TALLINNA TEHNIKAÜLIKOOL
INSENERITEADUSKOND
Elektroenergeetika ja mehhatroonika instituut

MASINNÄGEMISRAKENDUS ROBOTI JUHTIMISEKS MIKROKONTROLLERI ABIL

MICROCONTROLLER BASED MACHINE VISION APPLICATION FOR ROBOT CONTROL

BAKALAUREUSETÖÖ

Üliõpilane: Janno Kikojan

Üliõpilaskood: 163908EAAB

Juhendaja: Mart Tamre, professor

Tallinn, 2020

(Tiitellehe pöördel)

AUTORIDEKLARATSIOON

Olen koostanud lõputöö iseseisvalt.

Lõputöö alusel ei ole varem kutse- või teaduskraadi või inseneridiplomit taotletud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

"....." ..... 201.....

Autor: .....

/ allkiri /

Töö vastab bakalaureusetöö/magistritööle esitatud nõuetele

"....." ..... 201.....

Juhendaja: .....

/ allkiri /

Kaitsmisele lubatud

"....."..... 201... .

Kaitsmiskomisjoni esimees .....

/ nimi ja allkiri /

LÕPUTÖÖ LÜHIKOKKUVÕTE

Autor: Janno Kikojan *Lõputöö liik:* Bakalaureusetöö

Töö pealkiri: masinnägemisrakendus roboti juhtimiseks mikrokontrolleri abil

Kuupäev: 20.05.2020 62 *lk (lõputöö lehekülgede arv koos lisadega)*

Ülikool: Tallinna Tehnikaülikool

Teaduskond: Inseneriteaduskond

Instituut: Elektroenergeetika ja mehhatroonika instituut

Töö juhendaja(d): professor Mart Tamre

Töö konsultant (konsultandid): Puudub

Sisu kirjeldus:

Töö põhieesmärkideks on robotite õpetamise uurimine ja nende liikuma panemine masinnägemise abil, kus kasutavateks seadmeteks on nii värvi- kui ka sügavuspilti edastav Intel RealSense D415 kaamera ja NVIDIA Jetson Nano mikrokontroller.

Nende omavahelise liidestamise järel toimub markeri tuvastamine ja selle X-Y-Z koordinaatide väljastamine. Saadud andmebaas koos vajaliku informatsiooniga saadetakse arvutile edasiseks töötlemiseks. Seejärel genereeritakse selle järgi ka roboti liikumine.

Märksõnad: robotid, objekti tuvastus, koordinaadid, kaamera, mikrokontroller.

ABSTRACT

Author: Janno Kikojan *Type of the work:* Bachelor Thesis

Title: microcontroller based machine vision application for robot control

Date: 20.05.2020 62 *pages (the number of thesis pages including appendices)*

University: Tallinn University of Technology

School: School of Engineering

Department: Department of Electrical Power Engineering and Mechatronics

Supervisor(s) of the thesis: Associate Professor Mart Tamre

Consultant(s): None

Abstract:

The main objectives of this thesis are to study the teaching of robots and control their movements using machine vision. The author uses an Intel RealSense D415 camera that can transmit both color and depth picture and an NVIDIA Jetson Nano microcontroller.

After connection of the devices the object identification process occurs. The next operation is to find the X-Y-Z coordinates of the detail and store these in a database. These coordinates are sent to the computer for generating robot movement destinatinons.

Keywords: robots, object recoqnition, coordinates, camera, microcontroller.

LÕPUTÖÖ ÜLESANNE

Lõputöö teema:	**Masinnägemisrakendus roboti juhtimiseks mikrokontrolleri abil**
Lõputöö teema inglise keeles:	**Microcontroller Based Machine Vision Application for Robot Control**
Üliõpilane:	**Janno Kikojan, 163908EAAB**
Eriala:	**Elektroenergeetika ja mehhatroonika**
Lõputöö liik:	**bakalaureusetöö**
Lõputöö juhendaja:	**Professor Mart Tamre**
Lõputöö ülesande kehtivusaeg:	**Kehtivusaja annab juhendaja**
Lõputöö esitamise tähtaeg:	**20.05.2020**

_____	_____	_____
Üliõpilane (allkiri)	Juhendaja (allkiri)	Õppekava juht (allkiri)

1. Teema põhjendus

Üldiselt võtab tööstuslike robotite käsitsi seadistamine palju aega. Masina ülesseadmine uutele ülesannetele võib toimuda ka video või pildi abil. Selleks peab masin valitud tegevust või liigutust õppima operaatorit või teist robotit vaadates ning seejärel jäljendades. See aitab muuta inimeste elu lihtsamaks, sest osa tööst tehakse nende eest ära. Teema aitab ka algajatele selgitada uuemate kaamerasüsteemide võimalikke kasutamisviise ja töökindlust robootika valdkonnas.

2. Töö eesmärk

Töö eesmärgiks on uurida robotite õpetamist masinnägemise abil. Sellega tekitatakse süsteem, kus robot koostab endale iseseisvalt liikumiseks vajaliku programmi. Selleks kasutatakse Intel RealSense D415 kaamerat ja NVIDIA Jetson Nano mikrokontrollerit.

3. Lahendamisele kuuluvate küsimuste loetelu:

1) Intel RealSense kaamera D415 ja NVIDIA Jetson Nano kontrolleri liidestamine.
2) Pilditöötlus tarkvara loomine mikrokontrolleri jaoks.
3) Objekti/markeri tuvastamine ja X-Y-Z koordinaatide määramine.
4) Video või pildi pealt vajalike koordinaatide salvestamine andmebaasi.

5) Roboti liidestamine kontrolleriga ning koordinaatide saatmine robotile.
6) Roboti liikumise genereerimine.

4. Lähteandmed

Eesmärkide lahendamiseks plaanin kasutada nii kaamera kui ka mikrokontrolleri andmelehti ning veebikeskkonnast või foorumitest leitavaid sarnasel teemal olevaid lahendusi, mida enda ülesande järgi korrigeerida.

5. Uurimismeetodid

Töö tulemusteni jõudmine põhineb varasemate sarnaste lahenduste uurimisel, leitava teabe analüüsil ning katsetamisel.

6. Graafiline osa

Graafiline osa on peamiselt töö põhiosas. Sinna kuuluvad erinevad kirjutatavad koodilõigud ning kasutatavate objektide/detailide(kaamera, mikrokontroller jne) paiknemise skeem süsteemis.

