Nikita Pikkas

Masinnägemise kasutamine lahingurobotil

LÕPUTÖÖ

Tehnikainstituut
Robotitehnika õppekava
Juhendaja: Kristo Vaher

Tallinn 2023

Autori deklaratsioon ja lihtlitsents

Mina/meie,

Nikita Pikkas tõendan/tõendame, et lõputöö on minu/meie kirjutatud. Töö koostamisel kasutatud teiste autorite, sh juhendaja teostele on viidatud õiguspäraselt.

Kõik isiklikud ja varalised autoriõigused käesoleva lõputöö osas kuuluvad autori/te/le ainuisikuliselt ning need on kaitstud autoriõiguse seadusega.

Juhendaja (nimi, allkiri) Kristo Vaher

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, Nikita Pikkas
(*autori nimi*)
sünnikuupäev: 22.02.1999

annan Tallinna Tehnikakõrgkoolile (edaspidi kõrgkool) tasuta loa (lihtlitsentsi) enda loodud teose Masinnägemise kasutamine lahingurobotil (*lõputöö pealkiri*)

1. elektroonseks avaldamiseks kõrgkooli repositooriumi kaudu;
2. kui lõputöö avaldamisele on instituudi direktori korraldusega kehtestatud tähtajaline piirang, lõputöö avaldada pärast piirangu lõppemist.

Olen teadlik, et nimetatud õigused jäävad alles ka autorile ja kinnitan, et:

1. lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid ega muid õigusi;
2. PDF-failina esitatud töö vastab täielikult kirjalikult esitatud tööle.

Tallinnas, allkirjastatud digitaalselt.

LÜHENDID

PPM – Pulse Position Modulation (Impulsi asukoha modulatsioon)

PWM – *Pulse-width modulation* (Pulsilaiusmodulatsioon)

RC – *Radio Control* (Kaugjuhtimine)

BRG – *Blue Red Green* (Sinine, punane, roheline)

HSV – *Hue saturation value* (Värvitoon,saturatsioon,väärtus)

PID – Proportsionaal-integraal-diferentsiaal regulaator

OpenCV – *Open Source Computer Vision Library (*Avatud lähtekoodiga masinnägemise ja masinõppe tarkvara raamatukogu)

SISUKORD

JOONISTE TABEL

SISSEJUHATUS

Robootika valdkonnas on viimastel aastatel toimunud märkimisväärne areng, robotid leiavad rakendust alates tootmisest kuni tervishoiuni. Üks valdkond, mis on robootika kogukonnas tähelepanu pälvinud, on robotite kasutamine lahingutes. Seoses robotivõistluste kasvava populaarsusega suureneb nõudlus robotite järele, mis suudavad lahingu stsenaariumides hästi toimida. Sellega seoses on masinnägemise kasutamine osutunud paljulubavaks lähenemiseks, et parandada robotite sooritusvõimet lahingus.

Masinnägemine hõlmab algoritmide ja tehnikate kasutamist, mis võimaldavad arvutitel tõlgendada ja mõista ümbritseva maailma visuaalseid andmeid. Robootika kontekstis saab masinnägemist kasutada selleks, et anda robotitele võime tajuda end ümbritsevat keskkonda ja teha selle põhjal otsuseid. Konkreetsemalt saab masinnägemist kasutada selleks, et robotid suudaksid tuvastada ja jälgida objekte ning mustreid ja liikuda komplekssetes keskkondades.

Robotite võitluses annab masinnägemine robotitele konkurentsieelise. Kasutades kaameraid ja pilditöötlus algoritme, saavad robotid tuvastada oma vastaste asukohta ja orientatsiooni ning kasutada seda teavet oma liikumise ja rünnakustrateegiate kavandamiseks. Lisaks aitab masinnägemine robotitel navigeerida keerulistel lahinguväljakutel, vältida takistusi ja leida oma sihtmärke.

Robolahingus esitab areenikeskkond robotite tõhusaks tegutsemiseks keerulisi väljakutseid. Robotid peavad põiklema läbi mitmesuguste takistuste, sealhulgas teiste robotite ja mitmesuguste areeni omaduste, näiteks seinte, kirveste ja tõkete vahel. Lisaks sellele on keskkond sageli hämar ja muutuvate valgustingimustega, mis raskendab robotitel oma asukoha ja orientatsiooni täpset määramist. Ka lahingute kiire tempo on robotite jaoks kriitilise tähtsusega, et nad suudaksid oma vastased ajapiiridesse mahtudes täpselt tuvastada ning vastavalt reageerida. Selles kontekstis võib masinnägemise kasutamine oluliselt parandada robotite jõudlust robotite lahingus: andes täpset ja õigeaegset teavet robotite asukoha ja orientatsiooni kohta, võimaldab masinnägemine robotitel tõhusamalt liikuda ja kiiremini reageerida keskkonna muutustele.

Teine eelis, mida masinnägemise kasutamine robotitele võitluses annab, on võimalus saada täiendavaid kaaluboonuseid. Paljudel võistlustel on robotite kaalule seatud piirangud. Kasutades aga robotite piloodivabu juhtimismeetodeid, näiteks masinnägemist, võib robot saada kaaluboonust. Mõnel võistlusel võib see kaaluboonus olla kuni 20% roboti tavapärasest kaalupiirangust. Näiteks kui võistluse kaalulimiit on 30 kg, võib robot kaaluboonusega kanda täiendavalt 6 kg kaalu. See lisakaal võib anda lahingutes

märkimisväärse eelise, võimaldades robotil kanda täiendavat kaitset, relvi või muid kasulikke elemente, mis võivad tema jõudlust suurendada. Seega võib masinnägemise kasutamine lisaks roboti täpsuse ja jõudluse parandamisele anda robotite lahingus ka täiendavaid strateegilisi eeliseid.

Käesolevas autor uuris masinnägemise kasutamist robolahingus. Konkreetselt kasutas areeni laes asuvat laia vaatenurgaga kaamerat, et määrata roboti asukohta ja orientatsiooni ning liikumist, et teha kindlaks vaenlase robot. Roboti kontroll toimub läbi Pythoni skripti ja mooduli abil, mis kujutab endast USB PPM *pulse position modulation* (impulsi asukoha modulatsioon) muundurit, mis on ühendatud sülearvutiga ja RC(Kaugjuhtimine) saatjaga USB-kaabli kaudu. Lisaks autor on rakendanud värvide tuvastamise ja masinõppe meetodeid, et täpselt tuvastada oma robotit pildil. Kaameramooduli täpsuse suurendamiseks autor on loonud 3D-printeriga prinditud eraldi kinnituse, mis sisaldab mitut LED riba. Et valgusdioodidest kasu oleks, on roboti kerele paigutatud peegeldavad helkurid roboti orientatsiooni ja asukoha määramiseks. Töö käigus olid läbitud katsed, et hinnata masinnägemise kasutamise tõhusust, et tõsta võistlus roboti efektiivsust ning reageerimiskiirust lahingustsenaariumides.

1 ROBOTI VÕISTLUSED

Paljudes Euroopa riikides toimuvad robotivõistlused erinevates kaaluklassides, alates 150 grammist kuni 120 kilogrammini. Kõikide võistluste reeglid on sarnased ning kergelt erinevad, peamine on oma vastase roboti võitlusvõimetuks tegemine või hävitamine.

Siin on nimekiri kõige populaarsematest:

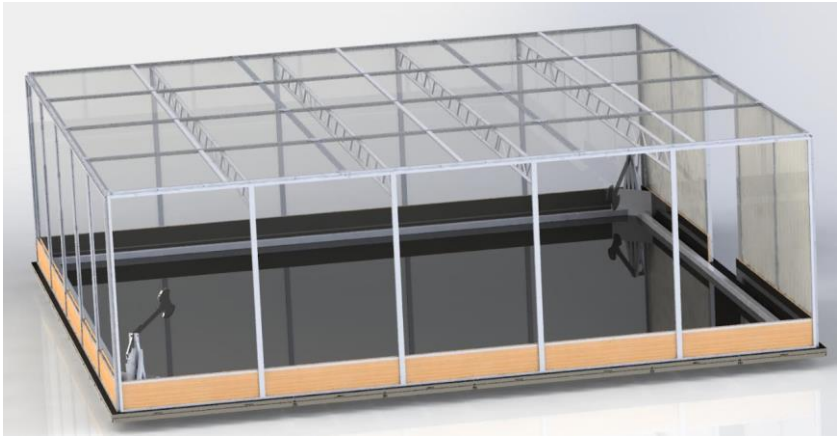
1 Robot Wars: See on tuntud võistlus, mis toimub Ühendkuningriigis. Selle käigus võitlevad robotid üksteisega areenil, eesmärgiga oma vastased liikumatuks muuta või hävitada. Võistlusel on mitu kaaluklassi, alates 150 g kuni 110 kg.
2 FMB maailmameistrivõistlused: See on Saksamaal toimuv võitlusrobootika võistlus. Sellel võistlusel võitlevad robotid üksteisega areenil, eesmärgiga saada punkte, lükates oma vastased areenilt välja või lammutades neid. Võistlusel on mitu kaaluklassi, alates 1,5 kg kuni 13,6 kg.
3 Dutch Robot Games: See on võitlusrobootika võistlus, mis toimub Hollandis. Selles osalevad robotid, mis võitlevad üksteisega areenil, eesmärgiga oma vastased liikumatuks muuta või hävitada. Võistlusel on mitu kaaluklassi, alates 150g kuni 120kg.
4 RoboChallenge: See on lahingurobotika võistlus, mis toimub Austrias. Selles osalevad robotid, mis võitlevad üksteisega areenil, eesmärgiga oma vastased liikumatuks muuta või hävitada. Võistlusel on mitu kaaluklassi, alates 150g kuni 120kg.
5 TechnoGames: See on Ühendkuningriigis toimuv võitlusrobootika võistlus. Selles osalevad robotid, mis võitlevad üksteisega areenil, eesmärgiga oma vastased liikumatuks muuta või hävitada. Võistlusel on mitu kaaluklassi, alates 150g kuni 100kg.
6 Robolahing: Robolahing on Eestis asuv puldiga juhitavatele või autonoomsetele lahingurobotitele mõeldud võistlus, kus masinad püüavad üksteist spetsiaalselt võitluseks ehitatud areenil kahjutuks teha. Võistlusel on mitu kaaluklassi, kuni 30 kg ning kuni 55kg.

Kuna Robolahingu reeglid lubavad autonoomseid roboteid kasutada ja annavad mehitamata robotite kasutamisel kaaluboonuse lisa 20%, samuti võimaldab meie ülikooli lähedus võistluspaigale lihtsat osalemist, otsustati ehitada autonoomne robot spetsiaalselt selle võistluse jaoks, kuid järgnevatel hooaegadel osaleda ka välisvõistlustel, et arendada meeskonda ja tõsta meie robotite võistlustaset ja kvaliteeti ning populariseerida meie kõrgkooli välismaal.

