overhead camera can be used, which observes the area in its field of view and the movements of the robot as well. The proposed solution has several benefits over the currently used solutions: a constant feedback of the robot's actual position is given and in addition, a much larger area can be observed compared to sensors that are mounted on robots.

The solution suggested in this thesis consists of three main components: an overhead camera, which observes the area beneath it in real time; a computer, which receives the picture and communicates with the robotic platform, and as already mentioned, a robotic platform, which receives commands from the computer but sends feedback data to the computer as well. The computer and the robotic platform communicate with each other using a wireless connection. Depending on the environment and available infrastructure, a wireless connection may as well be used between the computer and camera to exchange data but an available computer network may too be used. A slightly simpler solution was used for testing: the camera was connected to the computer via the camera's USB cable.

It is important to know the location of objects (obstacles), which must be avoided, when planning a path for the robot. An artificial neural network is used to recognise obstacles and free space in the observed area. The idea of using artificial neural networks is that they can be

trained to perform a certain task. A set of inputs and known output values is used to train the artificial network. During training the network's output values are compared to the known output values and the weights are adjusted to minimise the difference between the output values.

Large enough artificial neural networks can be used to approximate practically any function, it may not be the most efficient solution. Training large network may be time-consuming and performance is important when detecting objects in real time from camera's image stream. Therefore, it is recommended to use convolutional neural networks.

An overhead image of the area is used as an input for training the neural network. This image is divided into smaller patterns, which make up the training data set. In addition, two different methods can be used for creating the training data sets. These methods are compared and also the capability of these two methods to detect objects is evaluated.

Knowing the location of obstacles, a path from the robot's current position to its destination must be planned. A special map (costmap), where every cell is given a certain value based on its occupancy, is used for path planning. Knowing the cost of every cell, the cheapest path from current position to the destination point is calculated. In addition, the robot's dimensions are taken as well into account when planning a path. This means that obstacles are inflated by the radius of the robot and a path, which the robot would not be able to complete due to its dimensions, cannot be set.

Robotic Operating System (ROS) is used for costmap creation and path planning. ROS provides different libraries, drivers and visualization tools and supports many robotic platforms, including the one used for testing in this thesis. Using ROS greatly simplifies the creation of different applications for robots and also reduces time that it takes to set up robots.

In addition to recognising obstacles in the observable area, it is important to know the robot's location at any given time. Although robots can estimate the distance travelled by using odometry information, this data can become inaccurate over time. By using an overhead camera to track the robot, the actual position can be found and used as feedback to the robot, i.e. this data can be used to adjust the robot's position.

One of the options for locating a robot in an image is by using a certain pattern to distinguish the robot from the rest of the environment. This pattern is then searched in the image. By

locating the pattern, the robot is located as well. The same pattern is also used to calculate the robot's orientation.

A small area with the size of 1.92 m x 1.63 m was used for testing. A Logitech C250 webcamera was used as the overhead camera and Robotino XT as the robotic platform.

During testing the classification capability of different-sized networks was evaluated as well as the time it took for their training. Moreover, the two methods for creating the data sets were compared. The tests showed that there was no need to use large networks or big training data sets for the images used. The classification rate remained practically unchanged when a neural network with 6, 50, 100 neurons in the first hidden, second hidden and linear classifier accordingly and a set consisting of 30000 patterns was used compared to a set of 10000 patterns and a neural network with 3, 25, 50 neurons in first hidden, second hidden and linear classifier.

The function matchTemplate with its different methods was tested for robot's position estimation. A pattern consisting of concentrical circles was used as the template that was being searched for in the source image (camera stream). The tests showed that the bests results were achieved using methods CV_TM_CCOEFF and CV_TM_CCOEFF_NORMED, i.e. the precise position and orientation of the pattern was found in the source image.

ROS and its visualization tool Rviz was used for the control of the robotic platform. A costmap was used for path planning with the robots dimensions taken into account.

The different components of the system were thoroughly tested. However, there was not enough time to develop a whole solution where all the components are seamlessly interfaced with each other. This is a task for the future.

Moreover, the experiments were performed in quite simple conditions which may not adequately reflect real environments. Therefore, it is advised to perform tests under conditions which mimic real industrial environments – a larger area and a camera with better resolution to provide more data.

In conclusion, the solution that is suggested in this thesis can be a realistic a new way for controlling robots. However, the biggest risk for the suggested solution is the classification performance of objects, since the system must be able to function in real time. However, there computational performance of computers is rising and performance should not be an issue.