7. Töö struktuur

1) NVIDIA Jetson Nano ja Intel RealSense kaamera D415 tööle panek.
    - Kaamerapildi kuvamine ning sellest edastatava informatsiooni kättesaadavaks tegemine.
    - Pildil olevate markerite eristamine väliskeskkonnast.
    - Markerit kasutades visuaalse inimskeleti lüli tekitamine.
    - Video salvestamine pildina.
    - Vajalike koordinaatide salvestamine.
2) Informatsiooni robotile saatmine.
3) Roboti liikumise genereerimine.

8. Kasutatud kirjanduse allikad

1) https://www.intelrealsense.com/depth-camera-d415/
2) https://developer.nvidia.com/embedded/jetson-nano-developer-kit
3) https://ieeexplore.ieee.org/document/8901025
4) https://ieeexplore.ieee.org/document/8833143

9. Lõputöö konsultandid

Puudub.

10. Töö etapid ja ajakava

1) Intel RealSense kaamera D415 ja NVIDIA Jetson Nano andmelehtede uurimine (15.01.20).
2) Varasemate samalaadsete projektide uurimine ja lähteandmete kogumine (02.02.20).
3) Kaamera ja mikrokontrolleri tööle panek (03.03.20).

4) Informatsiooni robotile saatmine (20.03.20).
5) Roboti liikumise genereerimine (10.04.20).
6) Teoreetilise osa kirjutamine (15.04.20).
7) Järelduste ja kokkuvõtte kirjutamine (17.04.20).
8) Töö esimene versioon valmis (19.04.20).
9) Juhendajale läbilugemiseks saatmine (21.04.20).
10) Paranduste sisseviimine (03.05.20).
11) Juhendajale teiseks läbilugemiseks saatmine (05.05.20).
12) Töö lõplik versioon valmis (16.05.20).

SISUKORD

LÜHENDITE JA TÄHISTE LOETELU

BGR	3D värviruum, mis hõlmab vastavalt sinise, rohelise ja punase kanali väärtuste olemasolu andmemahulises kasvavas järjekorras (ingl Blue, Green, Red)
Depth	Sügavuspilt, tavaliselt kasutatakse Z16 formaati
Dilation	Protsess, mille käigus toimub pildil paiknevate kujundite parandamine nende kontuuri laiendamise või suurendamise abil
DisplayPort	Digitaalne pildi kuvamisliides
Erosion	Protsess, mille käigus toimub pildil paiknevate kujundite kontuuride parandamine nende vähendamise abil
EUR	Euro, rahaühik
Camera Extrinsic Value	Kaamera välised parameetrid, mille abil määratakse paiknemine reaalmaailmas
Gaussian filter	Pildi töötlemise rakendus, mis on mõeldud müra eemaldamiseks
GB	Andmete edastamiseks kasutatav infoühik (ingl Gigabyte)
HDMI	Digitaalne kõrglahutusega pildi ja heli kuvamisliides (ingl High-Definition Multimedia Interface)
HP	Rahvusvaheline infotehnoloogia ettevõte (ingl Hewlett-Packard)
HSV	Värviruum, mis on defineeritud värvitooni, küllastuse ja väärtuse (ingl Hue, Saturation, Value)
Imutils	Pilditöötlemise ja kontuuride joonistamise ning kuvamise rakendus
Intel RealSense Viewer	Rakendus kaamera käivitamiseks ning reaalajaliseks informatsiooni edastamiseks, kasutatud versiooni v2.31.0
Camera Intrinsic Value	Kaamera sisemised parameetrid, mis on fikseeritud
IP Rating	Kaitseaste, mis iseloomustab elektriseadme turvalisust välismõjude, sealhulgas nii tolmu kui ka vedelike suhtes (ingl International Protection)
IR	Infrapunakiirgus (ingl Infrared light)
IRB 14400(YuMi)	ABB roboti versioon, mis on võimeline inimeste kõrval samas keskkonnas töötama
IRB 1600	ABB tööstusroboti versioon (ingl Industrial Robot [IRB])
IRC5	Robootikatööstuses kasutatav kontroller (ingl Industrial Robot Controller)
Kinect	Seadmete komplekt, mille abil on võimalik luua virtuaalne ühendus tarkvara ja kasutaja vahel

LAN	Seadmete vaheline privaatne võrguühendus (ingl Local Area Network)
MicroSD	Mälukaardi formaat, mis on väikeste mõõtmetega
NumPy	Rakendus andmemassiivide töötlemiseks (ingl Numerical Python)
OpenCV	Masinnägemis- ja õppimisrakenduste kogum (ingl Open Source Computer Vision Library)
OpenPose	Tarkvara süsteem, mille abil võib toimuda virtuaalse inimskeleti loomine
Piksel	Kujutisel paiknev vähim osake või element, mida on võimalik töödelda
PyrealSense	Inteli kaamera seadete ja üldise kasutuse tarkvara
Python	Kõrgetasemeline programmeerimiskeel
Raspberry PI MIPI CSI	Mikrokontrolleri külge ühendatav kaamera süsteem
RGB	3D värviruum, mis hõlmab vastavalt punase, rohelise ja sinise kanali väärtuste olemasolu, andmemahulises kahanevas järjekorras (ingl Red, Green, Blue)
RobotStudio	ABB firma robotite juhtimise ja simulatsioonide tegemise programm
RobotWare	ABB roboti kontrolleri tarkvara
Slaider	Muudetavate väärtuste ribadega liugur
Socket server	Seadmete ühendus- ja andmevahetus kanal
TCP/IP	Usaldusväärne andmete edastus meetod koos tuvastatava järjekorra ja veakontrolliga (Ingl Transmission Control Protocol/Internet Protocol)
Ubuntu	Kasutatava mikrokontrolleri operatsioonisüsteem
USB	Seadmete omavaheliseks ühendamiseks mõeldud standard, mis hõlmab eriversioone, sealhulgas A-, C- ja Micro tüüpe (ingl Universal Serial Bus)