Sellel võistlusel saavad robotid osaleda kahes kaalukategoorias, raskekaal- kuni 55kg ja kergkaal- kuni 29,9kg. Võistluse eesmärk on koguda kokku tehnoloogiast huvitatud ja robootika alal tegutsevad isikud ning panna nad proovile, võideldes oma robotitega üksteise vastu ja kasutades erinevaid ohtlikke relvi. Seda tüüpi võistlus nõuab robotite hoolikat planeerimist ja projekteerimist, et nad peaksid vastu areenil valitsevatele ohtudele ja suudaksid samal ajal teisi roboteid edukalt rünnata ja ennast kaitsta. Lisaks on robotvõistlusest saanud populaarne meelelahutusvorm ja suurepärane võimalus tutvustada tehnoloogia ja robootika viimaseid arenguid.

1.1 AHHAA Robolahingu areen

Nii publiku kui ka osalejate ohutus on iga robotivõistluse puhul ülimalt tähtis. Selle tagamiseks on paljudel võistlustel kehtestatud konkreetsed ohutusnõuded. Üks selline eeskiri puudutab areeni ehitust. Tavaliselt on areen valmistatud materjalidest, mis peavad vastu suurele löögile ja jõule. Näiteks robolahingu areen on valmistatud kahekihilisest kuulikindlast klaasist, mis kaitseb publikut ja piloote ohtlike robotite kildude või muude ohtude eest, mis võivad võistluse ajal õhku lennata. Lisaks areen on projekteeritud nii, et aku rikke korral ei leviks ohtlikke kemikaale või aurusid. Selliseid ohutusmeetmeid eelistades suudavad robotivõistlused pakkuda kõigile osalejatele põnevat ja köitvat elamust, tagades samas kõigi osalejate ohutuse Joonis 1.



Joonis 1. Robolahingu areen

Robolahing on võistlus, mis toimub spetsiaalselt ehitatud 8 korda 8 meetri suurusel ja 2,4 meetri kõrgusel areenil Joonis. 1. Sinna on paigutatud erinevad ohtlikud elemendid, mis tekitavad võistlejatele ja nende robotitele raskusi. Lisaks robotitele endile saavad osalejad juhtida ka kahte suurt kirvest, mis asuvad areeni vastaskülgedel ja mida saab kasutada vastaste täiendavaks kahjustamiseks. Areenil on ka kaks

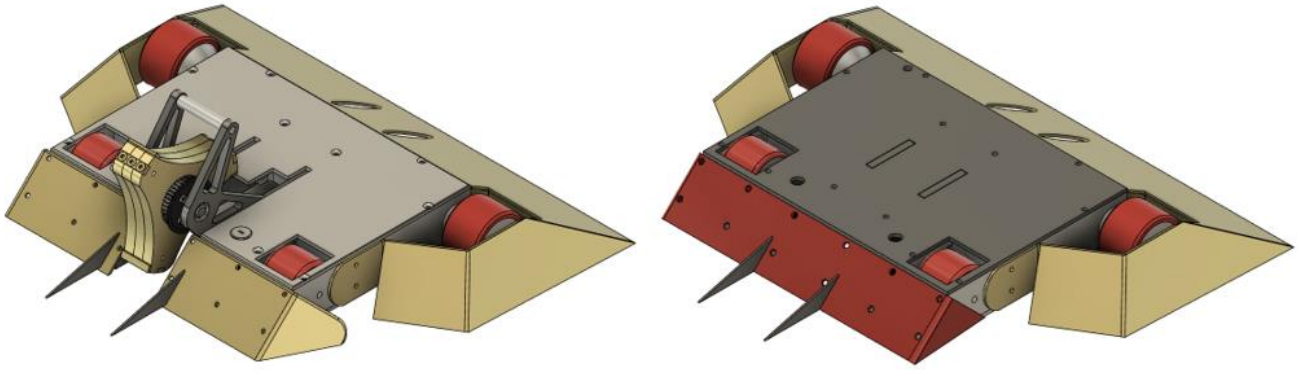
tule- ja kaks suitsuallikat, mis võivad oluliselt mõjutada robotite asukoha määramiseks kasutatava kaamera nähtavust. Need tegurid muudavad võistluse veelgi keerulisemaks ja põnevamaks nii võistlejate kui ka pealtvaatajate jaoks. Seega on võistluse oluline osa sellise roboti konstrueerimine, mis peab vastu areenil valitsevatele ohtudele ja on samas tõhus võitluses.

1 VÕISTLUS ROBOT

Roboti projekteerimine on keeruline ülesanne, mis nõuab hoolikat planeerimist ja tähelepanu detailidele. Antud juhul kasutati roboti projekteerimisel kõrgetasemelist Fusion 360 programmi, mis võimaldas õpilastel luua keerulised ja võimsad masinad. Aasta 2022. robot oli lähtepunktiks autonoomsuse saavutamiseks. Samuti töös oli kasutatud ka 2021. aasta robot mis on algversioon 2022. aasta roboist.

1.1 Archangel 2021

Roboti nimetus on Archangel. Robot on neljarattaline vertikaalne spinner, mille peamine kaitse on ettepoole paigutatud. Konstruktsioon on projekteeritud ja ehitatud autori poolt. Võistlusrobot omas kaks konfiguratsiooni, üks vertikaalse spinneriga ja eraldi paigutatud esikaitse plaatidega ning teine oli loodud varuvariandiks millel puudub aktiivne relv. Konfiguratsiooi kasutadamine oli mõeldud ainult rammimiseks ja juhul kui võistluse käigus esimese konfiguratsiooni parandamine kohapeal on võimatu.



Joonis 2. Archangel 2021 CAD keskonnas: vasakul – aktiivse relvaga konfiguratsioon; paremal – rammimis konfiguratsioon

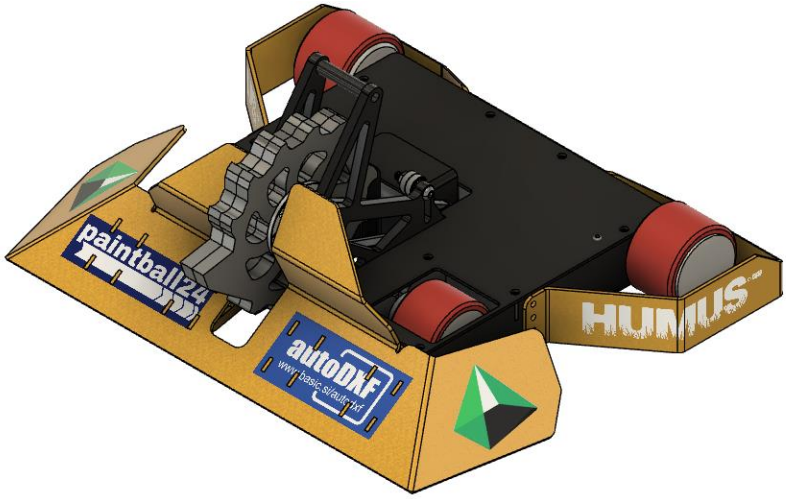
1.2 Archangel 2022

Uus versioon oli projekteeritud autori poolt ning ehitatud kooli robolahingu meeskonna abiga. Ehitati stabiilne robot, mis esines suhteliselt hästi 2022. aastal toimunud Robolahingu võistlusel. Robot paistis silma suurepärase hävitusjõu ning kiirusega ja suutis vastastele märkimisväärset kahju tekitada. Siiski esinesid võistluse ajal probleemid energiasüsteemiga, mis lõppkokkuvõttes viisid meeskonna väljalangemiseni. Vaatamata sellele tagasilöögile suudeti võistluselt koguda väärtuslikke kogemusi ja

teadmisi ning uut indu ja motivatsiooni, et konstruktsiooni ja juhitavust tulevaste võistluste jaoks täiustada.

Selle aasta 2022. roboti üks peamisi täiustusi oli hardox450 terase kasutamine kere ja esikaitse puhul, mis pakub suurepärast kaitset löökide ja muude ohtude vastu. Siseseinad, mis kannavad relva löökide koormust, on valmistatud S355 terasest, mis on vähem rabe ja pehmem kui hardox450 teras.

Kokkuvõttes kujutab selle roboti disain ja ehitus endast olulist sammu edasi robootikatehnika valdkonnas, näidates viimaseid edusamme materjaliteaduses ja mehaanilises disainis. Oma hirmuäratava relva ning kiiruse ja juhitavusega on see robot kindlasti suurepärane konkurent igal võistlusel.



Joonis 3. Archangel 2022 CAD keskonnas

2 ROBOTI JUHTIMINE

Robolahingu võistlusel on juhtimismehhanism oluline aspekt, mis võib määrata meeskonna edu. Tavaliselt hõlmab juhtimismehhanism ühte või mitut raadiosaatjat, mida piloot kasutab roboti erinevate osade juhtimiseks. Nende osade hulka võivad kuuluda muu hulgas relvad, ümberpööramismehhanism, roboti juhtimis- ja liikumissüsteem.

Keerulisemate robotite puhul võib juhtimine olla jagatud mitme meeskonnaliikme vahel, kus iga liige juhib roboti konkreetset osa ning kontsentreerub ainult sellele. Selline lähenemine tagab, et iga meeskonnaliige vastutab roboti konkreetse valdkonna eest, vähendades seeläbi segaduse ja vigade tõenäosust.

Võistlusel on võimalik juhtida ka mitut "multirobotit", kusjuures robotite arv ei ole võistlusreeglitega piiratud. Selline lähenemine võib anda meeskonnale märkimisväärse eelise, kuna see võimaldab robotitel töötada koos ühise eesmärgi saavutamiseks. Samas nõuab see ka kõrgetasemelist koordineerimist ja suhtlemist meeskonnaliikmete vahel, et tagada iga roboti tõhus tööülesande täitmine.

Üldiselt on juhtimine robolahingu võistluse kriitiline aspekt, sest kui robot ei liigu teda peatakse liigutuks ja ta kaotab. Robot võib põleda ja ilme relvata jääda aga kui ta liigub tal on veel võimalus võita.

2.1 Pulsilaiusmodulatsioon

Pulsilaiusmodulatsioon (PWM) *Pulse-width modulation* ehk impulsilaiusmodulatsioon ehk laius impulssmodulatsioon on modulatsiooni liik, milles väljundpinge reguleerimiseks muudetakse impulsside laiust. Lühend PWM tuleb ingliskeelsest terminist *Pulse Width Modulation*.

Kuigi pulsilaiusmodulatsiooni saab kasutada informatsiooni edastamiseks, on selle peamine kasutusala elektriseadmete võimsuse reguleerimine, eriti suure inertsiga koormiste, näiteks elektrimootorite puhul (Lazaridis, 2009).

Pildil kujutatud robotit Joonis 3 juhitakse kaugjuhtimispuldi abil, mille signaali võtab vastu roboti sees olev vastuvõtja. Et üleviia kontrolli antud töös käsitletava isejuhitava roboti üle, kasutati kaugjuhtimispuldi sees olevat treeneri funktsiooni, mis võimaldas võtta juhtimise üle ja saata oma signaali vastuvõtjale, ilma et keegi kaugjuhtimispulti füüsiliselt puudutaks. Selle saavutamiseks loodi

USB PPM-mooduli, mis ühendub sülearvutiga ja saab andmeid Pythoni skriptist *serial* kanali kaudu ning konverteerib neid PPM signaaliks. Kaugjuhtimispult koos mooduliga toimib sel juhul sillana arvuti ja roboti vahel, võimaldades võistluse ajal suuremat kontrolli ja täpsust roboti üle.