KASUTATUD KIRJANDUS

1. Guidance and Navigation for Transbotics AGV and AGC automatic guided vehicle manufacturer. [WWW]
   http://www.transbotics.com/learning-center/guidance-navigation/
   (5.04.2013)
2. Barberá, H. M., Quiñonero, J. P. C., Izquierdo, M. A. Z., Skarmeta, A. G. S. . i-Fork: a Flexible AGV System using Topological and Grid Maps. Hispaania, Murcia Ülikool. 2003
3. Laser Target Technology. [WWW]
   http://www.aimagv.com/laser-target.html
   (5.04.2013)
4. How do the vehicles work in an AGV system? [WWW]
   http://www.mhi.org/downloads/industrygroups/agvs/elessons/vehicles-work-agv.pdf
   (5.04.2013)
5. Automated Guided Vehicles (AGV) [WWW]
   http://ichainnel.com/en/news/523128_ztj2yv-automated-guided-vehicles-agv/
   (6.04.2013)
6. Kelly, A., Nagy, B., Stager, Unnikrishnan, D. R. (2007). An Infrastructure-Free Automated Guided Vehicle Based on Computer Vision - *IEEE Robot. Automat. Mag, 14 (3). 24-34.* The Blue Social Bookmark and publication sharing system.
7. Hudjakov, R. Kaugmaa navigatsioonisüsteem maastikuvõimekusega autonoomsetele liikuritele: doktoritöö. Tallinna Tehnikaülikool, 2013.
8. Lumiste, R. Võrgustikud ja innovatsioon masina- ja elektroonikatööstuse arendamisel (Eesti juhtumite analüüs) : doktoritöö. Tallinna Tehnikaülikool. 2008.
9. Tehisnärvivõrgud ja nende rakendused. [WWW]
   http://www.dcc.ttu.ee/las/ISS0010/2012/ISS0010-8osa.pdf
   (15.04.2013)
10. ROS move base package. [WWW]
    http://wiki.ros.org/move_base
    (13.03.2013)
11. ROS costmap implementation. [WWW]
    http://wiki.ros.org/costmap_2d
    (13.03.2013)

12. Gerkey, B. P., Konolige, K. Planning and Control in Unstructured Terrain. Willow Garage, SRI Artificial Intelligence Center. 2008.
13. Bradski, G, Kaehler, A. Learning OpenCV Computer Vision with the OpenCV library. Sebastopol, O'Reilly Media. 2008.
14. USB Video Class. [WWW] http://www.usb.org/developers/devclass_docs/USB_Video_Class_1_5.zip (1.04.2014)