SISSEJUHATUS

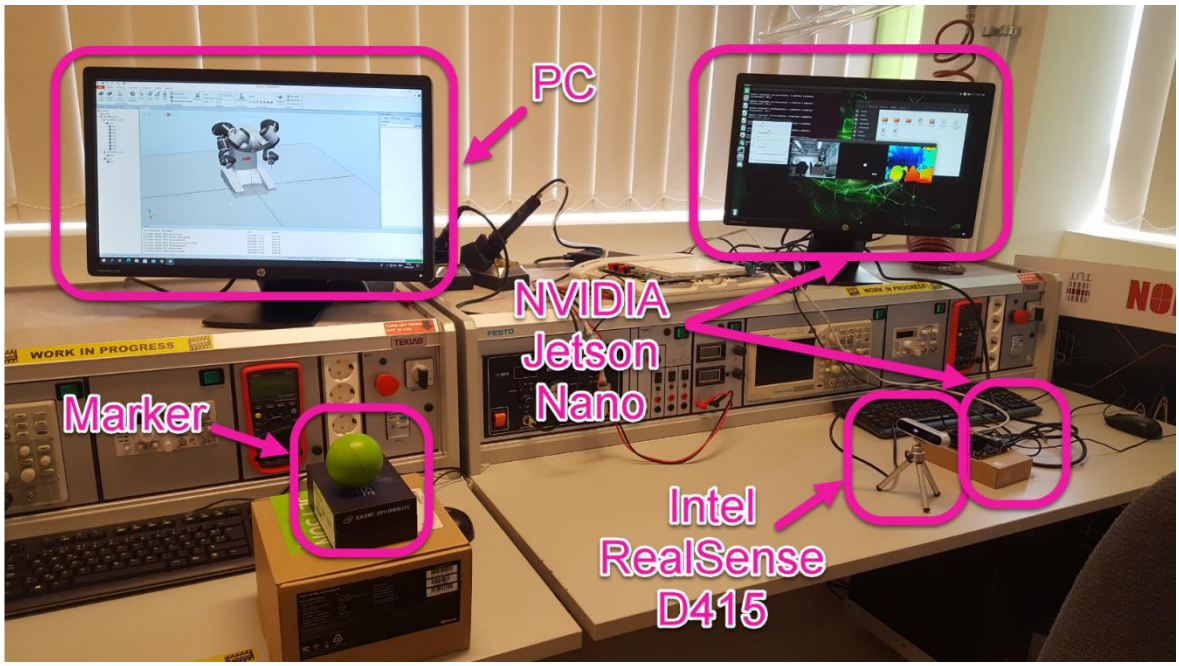
Lõputöö eesmärgiks on uurida robotite õpetamist masinnägemise abil. See võimaldab süsteemi loomist, kus robot koostab endale iseseisvalt liikumiseks vajaliku programmi. Selleks kasutatakse peamiselt Jetson Nano mikroarvutit ja Intel RealSense D415 kaamerat. Autori arvates on mikrokontroller piisavalt võimekas – ehk suudab hallata rohkelt informatsiooni ning samal ajal olla hinna poolest soodne. Samuti on kaamera abil võimalik pildi või video peal olevaid andmeid kergesti töödelda.

Võimalik on kasutada ka veebikaamerat, kuid see edastab vaid kahe dimensioonilist pilti ega anna kauguste kohta informatsiooni. Ruumilise pildi saamiseks on vaja kasutada triangulatsiooni või samaaegset lokaliseerimis- ja kaardistamissüsteemi. Lisaks eeltoodule on võimalik kasutada ka Kinecti, kuid ka see on aegunud ja piiratud võimalustega, sest ametlikku arendustegevust enam ei toimu [1]. Töös kasutatav kaamera on robootika ja automaatika valdkonnas kõige perspektiivsem ning uuem tehnoloogia. Lisaseadmete nimekirjas on arvuti hiir, klaviatuur ja monitor. Soovituslik on kasutada veel ventilaatorit, et kontrollida Jetson Nano radiaatori jahutustemperatuuri.

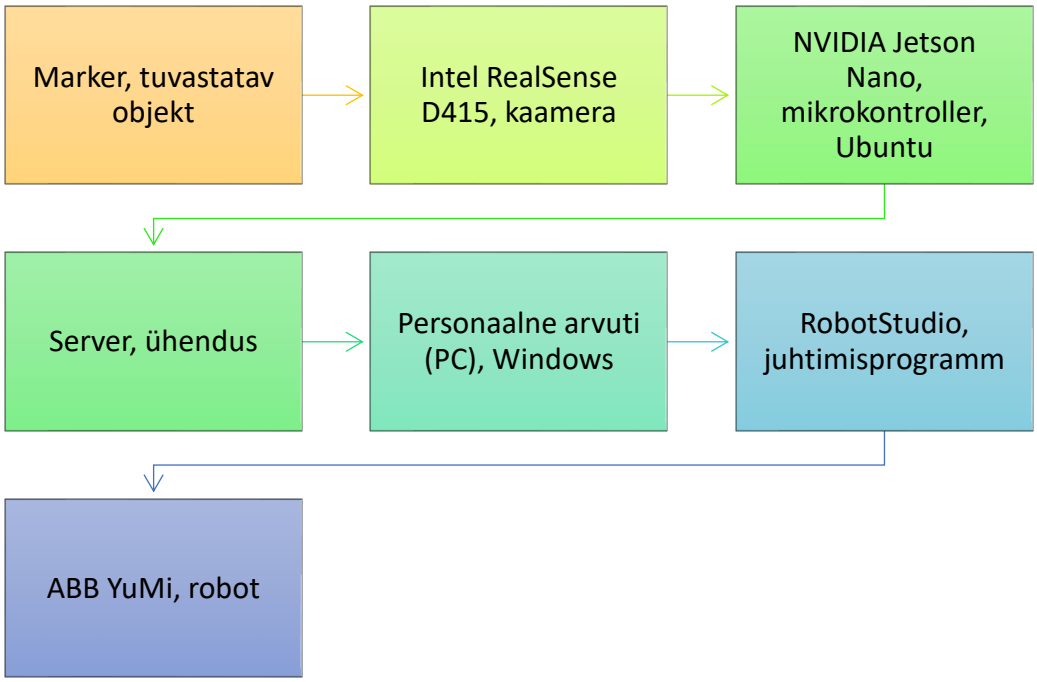
Esimeseks etapiks on NVIDIA Jetson Nano ja Intel RealSense D415 kaamera liidestamine (vt Joonis 0.1 ja 0.2). Kasutatud microSD mälukaart koos Ubuntu operatsioonisüsteemiga on suurusega 128 GB, mis on valitud eelkõige probleemide ennetamiseks, et oleks piisavalt vaba ruumi materjali haldamiseks ka pikaajalisel kasutamisel. Lisaks on valitud kaardil piisav lugemis- ja kirjutamiskiirus. Kaamera ühendatakse mikrokontrolleri külge USB kaabli abil.