2.2 USB PPM-moodul

Moodul loodi arvutist saatjale andmete edastamise probleemi tõttu, kuna uuringu käigus ei leitud muid sobivaid lahendusi sellele probleemile, otsustati luua arvuti ja saatja vahendus, mis muundab signaali digitaalväärtusest analoogväärtuseks PPM-signaali kujul. Signaal saadetakse kahe juhtmega saatja 3,5 mm pistikusse, mis vastutab treeningrežiimi sisendi eest. Üks juhtmetest on gnd ja teine on signaalijuhe.

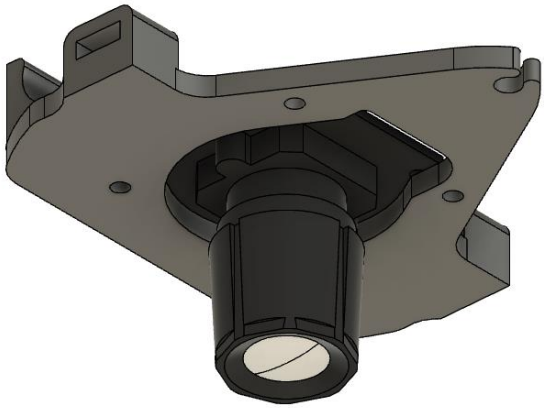
Treener Režiim on režiim, mis võimaldab õpilasel ühendada treeneri saatja ja juhtida mudelit, säilitades samal ajal treenerile võimaluse korrigeerida mudeli liikumist ja manööverdamist. See režiim on kõige kasulikum lennukimudelite, näiteks fikseeritud tiibade ja helikopterite puhul, kus tõenäosus mudelit juhtides kahjustada on kõige suurem.

USB PPM-moodul on oluline komponent roboti liikumises. Moodul koosneb arduino-mikrokontrollerist, mis on programmeeritud konkreetsete andmete vastuvõtmiseks *serial* kanalil, võimaldab signaali väärtuse edastamist vahemikus 1000-2000µs. Antud väärtusega juhitakse rattad. Iga kanali signaali impulss jääb vahemikku 1000µs kuni 2000µs, kus 1000 tähistab täielikku tagurdamist, 1500 tähistab seisvaid mootoreid ja 2000 tähistab täielikku edasiliikumist. Nende väärtustega manipuleerides saab juhtida roboti rattaid ja määrata selle liikumise suunda.

Joonis 4. USB PPM moodul saatja tagaküljel

2.3 Kaamera moodul

Antud uurimuses käsitletavas robotis kasutatav kaameramoodul on varustatud täiustatud funktsioonidega, mis muudavad selle autori eesmärkidele sobivaks.



Joonis 5. Kaamera kinnitus CAD keskkonnas

Selle projekti jaoks valiti usb-kaamera, mis suutis filmida kuni 60 kaadrit sekundis ning omas laia 120-kraadise vaatenurga, mis annab selge ja põhjaliku ülevaate areenist. Alla 60 kaadri sekundis ei sobinud, kuna nurga arvutamise algoritm ei suutnud järgida roboti tegelikku liikumist. Kaamera jaoks oli projekteeritud ning välja prinditud 3d printeri abil spetsiaalne korpus mis hoidis kaamera ning ka LED valgustus ribad, mis valgustasid reflektorite pinda. Valgustades, reflektorid peegeldavad valguse otse valgusallikasse mis soodustad selle tuvastuse kaameraga. LED-ide toiteallikaks on 12 V aku, mis koosneb kolmest 4,2 V li-ion akut, mis on ühendatud järjestikku. Taustavalgustuseks on kolm kokkujootetud lindi lõiku, mis kiirgavad tavalist valget valgust.

Joonis 6. Kaamera kinnitus koos LED ribadega

3 SIMULAATOR

Enne koodi kirjutamise alustamist otsustati luua lihtne simulaator, et testida roboti liikumist ja tuvastamist ohutult ja kiirelt. Pärast lühiajalist otsingut leiti lihtne mänguarenduskeskkond nimega *GDevelop*.

GDevelop on avatud lähtekoodiga, platvormideülene mänguarendustarkvara, mis võimaldab kasutajatel luua mänge, ilma et nad peaksid oskama koodi koostada. See pakub visuaalset kasutajaliidest mängude loomiseks, kasutades *drag-and-drop*-tegevusi ja sündmusi, mis teeb mänguarendusega alustamise algajatele lihtsaks. (gdevelop, n.d.)

GDevelop toetab 2D-mängude loomist ja sellel on palju funktsioone, sealhulgas füüsikamootor, mitme platvormi (*HTML5*, *Android* ja *iOS*) tugi ning võimalus eksportida mänge erinevatele platvormidele ilma lisavahendeid kasutamata.

GDevelop toetab ka laienduste kasutamist, millega saab tarkvarale lisada lisafunktsioone, näiteks integreeruda kolmandate osapoolte teenustega või lisada uusi mängumehhanisme.

Loodud simulaator võimaldab liigutada vastase robotit ja meie robotit eraldi ning arvutab ka kokkupõrkeid seintega ja robotite omavahelisi kokkupõrkeid, samuti arvestab hõõrdejõudu ja inertsi. Taust kujutab endast asuva Robolahingul põranda logo ja lisaks on lisatud ka tule emitatsioon, mis toimib roboti tuvastamise häirena videos, et testida algoritmide töökindlust

Saate näha, kuidas simulaator näeb välja ja töötab vaadates video Lisa 1 *Gdevelop simulator.*

4 KATSED

Käesoleva uurimisprojekti eesmärk oli uurida mitme tundmatu muutuja mõju konkreetse ülesande tulemusele. Ülesande keerukuse tõttu võisid tulemust mõjutada paljud tegurid, sealhulgas sisemised ja välised muutujad.

Nende muutujate ja nende võimaliku mõju paremaks mõistmiseks viidi läbi rida katseid. Need katsed olid mõeldud selleks, et uurida erinevaid lahendusi ja nende võimalikku mõju ülesande tulemusele.

Kogu testimisprotsessi jooksul puutus autor kokku arvukate probleemide ja takistustega, kuid analüüsides hoolikalt iga testi tulemusi ja kohandades oma lähenemisviisi vastavalt, suudeti lõpuks kindlaks teha mitu võtmetegurit, mis mõjutasid oluliselt ülesande tulemust.

Kokkuvõttes andis uurimisprojekt väärtuslikke teadmisi selle ülesande keerukusest ja sellest, kui oluline on soovitud tulemuse saavutamisel arvestada mitmete teguritega. Käesoleva uuringu tulemused avaldavad olulist mõju selle valdkonna tulevastele teadusuuringutele ja võivad aidata tulevikus suunata tõhusamate lahenduste väljatöötamist.

4.1 Platvormi valimine

Masinnägemise projekti jaoks sobiva platvormi valiku kaalumisel on oluline hinnata robotile esitatavaid erinõudeid, sealhulgas tuvastatavust, elemendi nähtavust, parandatavust ning võimalikke ohutegureid testimise ja võistluse ajal.

Platvorm on roboti liikuv alus, kuna robot on modulaarne, siis nimetame alust platvormiks.

Pärast erinevate võimaluste hoolikat kaalumist, sealhulgas uue platvormi ehitamist nullist, kitsendati valikut kahe olemasoleva platvormini: 2021. aasta robot ja 2022. aasta robot. Arvestades uue platvormi ehitamisega kaasnevaid olulisi kulusid ja tööjõudu, jäeti see variant lõpuks kõrvale.

Lõpliku otsuse tegemiseks hinnati iga platvormi eeliseid ja puudusi, võttes arvesse selliseid tegureid nagu mass, stabiilsus, remonditavus ja ohutus testimisel.

Masinnägemise projekti jaoks sobivaks platvormiks valis autor esialgu kergaalu kategooriast vertikaalse kettaga roboti, mida oli kasutatud eelneval aastal 2022 Joonis 3. Erinevate katsete läbiviimisel roboti

asukoha ja orientatsiooni määramiseks tuvastati mitu probleemi, mille lahendamine nõudis oluliselt ressursse.

Üheks peamiseks probleemiks oli suure massiga roboti relv, mis paiknes roboti kohal. See tõstis raskuskeset, vähendades stabiilsust pöörlemisel. 5kg relva pöörlemisel umbes 8000 pööret minutis, oli tulemuseks märgatav güroskoopiline efekt, mis tõstis roboti külje põrandast lahti, mis omakorda viis haardumise ja kontrolli kaotamiseni.

Teiseks valitud platvormi puuduseks oli relva kinnitamiseks kasutatava raami kõrgus, mis takistas teatud nurkade all vaadet tuvastus elemendile. Arvestades, et elemendi orientatsioon ja asend olid projekti edukuse seisukohalt kriitilise tähtsusega, kujutas see vähenenud nähtavus endast tõsist probleemi.

Kolmas võimalik probleem seisneb kogu süsteemi töökindluses – relva mitmeosalisus suurendab selle purunemise tõenäosust ja relva aktiivsus vähendab ohutust, eriti autonoomse süsteemi puhul.

Pärast põhjalikku kaalumist ja ulatuslikku testimist otsustas autor lõpuks kasutada oma masinnägemise projektis 2021. aasta platvormi. Kuigi see platvorm oli mitmes mõttes sarnane 2022. aasta platvormiga, oli selle mass väiksem, kuna peamise materjalina kasutati alumiiniumit. Samuti omas see laiemat raami mis suurendas paindlikkust tuvastatava kujundi valimisel.

Antud platvormi üks olulisemaid eeliseid oli, et see võimaldas kahte erinevat konfiguratsiooni, üks vertikaalse relvaga ja teine ilma. Teise konfiguratsiooni puhul asendati esikaitse massiivse plaadiga ja relva seinad langetati ülejäänud raamiga samale kõrgusele, mis tekitas elemendi tuvastamiseks piisavalt ruumi ja tagas sellele takistusteta vaadet. Arvestades projekti erinõudeid, osutus see konfiguratsioon sobivaimaks variandiks.

Teine oluline eelis 2021. aasta platvormi puhul oli selle pööratavus, mis võimaldas robotit juhtida ümberpööratud olekus. See funktsioon laiendab võimalike manöövrite valikut, mida robot saab sooritada.

Kokkuvõttes osutus otsus kasutada platvormi 2021. aasta targaks valikuks, sest see võimaldas arendada töökindlamat ja tõhusamat masinnägemissüsteemi, mis lahendas edukalt projektiga seotud eriprobleemid. Lisaks andis 2021. aasta platvormi ümberpööratavus autorile suurema paindlikkuse roboti juhtimiseks.