LISAD

LISA 1. Pildilt mustrit otsiva programmi kood.

```
#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/Image.h>
#include <sensor_msgs/image_encodings.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv/cv.h>
#include <math.h>
#include <opencv2/opencv.hpp>
#include <opencv2/nonfree/features2d.hpp>
#include <iostream>

namespace enc = sensor_msgs::image_encodings;

#define PI 3.14159265

static const char WINDOW[] = "Pattern Recognition";
static const char MATCH_METHOD[] = "matchTemplate Method";

class patternRecognition {

public:
    cv::Mat icon1;
    cv::Mat icon2;
    int method;

private:

    ros::NodeHandle nh;
    image_transport::ImageTransport it;
    image_transport::Publisher image_pub;
    image_transport::Subscriber image_sub;

    double m1,M1;
    double m2,M2;

    cv::Point point11;
    cv::Point point12;
    cv::Point point21;
    cv::Point point22;

    int font;
    char match_method;

public:
    patternRecognition()
        : it(nh)
    {

        image_pub = it.advertise("camera/pattern_recognition",1);
        image_sub          =          it.subscribe("image_raw",          1,
&patternRecognition::patternRec, this);

        cv::namedWindow(WINDOW,1);
        cv::namedWindow(MATCH_METHOD, 1);
```

```
    }
    ~patternRecognition()
    {
        cv::destroyWindow(WINDOW);
    }

    void patternRec(const sensor_msgs::ImageConstPtr& image){

        ros::Rate(1.0);
        cv::Mat camera;
        font = CV_FONT_HERSHEY_SIMPLEX;

        cv_bridge::CvImagePtr cv_ptr;
        try
        {
            cv_ptr = cv_bridge::toCvCopy(image, enc::BGR8);
        }
        catch (cv_bridge::Exception& e)
        {
            ROS_ERROR("cv_bridge exception: %s", e.what());
            return;
        }
        camera = cv_ptr->image;
        ROS_INFO("source: %i  %i",camera.cols, camera.rows);
        ROS_INFO("template: %i  %i", icon2.cols, icon2.rows);
        cv::imshow("pattern",icon1);

        if (method == 1)
        {
            match_method = CV_TM_SQDIFF;
        }

        else if (method == 2)
        {
            match_method = CV_TM_SQDIFF_NORMED;
        }
        else if (method == 3)
        {
            match_method = CV_TM_CCORR;
        }
        else if (method == 4)
        {
            match_method = CV_TM_CCORR_NORMED;
        }
        else if (method == 5)
        {
            match_method = CV_TM_CCOEFF;
        }
        else if (method == 6)
        {
            match_method = CV_TM_CCOEFF_NORMED;
        }

        int resultW1 = camera.cols - icon1.cols + 1;
        ROS_INFO("resultW1: %i", resultW1);
        int resultH1 = camera.rows - icon1.rows +1;
        cv::Mat result1 = cv::Mat(cvCreateImage(cvSize(resultW1, resultH1),
IPL_DEPTH_32F, 1));
        cv::matchTemplate(camera, icon1, result1, match_method);
        cv::minMaxLoc(result1,&m1,&M1,&point12,&point11);
        ROS_INFO("1 min: %f  max: %f", m1, M1);
```

```
        int resultW2 = camera.cols - icon2.cols + 1;
        int resultH2 = camera.rows - icon2.rows +1;
        cv::Mat result2 = cv::Mat(cvCreateImage(cvSize(resultW2, resultH2),
IPL_DEPTH_32F, 1));
        cv::matchTemplate(camera, icon2, result2, match_method);
        cv::minMaxLoc(result2,&m2,&M2,&point22,&point21);
        ROS_INFO("2 min: %f  max: %f", m2, M2);

        //maxLoc pt1
        cv::circle(camera,cv::Point(point11.x,
point11.y),5,cv::Scalar(0,0,255,0),1,0,0);
        char maxLoc1 [50];
        sprintf(maxLoc1,"maxLoc1 x:%i   y:%i", point11.x, point11.y);
        cv::putText(camera,  maxLoc1  ,cv::Point(  point11.x,  point11.y),
font, 0.5, cv::Scalar( 0, 0, 255, 0 ),1,0,false);

        //minLoc pt1
        cv::circle(camera,cv::Point(point12.x,
point12.y),5,cv::Scalar(0,0,255,0),1,0,0);
        char minLoc1 [50];
        sprintf(minLoc1,"minLoc1 x:%i   y:%i", point12.x, point12.y);
        cv::putText(camera,  minLoc1  ,cv::Point(  point12.x,  point12.y),
font, 0.5, cv::Scalar( 0, 0, 255, 0 ),1,0,false);


        //maxLoc pt2
        cv::circle(camera,cv::Point(point21.x,
point21.y),5,cv::Scalar(255,0,0,0),1,0,0);
        char maxLoc2 [50];
        sprintf(maxLoc2,"maxLoc2 x:%i   y:%i", point21.x, point21.y);
        cv::putText(camera,  maxLoc2  ,cv::Point(  point21.x,  point21.y+15),
font, 0.5, cv::Scalar( 255, 0, 0, 0 ),1,0,false);

        //minLoc pt2
        cv::circle(camera,cv::Point(point22.x,
point22.y),5,cv::Scalar(255,0,0,0),1,0,0);
        char minLoc2 [50];
        sprintf(minLoc2,"minLoc2 x:%i   y:%i", point22.x, point22.y);
        cv::putText(camera,  minLoc2  ,cv::Point(  point22.x,  point22.y+15),
font, 0.5, cv::Scalar( 255, 0, 0, 0 ),1,0,false);

        ROS_INFO("method: %i", match_method);

        if (method == 1 || method == 2)
        {
               ROS_INFO("method1_2: %i", match_method);
               if (m1 <= 0.2 && m2 <= 0.2)
               {

                    cv::rectangle(camera,  point12,  cv::Point(  point12.x  +
icon1.cols,  point12.y + icon1.rows ),  cv::Scalar( 0,  0,  255 ,  0 ),  1,  0,  0
);
                    cv::rectangle(camera,  point22,  cv::Point(  point22.x  +