Seejärel alustatakse kaamerasüsteemi etapiga, kus hakatakse kirjutama programmi, et kaamerast tulevat pilti töödelda. Selle eesmärgiks on varem defineeritud detaili või markeri tuvastamine, et võimalusel hilisemalt virtuaalne inimskelett koostada. Edasisteks tegevusteks on videomaterjalist üksiku pildi tegemine ning sellest vajaliku informatsiooni saamine. Täpsemalt salvestatakse andmebaasi varem mainitud eseme X-Y-Z koordinaate.

Järgmiseks luuakse mikrokontrolleri ja arvuti vahel kanal andmete vahetamiseks ning edastatakse tekstifail. Edasine tegevus toimub robotkeskkonnas RobotStudio programmis, kus saadud fail loetakse sisse ja muudetakse masinale arusaadavaks. Lisaks koostatakse selle järgi masina liikumisskeem.



Joonis 0.1 Reaalne komponentide paiknemine



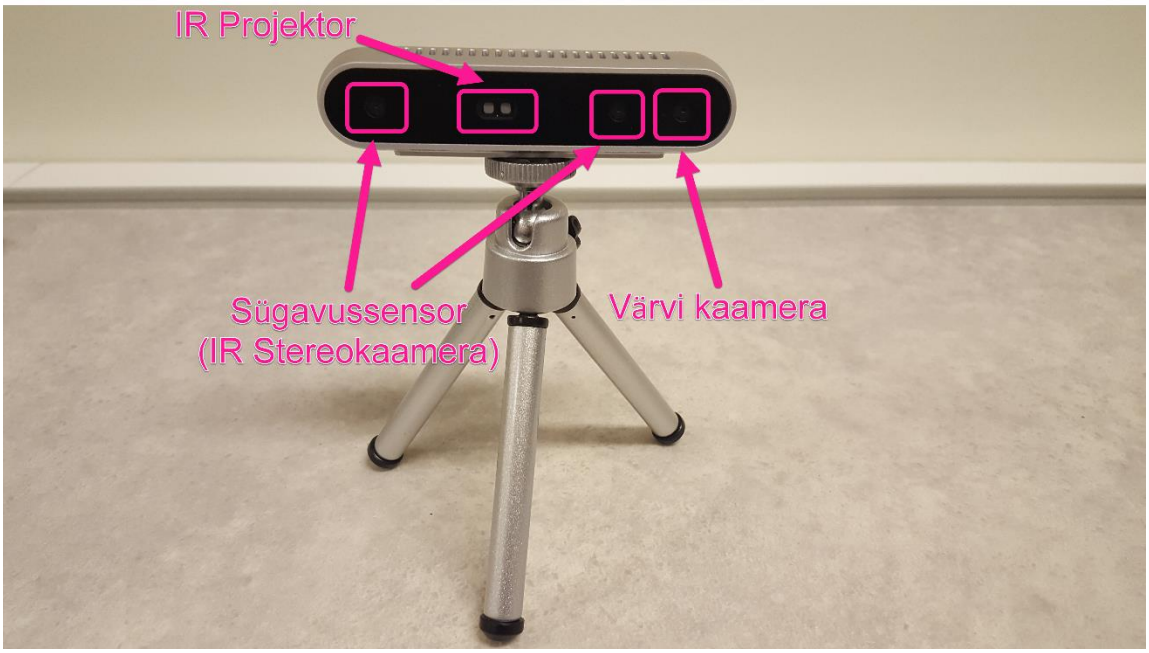
Joonis 0.2 Asetusskeem

1. INTEL REALSENSE D415 KAAMERA

1.1 Andmed

Kaamera on piisavalt võimekas ja tarkvara kergesti töödeldav ning arendatav. See sobib valdkondadesse, kus täpsus on olulisel kohal. Sinna hulka kuuluvad näo- või objektide tuvastamine, robotite ja/või muude sõidukite juhtimine ning siseruumide või väliskeskkonna ruumiline skaneerimine. Lisaks on seadmega võimalik näha nii tavalist värvipilti (RGB) kui ka sügavuspilti (Depth). Viimast kasutades on võimalus kahe IR stereokaamera (vt Joonis 1.1) abil teada saada ka ümbritsevate detailide kauguseid, mis on minimaalselt ligikaudselt 0,2 ja maksimaalselt 10 meetrit. See on limiteeritud ning sõltub peamiselt kasutatavast resolutsioonist, vaadeldavate esemete suurusest ning peale langevast valgusest. [2]

Lisaks on RealSense kaamera oma mõõtmetelt väike (99 [pikkus] x 20 [laius] x 23 [kõrgus] mm), massilt kerge (72 g) ja hinna poolest soodne (ligikaudselt 140 EUR [3]). Maksimaalsed resolutsioonid värvi- ja sügavuspildi juures on vastavalt 1920 x 1080 pikslit 30-kaadriga sekundis ning 1280 x 720 pikslit 90-kaadriga sekundis ja vaatevälja nurgad (horisontaalne x vertikaalne x diagonaalne) vastavalt 69° ± 1° x 42° ± 1° x 77° ± 1° ning 65° ± 2° x 40° ± 1° x 72° ± 2°. Kaameraga on kaasas veel õpetusraamat, kolmjalgne alus ning välisseadmega ühendatav USB-C kaabel. [4]



Joonis 1.1 Intel RealSense D415 koos vajalike funktsioonidega, kus paremalt vasakule lugedes paiknevad värvi- ja kaks infrapuna kaamerat ning nende vahele jääv infrapuna sensor/projektor sügavusandmete kogumiseks.

1.2 Kasutamine ja kalibreerimise vajalikkus

Kaamera peab olema asetatud stabiilsele ja tasapinnalisele alusele (vt Joonis 0.1). Olulisel kohal on ka ruumi valgustatus. Põhieesmärgiks on seadme ees oleva detaili nägemine, et hilisemaks töötlemiseks võimalikult palju andmeid saada. Lisaks toimub toite saamine läbi mikrokontrolleri. Värvi- ja sügavuspildi kasutamise tõttu sobib mainitud kaamera lõputöö valdkonda.