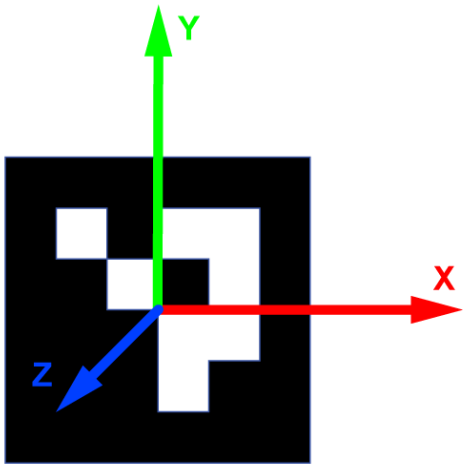
4.2 Tuvastuselement

Roboti jaoks sobiva elemendi valimine komplekses keskkonnas tegutsemiseks on keeruline ülesanne. Areeni keskkond on reeglina täis arvukaid segavaid tegureid, sealhulgas leeki, suitsu ja tolmu. Need tegurid võivad oluliselt takistada roboti võimet määrata täpselt oma asukohta ja orientatsiooni. Selle probleemi lahendamiseks viidi läbi rida katseid, et leida optimaalne element roboti asukoha määramiseks antud keskkonnatingimustes. Katsetes võeti arvesse erinevaid tegureid, mis tõenäoliselt mõjutavad roboti tööd, ning tulemusi analüüsiti hoolikalt, et määrata kindlaks parim element.

ArUco marker

ArUco markerid on binaarsed ruudukujulised koordinaatmarkerid, mida saab kasutada kaamera asendi hindamiseks. (ArUco marker detection (aruco module), kuupäev puudub)

Ideede mõtestamisel leiti võimalikke lahendusi roboti asukoha ja orientatsiooni määramiseks prooviti võimaliku lahendusena ArUco markeri abil (Joonis 8). See marker võimaldab anda väärtuslikku informatsiooni, näiteks kaamera ja markeri vahelist kaugust, selle orientatsiooni ruumis ja asukoha (X, Y, Z) positsiooni. Esmapilgul tundus ArUco marker ideaalse lahendusena probleemile, kuid lähemal vaatlusel selgus, et selliste markerite kasutamine on mõttekas vaid staatiliste ruumide puhul, kus järsk liikumine ja keskkonnamuutused on minimaalsed. Kui markerit liigutatakse liiga kiiresti, võib see põhjustada pildi ähmastumist, mis muudab selle määratlemise peaaegu võimatuks. Lisaks sellele võib marker kergelt kahjustada saada, mis tooks kaasa roboti positsiooni tuvastamise kaotuse, millega omakorda kaasnevad ebasoovitavad tagajärjed. Arvestades neid piiranguid, jõudis autor järeldusele, et tuleb leida alternatiivne lahendus.



Joonis 7. Aruco marker

Kleeplint

Teine võimalik lahendus, mida uuriti, oli heleda ja hästi nähtava kleeplindi kasutamine. Katsed näitasid siiski, et kleeplindi nähtavus sõltub suuresti valgustus tingimustest. Võistlus Keskkonnas võib valgustus varieeruda väga heledast kuni täiesti pimedani, mis muudab kleeplindi kasutamise ebapraktiliseks. Seetõttu jäeti ka see variant kõrvale, kuna see ei sobinud antud ülesande täitmiseks.

Kleepsud

Katsetati ka erksavärvilisi kleebiseid, kuid nende väikese suuruse tõttu ei suutnud kaamera neid ümbritsevast keskkonnast eristada. Kuna kleebised ei peegelda valgust, ei ole neist hämaras keskkonnas kasu. Nende tulemuste põhjal jõuti järeldusele, et kleebiste kasutamine ei ole otstarbekas.

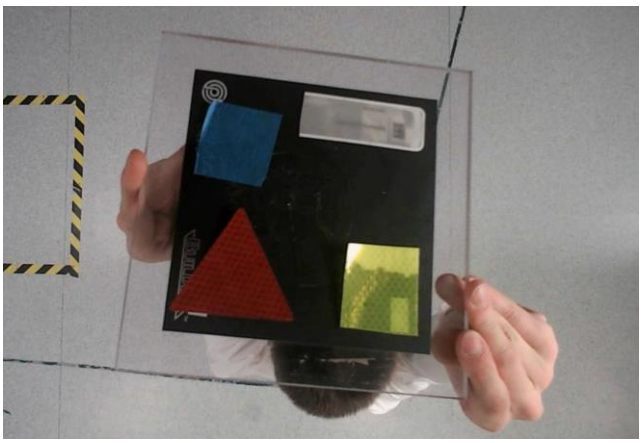
Reflektor

Pärast erinevate võimaluste kaalumist otsustas autor lõpuks kasutada reflektorit, et lahendada roboti asukoha ja orientatsiooni määramise probleem. Reflektorite eeliseks on võime peegeldada valgust otse tagasi selle allikasse, mis teeb neist ideaalse lahenduse antud probleemi puhul. Siiski oli vaja luua sobiv valgusallikas, et tagada helkuri nõuetekohane tuvastamine. Töötati välja spetsiaalse valgusallikas, mis suutis valgustada reflektorit, mille järel oli lihtne kujundit taustast eristada ning arvutada positsiooni ja orientatsiooni. Selle tulemusena osutus helkuri kasutamine praktiliseks ja tõhusaks lahenduseks robotite asukoha ja orientatsiooni määramise jaoks komplekses keskkonnas.

4.3 Tuvastuselementide nähtavuse katsed

Roboti jaoks sobivaima tuvastus elemendi kindlaksmääramiseks viidi erinevate elementidega läbi rida katseid. Katsetati nelja erinevat elementi (Joonis 1), sealhulgas sinine kleeplint, jalakäijate helkur, jalgratta helkur ja liiklusmärkide reflektor. Elementide kontrasti saavutamiseks asetati need mustale plaadile ja erinevate valgustingimuste puhul kasutati lakke paigaldatud kaamerat, et salvestada pilte. Katsed olid järgmised: ruumi valgustus sisse lülitatud ja kaamera taustavalgus sisse lülitatud, ruumi valgustus sisse lülitatud ja ilma kaamera valgustuseta, pimedas, kui kaamera valgus oli sisse lülitatud, ja pimedas ilma kaamera valgustuseta.

Katse eesmärk oli tuvastada kõige sobivam element, mille abil saaks luua kujutise, mida algoritm suudaks ära tunda ja mille abil saaks roboti asukoha ja orientatsiooni täpselt välja arvutada. Autor viis katsed läbi erinevates valgustus tingimustes, et simuleerida keskkonda, milles robot töötaks võistlusel. Autor oli huvitatud iga elemendi jõudluse hindamisest erinevates valgustus tingimustes, et teha kindlaks, milline element sobib antud probleemi lahendamiseks kõige paremini.



Joonis 8. Tuvastatavate ementide katsekeha: sinine – kleeplint; valge – jalakäijate helkur; punane – jalgratta helkur; helekollane – liiklusmärkide reflektor

Katseid alustati kleeplindi sobivuse hindamisega. Kuigi kleeplint paistis valgustatud ruumis hästi silma, Joonis 9: a ja c, ei paistnud see pimedas üldse välja Joonis 9: b, d. Seetõttu ei peetud seda sobivaks võimaluseks. Seejärel katsetati jalakäijate helkurit ja jalgratta helkurid, mis toimisid ootuspäraselt suhteliselt hästi. Kõige enam paistis aga silma liiklusmärkide reflektor, mis tänu oma heledale värvile ja peegeldusvõimele oli hästi nähtav ka hämarates tingimustes ilma kaameravalguseta.

Joonis 9. Tuvastatavate ementide katsed:a – toavalgusega ja kaamera valgustiga; b – pimedas ja kaamera valgustiga; c – toavalgusega ilma kaamera valgustita; d – pimedas ilma kaamera valgustita

Kokkuvõttes näitasid katsed, et helkurid toimisid paremini kui kleeplint. Helkurid olid kergesti tuvastatavad ja andsid usaldusväärsemaid andmeid. Kõige rohkem paistis silma liiklusmärkide reflektor, mis tänu oma heledale värvile ja peegeldusvõimele oli hästi nähtav kõikides valgustingimustes. Seetõttu jõudis autor järeldusele, et liiklusmärkide helkur on antud ülesande jaoks kõige sobivam variant.

4.4 Tuvastatav kujund

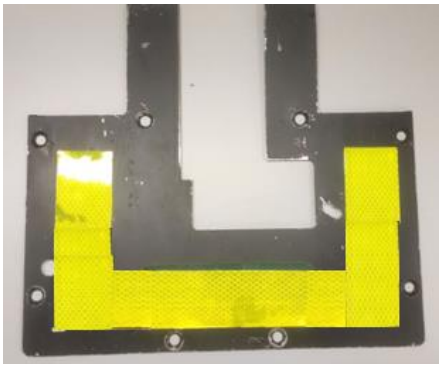
Autor viis läbi uurimuse, et selgitada välja erinevad meetodid liikuvate objektide tuvastamiseks piltidel. Selle eesmärgi saavutamiseks katsetas autor erinevaid algoritme pildi elementide tuvastamiseks, näiteks Template Matching ja värvimaskid. Lisaks tehti katseid masinõppe mudeli välja töötamiseks, et määrata kindlaks selle tõhusus liikuvate objektide tuvastamisel. Käesoleva uuringu eesmärk oli saada põhjalik ülevaade erinevatest lähenemisviisidest, mida saab kasutada liikumise tuvastamiseks ja jälgimiseks. Nende meetodite hindamise kaudu suutis autor kindlaks määrata kõige sobivama lähenemisviisi antud keskkonnas roboti tuvastamise jaoks.

Järgmised kaks katset olid läbitud roboti 2022 platvormil. Kuigi platvormi peal asuvatest kujunditest pilte ei salvestunud oli tehtud korduvad pildid ilma platvormita. Kuna plaadi peal on vähe ruumi pidi kontuur olema paigutatud välisäärel. Samuti segasid relva toeseinad vaadet mõnede nurkade all.

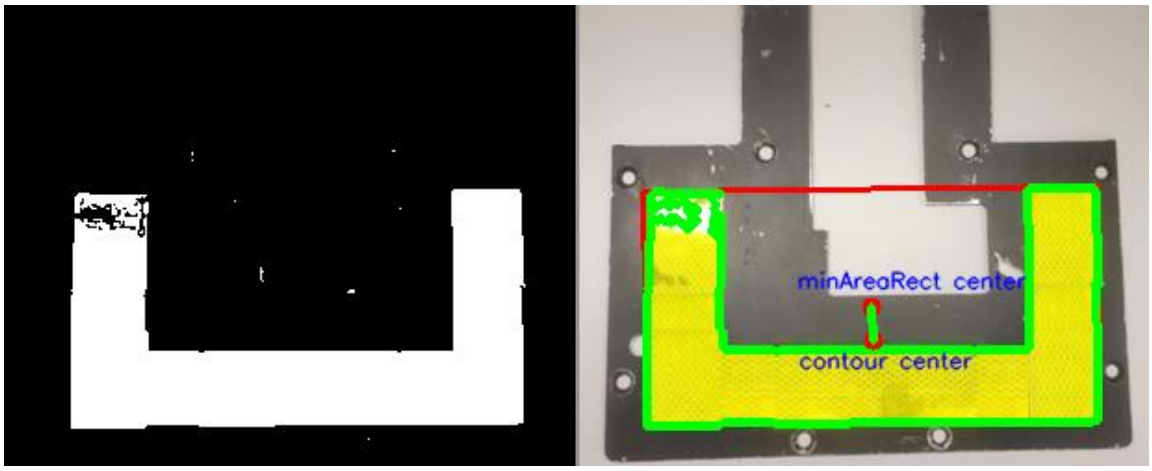
U-kujund

Esimese katsena otsustati kasutada liiklusmärkide reflektoreid roboti katuse plaadi peal, kleepides neid selliselt, et sellest tuleks välja kujund mille massikese oleks nihutatud kogu kujundi pikkuse suhtes,

Joonis 10. Samuti peab kujund olema sellise kujuga, mis ei sarnaneks mitte millegagi, mis võiks areenil potentsiaalselt tekkida, misiganes vaenlase roboti või valguspeegelduse kujul. Lisaks peab olema konkreetse suunaga, et nurka kahe punkti vahel kätte saada, näiteks ruut ja ring ei sobiks. Järgmisena kasutati masinnägemise algoritmi, et eraldada fotolt värvimask ning pärast suletud kontuuride leidmist arvutati kontuuri massi keskpunkt, mis andis esimese punkti kui roboti asukoha pildi suhtes. Seejärel arvutati kontuuri ümbritseva ristküliku minimaalne pindala, mis andis teise punkti. Viimaks joonistati nende kahe punkti vaheline joon, mis andis roboti suuna pildi suhtes.