icon2.cols,  point22.y + icon2.rows ),  cv::Scalar( 255,  0,  0 ,  0 ),  1,  0,  0
);

                    //triangle sides - draw line
                    int x1 = point12.x + (icon1.cols)/2;
                    int y1 = point12.y + (icon1.rows)/2;
                    int x2 = point22.x + (icon2.cols)/2;
```

```
                    int y2 = point22.y + (icon2.rows)/2;
                    int x3 = x2;
                    int y3 = y1;

                    //drawLine between pattern centers
                    cv::line(camera,cv::Point(x1,          y1),cv::Point(x2,
y2),cv::Scalar(0,255, 0,0),1,0,0);

                    //kaatetid
                    cv::line(camera,cv::Point(x1,          y1),cv::Point(x3,
y3),cv::Scalar(0,255, 0,0),1,0,0);
                    cv::line(camera,cv::Point(x2,          y2),cv::Point(x3,
y3),cv::Scalar(0,255, 0,0),1,0,0);

                    double angle;
                    angle = atan2(y3-y2,x3-x1)*180/PI;
                    if ( angle <0){
                       angle = 360 + angle;
                    }
                    char orientation [50];
                    sprintf(orientation,"angle: %.2f",angle);
                    ROS_INFO("angle: %f", angle);
                    cv::putText(camera,orientation,cvPoint(point12.x+40,
point12.y+50),font,0.5, cvScalar(0,0,255,0));
                }
        }

        else if (method == 6)
        {
             ROS_INFO("method6: %i", match_method);
             if (M1 >= 0.6 && M2 >= 0.6)
                 {

                 cv::rectangle(camera,  point11,  cv::Point(  point11.x  +
icon1.cols, point11.y + icon1.rows ), cv::Scalar( 0, 0, 255 , 0 ), 1, 0, 0
);
                 cv::rectangle(camera,  point21,  cv::Point(  point21.x  +
icon2.cols, point21.y + icon2.rows ), cv::Scalar( 255, 0, 0 , 0 ), 1, 0, 0
);

                 //triangle sides - draw line
                 int x1 = point11.x + (icon1.cols)/2;
                 int y1 = point11.y + (icon1.rows)/2;
                 int x2 = point21.x + (icon2.cols)/2;
                 int y2 = point21.y + (icon2.rows)/2;
                 int x3 = x2;
                 int y3 = y1;

                 //drawLine between pattern centers
                 cv::line(camera,cv::Point(x1,          y1),cv::Point(x2,
y2),cv::Scalar(0,255, 0,0),1,0,0);

                 //kaatetid
                 cv::line(camera,cv::Point(x1,          y1),cv::Point(x3,
y3),cv::Scalar(0,255, 0,0),1,0,0);
                 cv::line(camera,cv::Point(x2,          y2),cv::Point(x3,
y3),cv::Scalar(0,255, 0,0),1,0,0);

                 double angle;
                 angle = atan2(y3-y2,x3-x1)*180/PI;
                 if ( angle <0){
```

```
                    angle = 360 + angle;
                }
                char orientation [50];
                sprintf(orientation,"angle: %.2f",angle);
                ROS_INFO("angle: %f", angle);
                cv::putText(camera,orientation,cvPoint(point11.x+40,
point11.y+50),font,0.5, cvScalar(0,0,255,0));

            }
        }
        else if (method != 1 || method != 2 || method != 6)
        {
            if (M1 >= 0.8 && M2 >= 0.8)
                {
                cv::rectangle(camera,  point11,  cv::Point(  point11.x  +
icon1.cols, point11.y + icon1.rows ), cv::Scalar( 0, 0, 255 , 0 ), 1, 0, 0
);
                cv::rectangle(camera,  point21,  cv::Point(  point21.x  +
icon2.cols, point21.y + icon2.rows ), cv::Scalar( 255, 0, 0 , 0 ), 1, 0, 0
);

                //triangle sides - draw line
                int x1 = point11.x + (icon1.cols)/2;
                int y1 = point11.y + (icon1.rows)/2;
                int x2 = point21.x + (icon2.cols)/2;
                int y2 = point21.y + (icon2.rows)/2;
                int x3 = x2;
                int y3 = y1;

                //drawLine between pattern centers
                cv::line(camera,cv::Point(x1,            y1),cv::Point(x2,
y2),cv::Scalar(0,255, 0,0),1,0,0);

                //kaatetid
                cv::line(camera,cv::Point(x1,            y1),cv::Point(x3,
y3),cv::Scalar(0,255, 0,0),1,0,0);
                cv::line(camera,cv::Point(x2,            y2),cv::Point(x3,
y3),cv::Scalar(0,255, 0,0),1,0,0);

                double angle;
                angle = atan2(y3-y2,x3-x1)*180/PI;
                if ( angle <0){
                    angle = 360 + angle;
                }
                char orientation [50];
                sprintf(orientation,"angle: %.2f",angle);
                ROS_INFO("angle: %f", angle);
                cv::putText(camera,orientation,cvPoint(point11.x+40,
point11.y+50),font,0.5, cvScalar(0,0,255,0));
                }

        }


        cv::imshow(WINDOW, camera);
        cv::imshow(MATCH_METHOD, result1);

        cv::waitKey(3);

        image_pub.publish(cv_ptr->toImageMsg());
```

```
    }

};


int main(int argc, char** argv) {

    ros::init(argc,argv,"pattern_recognition");
    patternRecognition pat;
    pat.icon1=cv::imread(argv[1],1);
    pat.icon2=cv::imread(argv[2],1);
    std::cin >> pat.method;

    if (argc !=3 ){
        std::cerr << "Incorrect number of arguments" << std::endl;
    }
    ros::spin();
    return 0;
}
```