Kaamera kalibreerimine on oluline kvaliteetsema pildi saavutamiseks. Selle käigus korrigeeritakse objektiivi ja sensorite parameetreid, et tekkinud moonutusi vähendada. Olulisel kohal on reaalse ruumilise ja kahe-dimensionaalse pildi punktide omavahelise vastavuse loomine. Üheks nende saamise võimaluseks on male- või kabelaua kasutamine, kus vastavad koordinaadid saadakse mustri nurkadest. Parema tulemuse saamiseks on soovituslik vaadeldavast objektist teha rohkem kui kümme pilti, seda nii kaamera ees erinevates asukohtades liigutades kui ka nurga alla pöörates. [5]

Kalibreerimise käigus leitakse kaamera maatriks, mis on iga seadme jaoks ainulaadne. Arvutustest tulenevalt saadakse ka projektsioonide väärtused ideaalse olukorra kohta. Mida lähemal on nende omavaheline erinevus nullile, seda täpsemad on kaamera parameetrid [6]. Kui kaamera on saanud mehaanilisi-või niiskuskahjustusi või kui arvutuste käigus saadakse ebatäpsed andmed, on kalibreerimise läbiviimine kohustuslik.

1.2.1 Töös vajaminevad teegid

- Pyrealsense – Intel RealSense kaamera parameetrite initsialiseerimise ja käivitamise kogum või moodul, mis sisaldab erinevaid kasutusnäiteid ning funktsioone [7]. Sealhulgas võimaldab kasutajale horisontaalse ja vertikaalse vaatevälja suuruse arvutamist, väljastada saadavate seadmete nimekirja, reaalajas edastamise parameetreid, tõrkeotsingu informatsiooni jne [8].

- NumPy – Andmemassiivide haldamise ja töötlemise võimalus, mis koosneb tööks vajalikest funktsioonidest [9]. Nende abil on võimalik väljastada ja ekraanile kuvada sama tüübi ja vormindusega numbrite maatrikseid või elementide tabeleid. Lisaks võib massiive sorteerida või nendest vastava järjekorra moodustada [10].

Autori arvates on nii põhi kui ka lisa toimingute tegemine, sealhulgas andmehulkade omavaheliste tehete vormistamine, kasutaja jaoks lihtsa teostusega, sest need on loogilises järjekorras ning kergesti mõistetavad.

- OpenCV – Masinnägemis- ja õppimisfunktsioonide ning reaalaja rakenduste kasutamiseks mõeldud kogumik [11]. Kasutusvaldkonda kuuluvad nii vaadeldavate objektide tuvastamisprotsessid, sõiduki sõiduradadel hoidmise süsteemid kui ka pildi töötlemise ning värvi formaadi muutmise võimalused. Lisaks eelnevale toimib mainitud funktsioonide süsteem ka erinevaid programmeerimiskeeli ja operatsioonisüsteeme rakendades, mis tarbijate jaoks kasutamise mugavamaks muudavad. [12]

- Imutils – Pilditöötlus funktsioonide kasutamiseks loodud funktsioonide nimekiri. Sinna hulka kuuluvad nii foto pööramine kui ka suuruse ehk pikkuse ja laiuse muutmine. Lisaks on mainitud rakendust kasutatud ka kontuuride joonistamiseks. [13]

```
# Teekide lisamine
import pyrealsense2 as rs # Kaamera käivitamine
import numpy as np # Andme massiivide kasutamine
import cv2 # Pildi töötlemine
import imutils # Kontuuride tegemine
```

Joonis 1.2 Kasutatavad teegid (vt Lisa 1)

1.2.2 Kaamera seaded ja käivitamine

Kasutatakse funktsiooni, mille abil on võimalik pilditöötlus toimingute teostamine ja videopildi edastamine reaalajas. Selleks on vajalik kaamera peamiste parameetrite defineerimine. Tavaliselt on nendeks formaat, resolutsioon, kaadrite arv ja värvuse tüüp. Vaadeldav pilt on ristkülik suurusega 640 x 480 (horisontaalne x vertikaalne) pikslit. Selle vasak ülemine nurk loetakse nullpunktiks (0 x 0) ning maksimaalne punkt asetseb paremas alumises nurgas. Seega väärtused suurenevad liikudes vasakult paremale (pikkus) ja ülevalt alla (laius). Kaamera käivitamise ja mainitud seadete defineerimise programmikoodi on näha ka joonisel (vt Joonis 1.3).

```
# Kaamera seadete(suurus, värv, kaadrite arv) defineerimine
pipeline = rs.pipeline()
config = rs.config()
config.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)
config.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)

# Kaamera käivitus
profile = pipeline.start(config)
```

Joonis 1.3 Kaamera seadete profiili tegemine (vt Lisa 1)

1.2.3 Kaadriliikumise ja voogedastuse haldamine

Kaadrite haldamine toimub peamiselt süsteemi draiverites või tarkvaras, mistõttu on olulisel kohal latentsusaja ja jõudluse omavaheline suhe. Et saada reaalaajas piisavalt suure resolutsiooniga sujuvat kaamera pilti on vaja saadud kaadreid hilisemaks kasutuseks oote järjekorda salvestada. [14] Autori arvates võivad oote olukorra pikkust mõjutada arvutid või mikrokontrollerid, millel on vanemad protsessorid või aeglase lugemiskiirusega kõvaketas. Sellisel juhul ei jõua need piisavalt kiirelt sissetulevat andmehulka hallata. Kaadriliikumise ja voogedastuse haldamine ilmneb lisaks ka joonisel (vt Joonis 1.4).

```
# Kaadrite järjekorda salvestamine
# Värvi,-ja sügavuspilt
frameset = pipeline.wait_for_frames()
color_frame = frameset.get_color_frame()
depth_frame = frameset.get_depth_frame()
if not depth_frame or not color_frame:
    continue
```

Joonis 1.4 Värvi- ja sügavuspildi kaadriliikumise haldamine (vt Lisa 1)

1.2.4 Pildi kuvamine ja protsessi lõpetamine

Kasutatakse funktsiooni, mille abil on võimalik nii värvi, sügavus kui ka maskitud ehk töödeldud pildi ekraanile kuvamine. Lisaks on need varem mainitud resolutsiooniga ning asetsevad ühes aknas teineteise kõrval. Selleks, et kõik pildid ekraanile ära mahuksid on vaja kasutada vähenduskordajat, mille numbriline väärtus sõltub vastavalt kasutatava monitori resolutsioonist. Autor on valinud mainitud muutuja suuruseks 0,6.