Joonis 10. U-kujulise märgis



Joonis 11. värvi maski järgi u-kujulise kujundi tuvastus

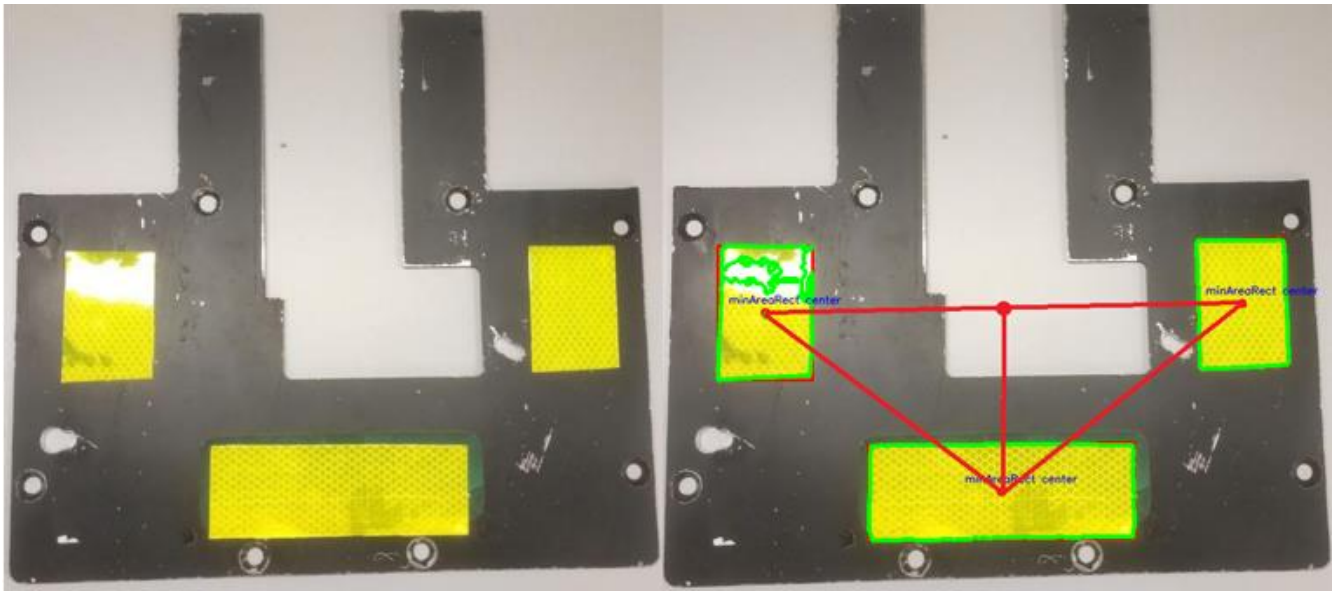
Katsete tulemused näitasid, et see meetod on kaamera poolt hästi tuvastatav ja orientatsiooni ning asukoha arvutamine pole samuti halb, kuid sellel kujul on suured puudused, millest üks on väga väike kaugus kontuuri keskpunktist kuni ristküliku minimaalse alani, mis muutis iga valguse kõikumise piisavaks mõjuteguriks nurga kõikumiseks +-20 kraadi piires, mis muudab täpse roboti juhtimise võimatuks vea signaali tugeva kõikumise tõttu.

Samuti pidas algoritm üsna sageli valguspeegeldusi ja võnkumisi roboti kontuuriks, kuna algoritm otsis antud juhul suurimat kontuuri konkreetses värvimaskis.

Kolm eraldi ristkülikut

Siinkorral otsustati eelmine kontuur jagada kolmeks eraldi ristkülikuks ja arvutada iga ristküliku kontuuride keskpunktid, et edaspidi märkida nende vahelised jooned ja luua pika alusega kolmnurk ning võtta pool aluse pikkusest, et saada suunapunkt. Seejärel oli võimalik alumise ristküliku vahelise joone tõmbamine kuni poole aluse pikkusega punktini, et saada kätte roboti suund.

Teoreetiliselt on selline paigutus efektiivsem, sest suunapunktid on üksteisest kaugemal, mistõttu nurga arvutamine on vähem vastuvõtlik kontuuri muutustele, mis tulenevad teatud nurkade puhul valgusefektidest, ning isegi kui üks ristkülikutest on kahjustatud, ei nihkuks kontuuri massi keskpunkt märkimisväärselt, ega mõjutaks arvutatava nurga täpsust.



Joonis 12. Kuju kolme ristkülikuga

Testi tulemus näitas samuti edukat nurga tuvastamist. Tulemused olid täpsemad ja nurga kõikumine vähenes +-4 kraadini, mis oli ootuspärane, kuna määratud punktide vaheline kaugus on suurem.

Kõikide kontuuride arvutamise, kolmnurga leidmise ja järgnevate arvutuste algoritm nõudis aga tohutult ressursse, mis vähendas ühe tsükli kiirust 300 ms-ni, mis on omakorda ülesande jaoks vastuvõetamatu.

Algoritm otsis kindla väärtusega kontuure, seejärel joonistas, arvutas kauguse lähima kontuurini ning kui see oli teatud kaugusel, siis tõmmati joon kõigi punktide vahel. Probleemiks osutus ka väike põrandalt peegeldunud valguse peegeldus, mis viis roboti jälgimise kadumiseni, mis on samuti lubamatu.

Kolmnurga kujuline kujund

Järgmine otsus tuli pärast platvormi asendamist 2021. aasta platvormiga, millel oli rohkem ruumi, et mahutada tuvastatav element, kuid millel puudus aktiivne relv, millest tuli loobuda, sest seda oleks olnud väga raske hallata ja katsetada ülikooli tingimustes, kuna meil puudus turvaline areen, mis kaitseks vigastuste ja hävitamise eest. Samuti otsustasime autonoomse roboti esimese versiooni puhul kasutada kõige lihtsamat lahendust, et teha kindlaks kontseptsiooni elujõulisus.

Katse jaoks kleebiti liiklusmärgi reflektor kolmnurga kujuliselt roboti ülemisele plaadile, ülejäänud osa sellest oli kaetud paberist kleeplindiga, et vältida alumiiniumpinna peegeldumist ja luua kontrasti tuvastatava objektiga. Kolmnurga valimist põhjendati sellega, et see kuju on areeni tingimustes üsna haruldane, ka varjud ja peegeldused ei võta tõenäoliselt sellist kuju.

Joonis 13. kolmnurkse kuju töötlemine: vasakpoolne – enne töötlust; keskel – värvimask; parempoolne – lõpptulemus

Katse tulemus oli, et kolmnurkne kuju oli märkimisväärselt parem kui eelmised, isegi mitte-ideaalses keskkonnas, kus värvimaskil esines müra. See kujund võimaldas ka nurga täpsust veelgi suurendada, sest algoritm võtab nurga määramiseks kolmnurga väiksema aluse ja kolmnurga tipu punkti. Lahendus vähendas nurga kõikumist +-0,5 kraadini, mis oli praktiliselt ideaalne nii roboti asukoha ja orientatsiooni arvutamiseks kui ka navigeerimiseks.

Oluline on ka see, et algoritmi tsükkel nurga arvutamiseks võttis aega 30 ms, mis suurendas oluliselt nurgavea tuvastamise ja arvutamise kiirust reaalajas, ilma et arvutuste tegemiseks oleks vaja võimsat riistvara.

Otsustati jätkata selle lahendusega, sest see tundus olevat eelmiste lahendustega võrreldes parim nii arvutuste kiiruse kui ka vigade tuvastamise ja häirivate tegurite osas, mis mõjutasid kolmnurkse kujundi äratundmist kõige vähem.

5 PROGRAMM

Programmikood on roboti aju, kus toimuvad kõik kiiruse ja nurkade arvutused, pilditöötlus, kontuuride tuvastamine ning andmete vastuvõtmine ja saatmine.

Programmeerimiskood kirjutati python programmeerimiskeeles *Pycharm* programmeerimise keskkonnas, kuna see on kasutajasõbralik ning lihtne ja kiire prototüüpimise võimalusega. Peamiseks raamatukoguks pildi tuvastamiseks oli kasutatud *OpenCV* (*Open Source Computer Vision Library*). Programmi kirjutamine osutus üsna mahukaks ja keeruliseks, sisaldades üle 30 funktsiooni. Suurem osa lõputöö teostamise ajast kulus programmi kirjutamisele ja vigade kõrvaldamisele ning optimiseerimisele.

OpenCV on avatud lähtekoodiga masinnägemise ja masinõppe tarkvara raamatukogu. *OpenCV* loodi selleks, et pakkuda ühist infrastruktuuri masinnägemise rakenduste jaoks ja kiirendada masinataju kasutamist kaubanduslikes toodetes. (About, n.d.)

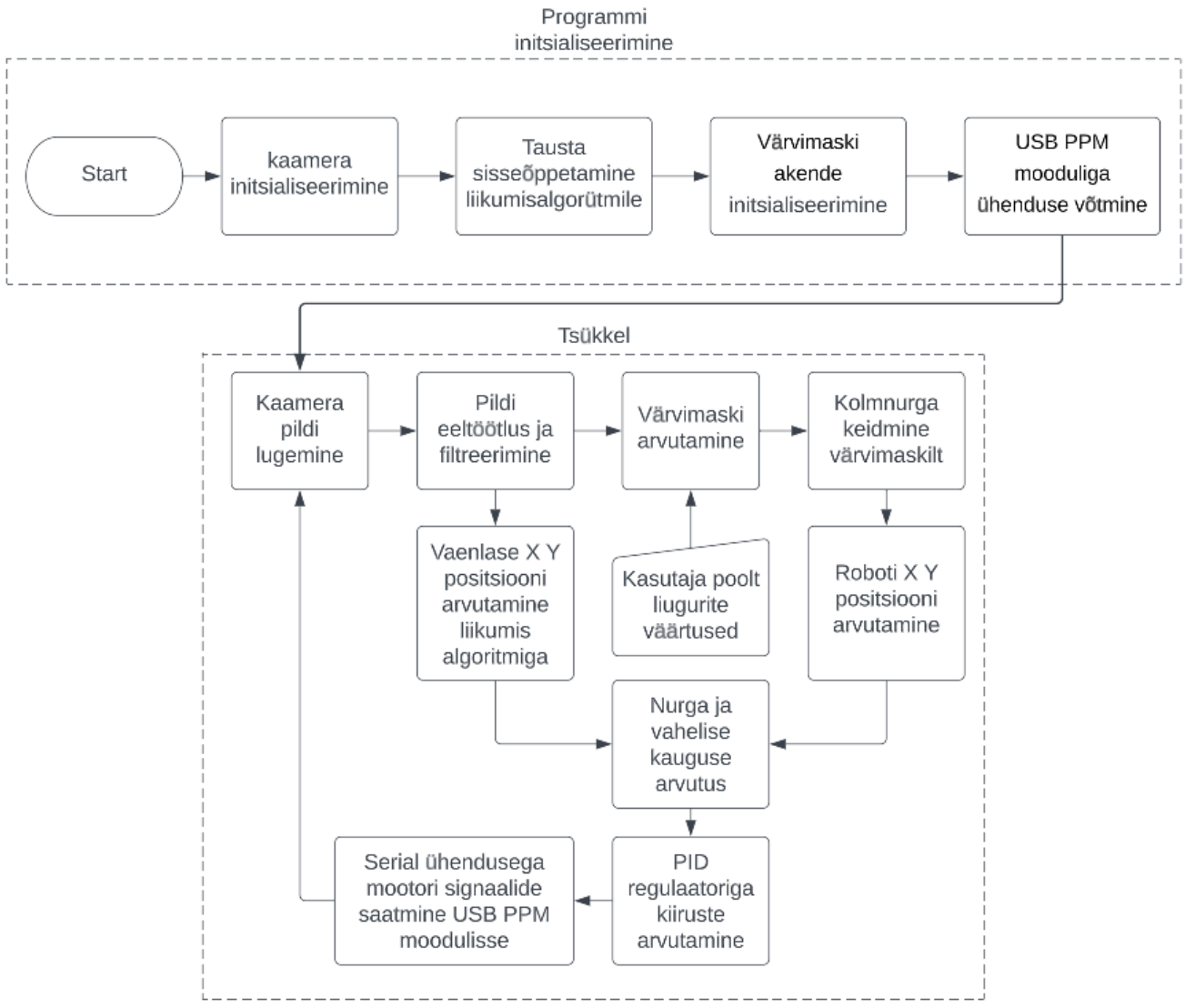
Raamatukogus on rohkem kui 2500 optimeeritud algoritmi, mis sisaldab põhjalikku komplekti nii klassikalistest kui ka moodsatest masinnägemise ja masinõppe algoritmidest. Neid algoritme saab kasutada nägude tuvastamiseks ja äratundmiseks, objektide tuvastamiseks, inimeste tegevuse klassifitseerimiseks videos, kaamera liikumise jälgimiseks, liikuvate objektide jälgimiseks, objektide 3D-mudelite väljavõtmiseks, stereokaameratest 3D-punktipilvede loomiseks, piltide kokkuühendamiseks, et saada kogu stseeni kõrgresolutsiooniga pilt, sarnaste piltide leidmiseks pildid andmebaasist, punaste silmade eemaldamiseks välguga tehtud piltidelt, silmade liikumise jälgimiseks, maastiku äratundmiseks ja markerite loomiseks, et seda täiendatud reaalsusega üle kanda jne. *OpenCV*-l on üle 47 tuhande inimesega kasutajaskond ja hinnanguline allalaadimiste arv ületab 18 miljonit. Raamatukogu kasutatakse laialdaselt ettevõtetes, uurimisrühmades ja valitsusasutustes. (About, n.d.)

Kood peab olema täitmisel kiire, sest see määrab roboti reaktsioonikiiruse, koodi peab olema lihtne ja mugav lugeda, sest selle soovituse mittetäitmine viib kiiresti olukorrani, kus programmeerija ei saa aru, mis toimub ja mis millest sõltub, ning kood peab olema piisavalt paindlik, et seda saaks muuta, mistõttu saab seda projekti veelgi täiustada ja arendada, lisades uusi funktsioone ja seeläbi laiendades roboti ja kogu süsteemi kui terviku võimekust.

Mõistlik on jagada suur programm väiksemateks mooduliteks, mis täidavad konkreetseid ülesandeid, ja seejärel kombineerida need kokku, et kogu programm toimiks, nii on lihtsam vigu leida, sest ühe mooduli testimine on palju lihtsam kui kogu programmi testimine.

Programm jagati kaheks failiks: põhifail "`main.py`", kuhu kõik funktsioonid kutsutakse, ja fail "`utils.py`", kus need funktsioonid asuvad ja kust põhifail neid välja kutsub.

Lõplik programm sisaldab enam kui 30 funktsiooni, mis vastutavad programmi erinevate aspektide eest, näiteks: moodulite vaheline suhtlus, punktidevahelise kauguse arvutamine, piltide filtreerimine, kolmnurga leidmine jne.



Joonis 14. Lõpliku programmi lihtsustatud plokskeem

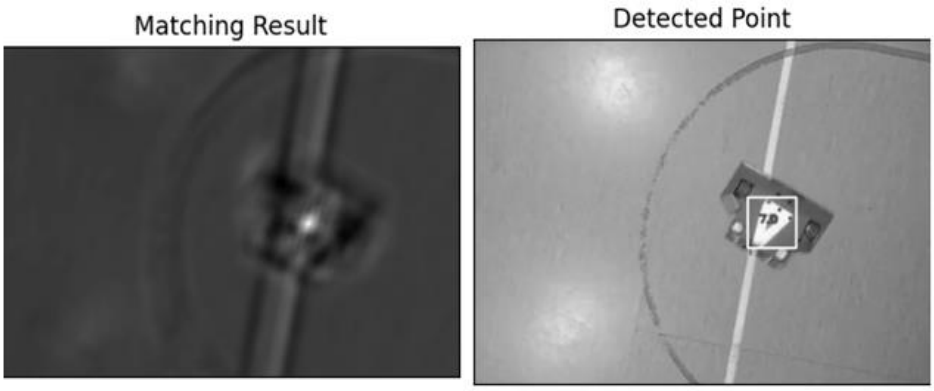
5.1 Tuvastus

Efektiivse ja õigeaegse roboti juhtimise poole püüdlemisel tekkis oluline väljakutse: vajadus kiire ja usaldusväärse tuvastusalgoritmi järele, roboti asukoha ja orientatsiooni täpseks määramiseks. Selle nõude täitmiseks viidi läbi põhjalik uuring, et leida kõige optimaalsem lähenemisviis.

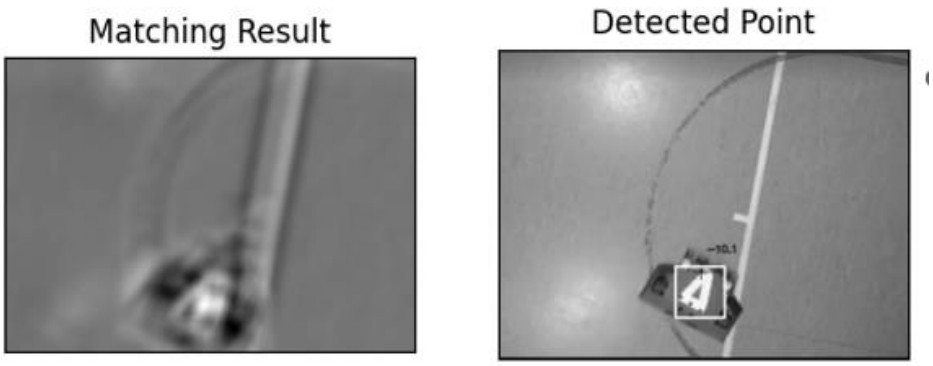
Uuringu tulemused näitasid, et on vaja algoritmi, mis võimaldaks töödelda ühte kaamera kaadrit maksimaalselt 200 ms jooksul. Mis tahes viivis, mis ületab seda piiri, tekitaks roboti juhtimises probleeme, takistaks kursikorrektsiooniliigutusi, saates liialt aeglaselt veasignaali muutusi ja aeglustaks üldist reaktsioonikiirust. See tähelepanek leidis kinnitust põhjalike katsete käigus, mis rõhutasid roboti juhtimissüsteemi selle aspekti tähtsust.

5.1.1 Template matching ehk šablooni sobitamine

Template Matching on meetod šablooni asukoha otsimiseks ja leidmiseks suuremas pildis. *OpenCV*-l on selleks funktsioon `cv.matchTemplate()`. See lihtsalt libistab šablooni üle sisendi pildi ja võrdleb šablooni ja sisendi pilti. (docs.opencv.org, kuupäev puudub)



Joonis 15. šablooni sobitamine 1



Joonis 16. šablooni sobitamine 2

Käesolevas olukorras analüüsime šablooni tuvastamist kahel fotol, Joonis 15 ja Joonis 16. Esimesel fotol Joonis 15: *matching result* – on selge muster, mida tõendab valge värvi suur kontsentratsioon vastavatel aladel. See näitab, et šabloon Joonis 17 oli fotol tõepoolest olemas ja meie mustrituvastuse meetod tuvastas selle edukalt. See oli oodatav, sest šabloon on võetud just sellelt fotolt. Joonis 16: *detected point* – joonestas algoritm ruudu ümber tuvastatava punkti märkides šablooni piirid.

Joonis 17. šabloon

Seevastu teine foto kujutab endast mustri tuvastamisel keerulisemat juhtumit. Kuigi sellel ei ole selgeid valgeid alasid, mis vastaksid mustrile, suutis algoritm selle olemasolu siiski tuvastada. See on oluline, sest mustri tuvastamise algoritm ei tööta tavaliselt hästi orientatsiooni muutnud piltidel, mis teeb selle leiu veelgi paremaks. Lisaks märgime, et fotol olevad valged alad vastavad täpselt mustri asukohale.

Paraku selgus pärast šablooni sobitamise algoritmi testimist videosalvestusel, et selle meetodi kasutamine tuvastamiseks on võimatu, tuvastamisele kuluva liigselt pika aja tõttu, mis ületas eelnevalt seatud 200 ms piiri. Oli vaja leida tuvastamiseks teine, kiirem meetod.

5.1.2 Värvi järgi eraldamine

Pildil värvide eraldamiseks oli tehtud pildi filtreerimine, seejärel kasutati värvide tuvastamiseks kasutaja poolt määratud värvide vahemikku, mis veidi ületas tuvastatava objekti värvi väärtused, st helkuri värvi. Seejärel loodi töödeldud pildist värvimask määratud vahemikus, mis välistas pildilt kõik, mis ei kuulunud antud vahemikku. Näidet võib näha Joonis 11 vasakul pildil.