Vajutades klaviatuuril nuppu „q“ või „Esc“ toimub protsessi katkestamine ning uuesti avamiseks on vajalik rakenduse taaskäivitamine. Lõplikku piltide virnastamise ja ekraanilt väljumiseks loodud programmikoodi on samuti näha joonistel (vt Joonis 1.5 ja 1.6).

```
# Lõpliku pildi kuvamine
imgStack = stackImages(0.6,([frame, mask, colorized_depth]))
cv2.imshow("Frame, Mask, Depth", imgStack)
key = cv2.waitKey(1)
```

Joonis 1.5 Pildi kuvamine (vt Lisa 1)

```
#Pildilt väljumiseks tuleb vajutada "q" tähte või "esc" nuppu
if key & 0xFF == ord("q") or key == 27:
    cv2.destroyAllWindows()
    break
```

Joonis 1.6 Pildilt väljumine (vt Lisa 1)

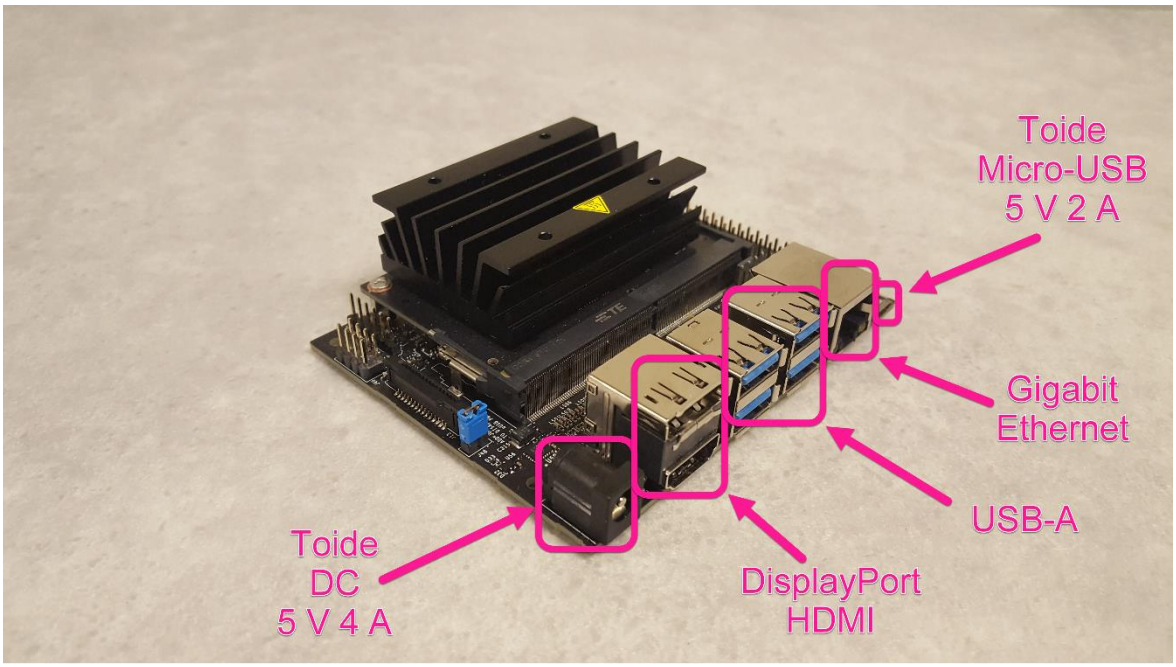
2. NVIDIA JETSON NANO MIKROKONTROLLER

2.1 Andmed

Mikrokontroller on mõõtmetelt väike (100 [pikkus] x 80 [laius] x 29 [kõrgus] mm), võimekas ning kergesti kasutatav arvuti laadne süsteem (vt Joonis 2.1), mille peamisteks kasutusvaldkondadeks on objekti tuvastamine, lokaliseerimine ja tehisintellekti või närvivõrkude töötlemine [15]. Lisaks on Jetson energiatõhus platvorm, mis võimaldab robotite õpetamist ja masinnägemise protsesse uurida reaalajas ning suurtes andmemahtudes [16]. Toite saamiseks võib kasutada Micro-USB 5 V 2 A porti või DC 5 V 4 A pesa [17]. Monitori jaoks on HDMI ja DisplayPort-pistik ning klaviatuuri, hiire või muu lisaseadme ühendamiseks neli USB-A (3.0) porti. Samuti on võimalus ka internetikaabli (Gigabit Ethernet) ja Raspberry PI MIPI CSI kaamera kasutamiseks [18]. Juhtmevaba võrgu tekitamiseks on vaja eraldi antennidega moodulit. Kontrolleri küljes oleva radiaatori kuumenemise tõttu on soovituslik kasutada 40 mm laiusega pulsilaiusmodulatsiooniga juhitavat ventilaatorit.

2.2 Kasutamine

Kontroller paikneb laual asetseva pappkarbist aluse peal, mille läheduses on samuti teised töös rakendatavad seadmed (vt Joonis 0.1). Toimingute tegemiseks on kasutatud tootja HP poolt loodud arvuti hiirt ja klaviatuuri ning tegevuste kuvamiseks EliteDisplay E231 monitori. Lisaks on hilisemates plaanides spetsiaalse korpuse loomine, mida on eelkõige lihtne kasutada ja mis kaitseb NVIDIA kontrollerit tolmu, vedeliku, kukkumiste ning muude välismõjude eest. Mikrokontrolleri peal toimub kaamerast saadud andmete töötlemine ja saatmine arvutile. Ühenduse loomiseks on kasutatud kaabliga koolivõrku. Rakenduste programmeerimise keel Python (versioon 2,7) on valitud autori kasutusoskuste järgi ning nende käivitamine toimub kontrolleri Ubuntu operatsioonisüsteemi terminali kasutades.



Joonis 2.1 NVIDIA Jetson Nano mikrokontroller koos peamiste kasutavate liidestega

3. OBJEKTI TUVASTAMINE

3.1 Defineerimine

3.1.1 Nupp ja järjekord

Kasutatud on OpenCV mooduli funktsiooni, mis võimaldab hetkelist olukorra muutumist või protsessi peatamist. Selleks on oluline defineerida klaviatuuril vajutatava nupu väärtus ning lülituse ajaline kestus, milleks praeguses ülesandes on üks millisekund. Lisaks toimub piltide salvestamine, mille nimeline järjekord algab number ühest. Nii nupu määratlemine kui ka järjekorra numbriline algus on edastatud ka programmikoodi joonisena (vt Joonis 3.1).

```
key = cv2.waitKey(1) # Nupu defineerimine
num = 1 # Salvestatud piltide number algab 1-st
```

Joonis 3.1 Nupu ja järjekorra alguse defineerimine (vt Lisa 1)

3.1.2 Piltide näitamine

Piltide väljastamise juures kasutatakse rakendust, mille abil on võimalik neil asetseda teineteise kõrval. Seda ainult juhul, kui tegemist on kas ühe-dimensionaalse olukorraga ehk ahelaga (vt Joonis 3.14) või kui üleval ja all reas on sama palju elemente (mitme-tasandiline). Kujutiste virnastamise meetod aitab kasutajal luua mugavat vaatevälja toimuvast, sest kõik tegevused on nähtavad üksteise kõrval ning ühes aknas. Lisaks on kogu piltide näitamise funktsioon toodud programmikoodina (vt Lisa 1).