Pildimask on binaarne pilt, mida kasutatakse teise pildi välimuse valikuliseks muutmiseks või manipuleerimiseks. Mask koosneb pikslitest, mis on kas valged või mustad, kusjuures valged pikslid tähistavad pildi piirkondi, mida tuleks muuta, ja mustad pikslid tähistavad piirkondi, mis peaksid jääma muutmata. (Rosebrock, 2021)

Pildimaskid on sageli kasutusel pilditöötluse ja masinnägemise rakendustes, et teha selliseid toiminguid nagu pildi segmenteerimine, filtreerimine ja maskeerimine. Näiteks saab maski kasutada pildi konkreetse huvipakkuva piirkonna eraldamiseks, määrates kõik pikslid väljaspool seda piirkonda mustaks ja kõik pikslid selle piirkonna sees valgeks. Saadud maski saab seejärel kasutada mitmesuguste pilditöötluse operatsioonide rakendamiseks ainult huvipakkuvale piirkonnale, jättes ülejäänud pildi muutmata. (Rosebrock, 2021)

Et määratleda fotot värvide järgi, tuleb kõigepealt määrata fotol määratletav värvivahemik. selleks on kirjutatud klass "*ImageProcessor*" (Joonis 18), mis loob fototöötluse objekti ja loob kaks eraldi akent liuguritega maski reguleerimiseks ja pildi maski eelvaate avamiseks. Klass "*ImageProcessor*" sisaldab funktsiooni "*process_image*", mis võtab töödeldava foto, filtreerib seejärel pildi mürast ja rakendab hägususfiltrit.

Pärast saadetakse töödeldud foto funktsioonile "*create_mask*", mis konverteerib foto BRG (*Blue Red Green*)-ruumist HSV (*Hue saturation value*)-ruumi ja väljastab pildimaski selle ühes loodud aknas.

Tulemusena saame kätte meile vajaliku mürast puhastatud värvi vahemiku binaarse pildi kujul, mis on järgnevaks töötluseks ette valmistatud.

```
class ImageProcessor:
    def __init__(self, window_name):
        self.window_name = window_name
        self.lower = np.array([16, 0, 238])
        self.upper = np.array([255, 255, 255])
        self.image = None
        self.mask = None
        self.hsv = None
        self.create_trackbars()

    def process_image(self, image):
        kernel = np.ones((5, 5), np.float32) / 25
        dst = cv2.filter2D(image, -1, kernel)
        blur = cv2.blur(dst, (6, 6))
        self.image = cv2.medianBlur(blur, 5)
        self.create_mask()
        return self.image

    def create_trackbars(self):
        cv2.namedWindow(self.window_name)
                cv2.createTrackbar('Lower R', self.window_name, self.lower[0], 255, self.update_trackbars)
                cv2.createTrackbar('Lower G', self.window_name, self.lower[1], 255, self.update_trackbars)
                cv2.createTrackbar('Lower B', self.window_name, self.lower[2], 255, self.update_trackbars)
                cv2.createTrackbar('Upper R', self.window_name, self.upper[0], 255,  self.update_trackbars)
                cv2.createTrackbar('Upper G', self.window_name, self.upper[1], 255,    self.update_trackbars)
                cv2.createTrackbar('Upper B', self.window_name, self.upper[2], 255, self.update_trackbars)

    def update_trackbars(self, val):
        self.lower[0] = cv2.getTrackbarPos('Lower R', self.window_name)
        self.lower[1] = cv2.getTrackbarPos('Lower G', self.window_name)
        self.lower[2] = cv2.getTrackbarPos('Lower B', self.window_name)
        self.upper[0] = cv2.getTrackbarPos('Upper R', self.window_name)
        self.upper[1] = cv2.getTrackbarPos('Upper G', self.window_name)
        self.upper[2] = cv2.getTrackbarPos('Upper B', self.window_name)
        self.create_mask()

    def create_mask(self):
        self.hsv = cv2.cvtColor(self.image, cv2.COLOR_BGR2HSV)
        self.mask = cv2.inRange(self.hsv, self.lower, self.upper)
        return self.mask

    def show_image(self):
        cv2.imshow(self.window_name, self.mask)
        return self.mask
```

Joonis 18. Pilditöötlus kood

5.1.3 Kujundi tuvastamine

Kujundi leidmiseks kasutame eelnevalt töödeldud binaarset pilti. Kujundi tuvastamine pildilt on keerukas protsess mis nõuab kvaliteetselt töödeldud sisendit - sellest sõltub kõikide ülejäänud protsesside täpsus.

Binaarsest pildist kujundi tuvastamiseks oli kasutatud kontuuri määratlemist ümber kinniste objektide, seejärel kontuuri punktide sirgendamist, mis vähendab punktide arvu ning muudab kontuurid nurgelisemaks. Seejärel filtreeritakse kõikidest kontuuridest välja need, mille pindala on suurem kui määratud väärtus ja mille nurkade arv võrdub kolmega, mis tähendab kolmnurka. Kui kujund on leitud, leitakse kolmnurga alus arvutades kõigi kolme punkti omavaheline kaugus. Lühim kaugus ongi kolmnurga alus, mille pikkus jagatakse pooleks, et saada lõigu keskpunkt. Pärast lõigu keskpunkti leidmist arvutatakse kaugeim punkt, mis on kolmnurga tipp ja joonestatakse joon lõigu keskpunktist kolmnurga tipuni. Nimetame seda suuna vektoriks Joonis 13 parempoolne pilt.

```
def find_triangle(image, target_image):
    contours, hierarchy = cv2.findContours(image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
    # Initialize an empty array to store the coordinates
    point_array = []
    if len(contours) == 0:
        # No contours found, return None for all values
        return target_image, None, None
    else:
        for cnt in contours:
            area = cv2.contourArea(cnt)
            if 500 <= area <= 5000:
                peri = cv2.arcLength(cnt, True)
                approx = cv2.approxPolyDP(cnt, 0.07 * peri, True)
                objCor = len(approx)

                if objCor == 3:
                    for point in approx:
                        x, y = point[0]
                        point_array.append([x, y])
                        cv2.circle(target_image, point[0], 3, (0, 0, 255), -1)
                        #
                        # cv2.putText(target_image,str(point[0]), (point[0][0] - 20, point[0][1] - 20),
cv2.FONT_HERSHEY_SIMPLEX,
                        #           0.5, (255, 0, 0), 2, cv2.LINE_AA)

        if len(point_array) == 3:
            botom_point = find_closest_point(point_array)
            top_point = find_furthest_point(point_array, botom_point)

            cv2.circle(target_image, botom_point, 3, (0, 0, 255), -1)
            cv2.circle(target_image, top_point, 5, (255, 0, 0), -1)

            cv2.line(target_image, botom_point, top_point, (0, 0, 255), 2)
            cv2.putText(target_image, "Botom point", botom_point, cv2.FONT_HERSHEY_SIMPLEX,
                        0.5, (255, 0, 0), 1, cv2.LINE_AA)

            cv2.putText(target_image, "Top point", top_point, cv2.FONT_HERSHEY_SIMPLEX,
                        0.5, (255, 0, 0), 1, cv2.LINE_AA)
            return botom_point, top_point
        else:
            return None, None
```

Joonis 19. Kolmnurga leidmise kood

5.2 Vaenlase leidmine

Vastase leidmiseks oli algselt kasutatud sarnast lähenemist nagu värvimaski abil tuvastamisel, kuid see meetod on efektiivne vaid sel juhul, kui on olemas mingi eriline objekt, mida tuleb tuvastada Paraku ei saa eeldada, et vastased lubavad oma roboteid värvi või mingit tuvastatava elemendi abil märgistada, seega otsustati kasutada liikumistuvastuse meetodit fotol. Sellel meetodil on mitmeid puudusi, mis võivad takistada tuvastamist või anda valesid andmeid, sest areenil on peale vastase ka meie robot, mis samuti liigub ja mida tarkvara võib tuvastada, ajades juhtimissüsteemi segadusse. Selle probleemi ületamiseks otsustati tuvastada liikumine teatud järjekorras, kõigepealt määratakse kindlaks, kas meie robot on pildil tuvastatud, seejärel, kui see on tuvastatud, luuakse roboti ümber ala, kus vastase liikumist ei arvestata ja alles sellest alast väljas fikseeritakse liikumine ja arvutatakse juhtimissignaal sihtmärgi suunas liikumiseks. See lähenemisviis tuvastab liikumist videos küllaltki täpselt ja tuvastatava objekti suurust on samuti võimalik muuta.

Vaenlase koordinaatide saamiseks videovoo suhtes kasutasime klassi `MovingObjectDetector`, (Joonis 20), mis kasutab *OpenCV* raamatukogu sisseehitatud funktsiooni nimega `createBackgroundSubtractorMOG2`, mis analüüsib sissetulevat fotot mõnda aega ja mäletab sellel olevaid alasid kindlaksmääratud täpsusega, seejärel võrdleb salvestatud kaadrit sissetuleva kaadriga ja võimaldab arvutada nende erinevust, mille saab tõlgendada binaarseks pildiks ja leida sellel kontuurid, seejärel filtreerida suuruse järgi ja välja arvutada kontuuri massikeskme, mis annab meile liikuva objekti keskpunkti koordinaadid.

```
#   detects moving object (X, Y) and take point to not detect around it
class MovingObjectDetector:
    def __init__(self, learning_rate=0.01):
        self.subtractor = cv2.createBackgroundSubtractorMOG2(detectShadows=False)
        self.learning_rate = learning_rate

    def detect(self, image, xy_point=None, rect_scale=0):
        # Apply background subtraction to obtain the moving objects
        fgmask = self.subtractor.apply(image, learningRate=self.learning_rate)

        # xy_point_rect = (int(xy_point[0]-rect_scale), int(xy_point[1]-rect_scale)), (int(xy_point[0])+rect_scale,
int(xy_point[1])+rect_scale)

        # Apply morphology operations to remove noise and fill gaps in the moving objects
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
        fgmask = cv2.morphologyEx(fgmask, cv2.MORPH_OPEN, kernel)
        fgmask = cv2.morphologyEx(fgmask, cv2.MORPH_CLOSE, kernel)

        # Find contours in the foreground mask
        contours, hierarchy = cv2.findContours(fgmask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

        # Find the largest contour that is not within another contour and has an area > 500
        largest_contour = None
        for i, contour in enumerate(contours):
            area = cv2.contourArea(contour)
            if hierarchy[0][i][3] == -1 and area > 500:
                if largest_contour is None or area > cv2.contourArea(largest_contour):
                    if xy_point is not None:
                        rect = cv2.minAreaRect(contour)
                        if cv2.rotatedRectangleIntersection(rect, (xy_point, (225, 225), 0))[0] == cv2.INTERSECT_NONE:
                            largest_contour = contour

                        cv2.rectangle(image, (int(xy_point[0] - 100), int(xy_point[1] - 100)),
                                      (int(xy_point[0]) + 100, int(xy_point[1]) + 100), (255, 50, 50), 2)
                    else:
                        largest_contour = contour

        # Get the centroid of the bounding rectangle of the largest contour
        if largest_contour is not None:
            x, y, w, h = cv2.boundingRect(largest_contour)
            cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)
            centroid_x = x + w / 2
            centroid_y = y + h / 2
            return (centroid_x, centroid_y)
        else:
            return None
```

Joonis 20. Liikuvate objektide tuvastus kood

5.3 Liikumine

Robotit liigutavad 2 harjadeta elektrirulades kasutatavat mootorit, mida juhib *Dual FSESC4.20* kiiruskontroller. Kontroller võtab vastu PWM signaali vahemikus 1000-2000 µs, kus 1500 µs on keskväärtus, mille juures mootorid ei liigu. Signaali võtab vastu *TBS Crossfire Nano Rx* vastuvõtja. Kõik ülaltoodu on roboti korpuse sees ja seda ei ole muudetud, roboti autonoomseks tegemise käigus.

Signaali edastamiseks kasutatakse saatjat *RadioMaster TX16S* treener režiimiga, mis võimaldab saata signaali saatjale väljastpoolt, kaotamata seejuures võimalust robotit ise juhtida. USB to PPM moodul on ühendatud USB-kaabli abil sülearvutiga ja suhtleb arvutiga *serial* ühenduse kaudu.

Signaal luuakse kahe signaali kombinatsioonist, mis liidetakse kokku - signaal pööramiseks ja signaal edasiliikumiseks. Mõlemad signaalid luuakse PID regulaatori funktsiooniga. Pööramissignaal saadab PID regulaatori funktsioonile positiivse või negatiivse nurga vastaseni ja saab funktsioonilt väärtuse, millele liidetakse 1500. Nagu eelnevalt mainitud, tähendab see väärtus mootorite peatumist ja seda suurendades või vähendades hakkab ratas pöörama ühes või teises suunas, mis paneb roboti liikuma. Liikumiseks saadetakse roboti kaugus vaenlasest PID regulaatori funktsiooni sisendisse ja väljundväärtus lisatakse pööramissignaali väärtusele. Koos juhivad need 2 signaali robotit efektiivselt.

```
if not math.isnan(angle):
    signal = pid_rotation.calculate(0, angle)
    signal_speed = pid_distance.calculate(0, distance)
    signal_speed = abs(signal_speed)

    if angle < 0:
        signal = signal * -1

    center_pos = 1500 + signal_speed
    signal_left = center_pos - signal
    signal_right = center_pos + signal

    # print(signal_left, signal_right, txt, signal_speed)
    try:
        if serial:
            if distance > 50:
                data_sender.send_signal_to_motors(int(signal_right),
int(signal_left))
            else:
                data_sender.send_signal_to_motors(1500, 1500)
    except:
        print("Cant send SERIAL data")
        pass
```

Joonis 21. Roboti liikumis loogika

PID-regulaator on tehisjuhtimissüsteemidest üks levinumaid tagasisidestatud juhtimisskeeme, kus rööbiti on lülitatud proportsionaal-, integraal- ja diferentsiaalregulaatorid. Ajalooliselt on PID-regulaatorit peetud parimaks signaali reguleerijaks. (Bennett, 1993)

Signaali väärtust uuendatakse iga programmi tsükli järel ja saadetakse USB to PPM moodulile kasutades *serial* ühendust ja seda korrigeeritakse iga uue fotoga.

Roboti juhtimise loogika on maksimaalselt lihtne, robot püüab pöörata vaenlase suunas lisades pöörete signaali kauguse signaalile, mis paneb roboti liikuma sihtkoha suunas (Joonis 21).

6 KATSETE TULEMUSED

Pärast roboti liikumise viimistlemist rohkete katsetega oli järgmiseks sammuks mitmete eksperimentide tegemine, mille eesmärk oli saavutada roboti täpsem liikumine ja väiksem reaktsiooniaeg, kui inimese reageerimiskiirus seda võimaldaks.

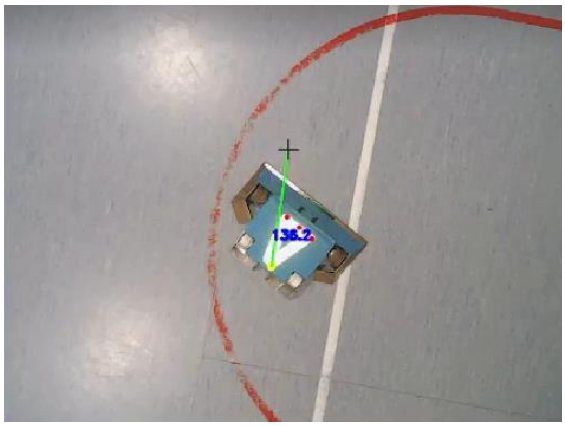
Katsete käigus muudeti mitmeid PID-regulaatori parameetreid, kiirusi ja muid programmi erinevate aspektide eest vastutavaid muutujaid.

Sinine ruut ümber robotit tähistab piirkonda kus liikumist ignoreeritakse et mitte ajada sassi enda tiimi robotit vastase omaga.

Hiire kursori katse

Esimene katse seisnes roboti liigutamises arvutiekraanil hiirega näidatud suunas Joonis 22. Kasutaja poolt sai anda ekraanil punkti, kuhu robot teoreetiliselt pidi jõudma. Selle katse käigus korrigeeriti mitu korda PID-kontrolleri väärtusi.

Selle katse arvukad kordused on andnud optimaalsed PID kontrolleri väärtused ja muljetavaldavad tulemused roboti reageerimiskiiruse ja liikumise täpsuse osas Lisa 2 *hiire kursori test.*



Joonis 22. Hiire kursoriga katse

Liikuva vaenlase tuvastamise katse

Selle katse eesmärk oli välja selgitada videovoo liikumise tuvastamise täpsus. Antud katses kasutati kaugjuhtimispuldi abil juhitavat omnidirektionaalset robotit, et võimaldada igas suunas liikumist. Nagu videost näha Lisa 3 *Liikuva vaenlase tuvastamise katse* õnnestus liikumise tuvastamine suurepäraselt, mis võimaldas ühendada selles katses saadud andmed eelmise katsega.

Liikuva objekti tuvastamise katse palli abil

Selles katses kasutati võrkpalli, et tekitada pildil liikumist. Nagu videost näha Lisa *4 Palliga liikumis tuvastus* pööras robot kohe, kui pall lahkus sinisest alast - ehk ala roboti ümber, kus ei toimu liikumise tuvastamist - sihtmärgi suunas ja liikus selle poole. Sinine ala on aga liiga suur ja niipea, kui pall sinna jõudis, jäi robot hetkeliselt seisma, kuid jätkas liikumist, kui sihtmärk sellest väljus.

Seisva objekti box-rush

Box rush on võitlusrobootikas kasutatav strateegia, mille puhul robot ründab kiiresti oma vastase suunas, püüdes teda areenilt välja lükata või ümber pöörata. Seda strateegiat kasutavad sageli robotid, millel on kiilu või küna kujuline esiosa, mis on mõeldud vastase roboti alla jõudmiseks ja selle üles tõstmiseks.

Box rush strateegia nõuab kiiret ja võimsat robotit, mis suudab kiiresti liikuda ja anda tugeva esialgse löögi. Eesmärk on üllatada vastast ja saada kiiresti ülekaal, lüües teda tasakaalust välja või põhjustades tema veojõu kaotamise.

Box rush on suure riskiga ja suure tasuvusega strateegia. Kui rush on edukas, võib robot oma vastase kiiresti demobiliseerida ja matši võita. Kui aga ründamine ebaõnnestub, võib robot takerduda või saada kahjustusi, mis muudab ta haavatavaks vastase vasturünnakute suhtes.

Selles katses asetati kaamera vaatevälja raske rätikpall, mis sümboliseeris vastast ja meie robotit, mis seisis kerge nurga all, et testida roboti reaktsioonikiirust ja täpsust liikumisel sihtmärgi suunas.

Tulemusena näitas robot vastase ründamisel suurepärast täpsust ja kiirust, mis on ideaalselt sobilik seda tüüpi robotile, sest box-rush nõuab suurt täpsust ja kiirust, Lisa 5 *box rush.*

KOKKUVÕTTE

Käesoleva lõputöö projekti käigus viidi antud ülesande lahendamiseks läbi rida teste ja katsetusi. Esialgu analüüsiti võistlusreegleid, milles robot osaleks, ning valiti autonoomse juhtimissüsteemi paigaldamiseks sobiv platvorm.

Projekti käigus projekteeriti ja arendati välja spetsiaalne moodul, mis edastab saatja ja arvuti vahel USB-ühendust kasutades PPM-signaali. Lisaks muudeti kaamerat, et parandada roboti tuvastamise kvaliteeti, paigaldades sellele lisa valgustuse, mis valgustas tuvastatud objekti, muutes tuvastamise lihtsamaks.

Pärast tuvastatava objekti valimist tunnistati reflektor parimaks vahendiks tuvastatavuse uuringu käigus, kasutades lisa valgustusega kaamerat. Järgmise sammuna valiti tuvastatava objekti kuju ning pärast kolme võimaluse kaalumist valiti kolmnurkne kuju, kuna selle nurga kõikumine oli minimaalne.

Vaenlase tuvastamiseks kasutati videovoos liikumise tuvastamise algoritmi, mis käivitub ainult siis, kui uurimistöös valminud robot on tuvastatud. Vale tuvastamise vältimiseks eemaldati see robot liikumise tuvastamise piirkonnast.

Kogu programm kirjutati Pythonis, mis valiti selle lihtsuse ja autori kogemuse tõttu selles keeles programmeerimisel. Programmi kood koosneb enam kui 1000 koodi reast, mis ei hõlma mitte ainult programmi ennast, vaid ka testimise, andmete kogumise ja andmeedastuse skripte.

Pärast koodi kirjutamist viidi roboti reaktsiooni kiiruse ja liikumise kontrollimiseks läbi arvukaid teste, mis on leitavad allalaaditavatest failidest lisades. Lõpptulemused näitavad, et robot saavutas edukalt projekti eesmärgi luua masinnägemise abil juhitav robot, mis suudab liikuda ja reageerida kiiremini kui inimpiloot. Katsete käigus oli roboti reaktsioonikiirus viis korda kiirem kui inimese keskmine reaktsioonikiirus, mistõttu oli projekt edukas.

KASUTATUD ALLIKAD

(n.d.). Retrieved from et.wikipedia.org.

(n.d.). Retrieved from gdevelop: https://gdevelop.io/

About. (n.d.). Retrieved from opencv.org: https://opencv.org/about/

ArUco marker detection (aruco module). (n.d.). Retrieved from docs.opencv.org: https://docs.opencv.org/4.x/d9/d6d/tutorial_table_of_content_aruco.html

Bennett, S. (1993). *et.wikipedia.org*. Retrieved from PID-regulaator: https://et.wikipedia.org/wiki/PID-regulaator

docs.opencv.org. (n.d.). Retrieved from Open Source Computer Vision: https://docs.opencv.org/3.4/d4/dc6/tutorial_py_template_matching.html

Lazaridis, G. (2009, märts 10). *PWM Modulation*. Retrieved from pcbheaven.com: http://pcbheaven.com/wikipages/PWM_Modulation/

Rosebrock, A. (2021, 1 19). *Image Masking with OpenCV*. Retrieved from pyimagesearch.com: https://pyimagesearch.com/2021/01/19/image-masking-with-opencv/

7 LISADE LOETELU

Videod

1. Lisa 1 *Gdevelop simulator*
2. Lisa 2 H*iire kursori test*
3. Lisa 3 *Liikuva vaenlase tuvastamise katse*
4. Lisa *4 Palliga liikumis tuvastus*
5. Lisa 5 *Box rush*