3.1.3 Muudetava slaideri tegemine

Selles peatükis defineeritakse slaider, mille abil korrigeeritakse pildil ilmuvate pikslite värve. Autori arvates on soovituslik algselt välja uurida vastava värvuse ligikaudsed suurused, sest see lihtsustab hilisemas töö osas vajaliku vahemiku leidmist ning vähendab sellega kaasnevat katsetamiseks kuluvat aega. Täpsemad väärtused on saadud erinevates ruumides korduvaid katseid tehes. Rohelist värvi markeri nägemiseks on leitud tema asetsemisvahemik, mis sõltub enamjaolt peale langeva valguse kogusest ja tugevusest. Sealjuures on autor slaideri akna füüsilised mõõtmed valinud kasutatava monitori järgi selliselt, et need koos kuvatavate piltidega ekraanile ära mahuvad. Reguleeritavate väärtuste võimalustega liuguri defineerimine ja selle initsialiseerimine on toodud ka joonistel (vt Joonis 3.2 ja 3.3).

```
# Slaideri tegemine värvi korrigeerimiseks
def empty(a):
    pass
cv2.namedWindow("Slaider")
cv2.resizeWindow("Slaider", 300, 400)
cv2.createTrackbar("L-H", "Slaider", 29, 255, empty)
cv2.createTrackbar("L-S", "Slaider", 86, 255, empty)
cv2.createTrackbar("L-V", "Slaider", 6, 255, empty)
cv2.createTrackbar("U-H", "Slaider", 64, 255, empty)
cv2.createTrackbar("U-S", "Slaider", 255, 255, empty)
cv2.createTrackbar("U-V", "Slaider", 255, 255, empty)
```

Joonis 3.2 Slaideri tegemine (vt Lisa 1)

```
# Initsialiseerime värvislaideri, anname slaideri ridadele väärtused
l_h = cv2.getTrackbarPos("L-H", "Slaider")
l_s = cv2.getTrackbarPos("L-S", "Slaider")
l_v = cv2.getTrackbarPos("L-V", "Slaider")
u_h = cv2.getTrackbarPos("U-H", "Slaider")
u_s = cv2.getTrackbarPos("U-S", "Slaider")
u_v = cv2.getTrackbarPos("U-V", "Slaider")
greenLower = (l_h, l_s, l_v)
greenUpper = (u_h, u_s, u_v)
```

Joonis 3.3 Slaideri initsialiseerimine (vt Lisa 1)

3.2 Värvipildi töötlemine

3.2.1 Värvi formaadi korrigeerimine

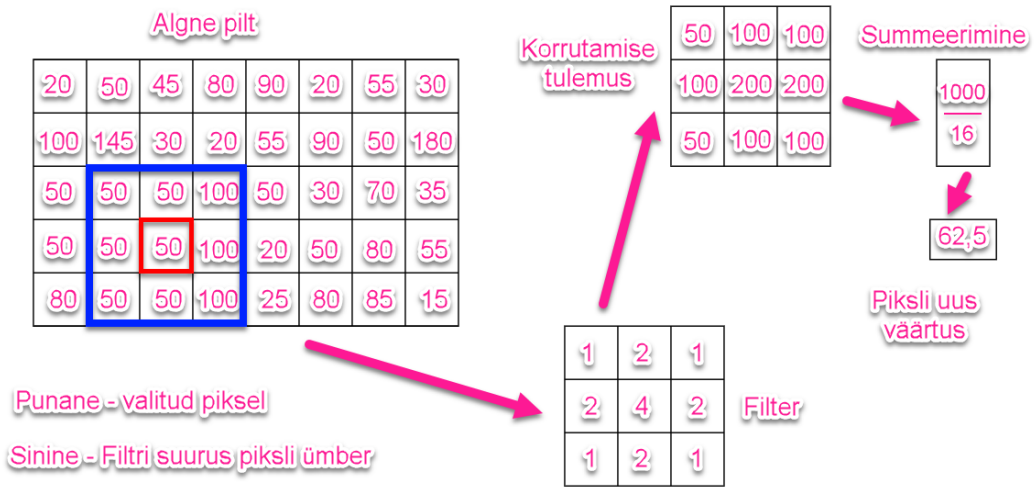
Autori arvates võib pildiks lugeda tasapinnalist kujutist või funktsiooni, mille koordinaatsüsteemi väärtused koosneb pikslitest X ja Y. Selle töötlemise käigus on sisendiks kaamerast saadav pilt ning väljundiks teatud vaadeldav detail – praegusel hetkel rohelist värvi marker. Vajaliku informatsiooni saamine toimub digitaalsel teel ehk manipuleeritakse sealsete pikslitega.

Kindla objekti tuvastamiseks on vaja seda teistest eristada. Selleks võib kasutada meetodit, kus ainult valitud markeri värv kuvatakse ekraanile valgena ning ülejäänud mustana – jääb taustaks. Liigutades objekti reaalajas on näha, kus see parasjagu paikneb. Lisaks kasutatakse andmemassiive, mis koosnevad varasemalt mainitud resolutsioonist ja värvi kanalitest. Viimast kirjeldab numbrite kogumik BGR, mis iseloomustab sinise (B), rohelise (G) ja punase (R) värvitoonide kogust. Need täisarvulised väärtused on nullist 255-ni. Sealjuures väljastab kaamera iga piksli kohta värve kaheksa biti jagu.

See aga ei anna teada informatsiooni pildi heleduse kohta ning seetõttu kasutatakse tihti värvi tuvastamise valdkonnas HSV formaati (vt Joonis 3.5). Täpsemalt iseloomustab see valitud värvitooni osa (Hue), kus võimalikku valikut on terve spekter, valge värvi sisaldust või küllastust (Saturation) ja värvi intensiivsuse tugevust (Value) süsteemis. Viimast võib võrrelda ka tumeduse reguleerimisega, sest madalatel väärtustel annab see musta, kõrgetel värvi enda. [19]

3.2.2 Värvipildi töötlemine

Lisaks asetseb pildil mitmeid segavaid faktoreid, millede teke võib olla põhjustatud nii valguse kõikumistest või kiirest muutusest kui ka sensori mürast või muudest häiringutest. Üks võimalus selliste probleemide lahendamiseks on saadud kujutise valitud koha udusemaks tegemine. See ühtlustab lähedal olevate pikslite intensiivsust ning aitab värvi üleminekuid sujuvamaks muuta [20]. Selliseks kasutatavaks funktsiooniks on Gaussian filter, mille tööpõhimõte põhineb valitud piksli väärtuse uuendamisel (vt arvutus allpool). Töös kasutatakse ruutu suurusega 11 x 11 pikslit (pikkus x laius), mis lohistatakse reaalse pildi kohale. Seejärel korrutatakse ristkülikusse jäävad väärtused ükshaaval samas asukohas asuvate filtri enda numbritega. Viimased on valitud selliselt, et keskkohas paikneb kõige suurem ning ääre poole liikudes väiksemad arvud. Tulemuste liites ja jagades filtri väärtuste summaga saadakse keskmine, mis on algselt valitud piksli uus väärtus (vt Joonis 3.4). Samuti mõjutab filtri suurus protsessi tegemiseks kuluvat aega, sest suurema ruudu kasutamine nõuab rohkem ressurssi. [21]



Joonis 3.4 Filtri (3 x 3) kasutamine, kus esialgselt pildilt on valitud piksel (punasega ümbritsetud) ning ümber selle joonistatud filtri suurune ala (sinisega ümbritsetud). Seejärel on korrutatud kõik filtri ja sinisesse alasse jäävad väärtused omavahel ning liidetud kokku. Saadud tulemuse jagamisel filtri summaga saadakse valitud piksli uus väärtus.

1) Pildi pikslite ja filtri väärtuste korrutamine

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix} = \begin{pmatrix} a_{11} \cdot b_{11} & a_{12} \cdot b_{12} & a_{13} \cdot b_{13} \\ a_{21} \cdot b_{21} & a_{22} \cdot b_{22} & a_{23} \cdot b_{23} \\ a_{31} \cdot b_{31} & a_{32} \cdot b_{32} & a_{33} \cdot b_{33} \end{pmatrix} \quad (3.1)$$

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix} = \begin{pmatrix} 50 \cdot 1 & 50 \cdot 2 & 100 \cdot 1 \\ 50 \cdot 2 & 50 \cdot 4 & 100 \cdot 2 \\ 50 \cdot 1 & 50 \cdot 2 & 100 \cdot 1 \end{pmatrix} = \begin{pmatrix} 50 & 100 & 100 \\ 100 & 200 & 200 \\ 50 & 100 & 100 \end{pmatrix}$$

kus $A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} 50 & 50 & 100 \\ 50 & 50 & 100 \\ 50 & 50 & 100 \end{pmatrix}$ – pildil valitud piksel ja teda ümbritsev ala,

$B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$ – 3 x 3 filter.

2) Tulemuse maatriksi väärtuste summeerimine

$$\sum_{c_{11}}^{c_{33}} C = c_{11} + c_{12} + c_{13} + c_{21} + c_{22} + c_{23} + c_{31} + c_{32} + c_{33} \quad (3.2)$$

$$\sum_{c_{11}}^{c_{33}} C = 50 + 100 + 100 + 100 + 200 + 200 + 50 + 100 + 100 = 1000$$

kus C – korrutamisel saadud maatriks.

3) Filtri väärtuste summeerimine

$$\sum_{b_{11}}^{b_{33}} B = b_{11} + b_{12} + b_{13} + b_{21} + b_{22} + b_{23} + b_{31} + b_{32} + b_{33} \quad (3.3)$$

$$\sum_{b_{11}}^{b_{33}} B = 1 + 2 + 1 + 2 + 4 + 2 + 1 + 2 + 1 = 16$$

kus B – 3 x 3 filter.

4) Piksli väärtuse leidmine

$$x = \frac{\sum_{c_{11}}^{c_{33}} C}{\sum_{b_{11}}^{b_{33}} B} \tag{3.4}$$

$$x = \frac{1000}{16} = 62{,}5$$

kus $\sum_{c_{11}}^{c_{33}} C$ – korrutamistulemuse maatriksi summa,

$\sum_{b_{11}}^{b_{33}} B$ – filtri väärtuste summa,

x – valitud piksli uus väärtus.

```
# Värvipildi töötlemine, massiviks tegemine
color_image = np.asanyarray(color_frame.get_data())
frame = color_image.copy()
blurred = cv2.GaussianBlur(frame, (11, 11), 0)
hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
```

Joonis 3.5 Värvipildi töötlemine (vt Lisa 1)

3.3 Sügavuspildi töötlemine

3.3.1 Sügavussensori käivitamine ja pildi töötlemine

Korrektse sügavusparameetri saamiseks on vaja peale kaamera sensori käivitamist leida skaala mõõtühikud. Seda kasutatakse hilisemaks kauguse arvutamiseks. Lisaks kuvatakse väärtus ekraanile. Anduri käivitamist ja vastavate parameetrite saamist on näha ka joonisel (vt Joonis 3.6).

Sügavusandmete leidmine võib toimuda värvipildi kaadrite ja vastava anduri abil. Selleks ühildatakse omavahel varasemalt kasutatud värvi- ja tehtav sügavuspilt. Kaugusandmete saamine toimub pikslite värvi alusel [22]. Sügavuspildi formaadiks on Z16, mille käigus edastatakse piksli kohta 16 bitti vajalikke kauguse andmeid (vt Joonis 3.7). Optimaalne sügavuspildi resolutsioon on 1280 x 720 pikslit (horisontaalne x vertikaalne), töös kasutatakse aga tunduvalt madalamat. Väiksema eraldusvõime kasutamine toob ühelt poolt kaasa sügavuse täpsuse halvenemise. Siinkohal tuleb arvestada, et protsesse haldab mikrokontroller, mitte võimekas personaalarvuti. Seega ilmnevad kõrge sisendresolutsiooni kaasamisel nii pildi kvaliteedi tõus ja sügavusparameetrite täpsus kui ka jõudluse nõudluse suurenemine. [23]

```
# Sügavussensori käivitamine
depth_sensor = profile.get_device().first_depth_sensor()
depth_scale = depth_sensor.get_depth_scale()
print("Sügavusparameeter on: " , depth_scale)
```

Joonis 3.6 Sügavussensori käivitamine (vt Lisa 1)

```
# Sügavus pildi töötlemine, massiviks tegemine
colorizer = rs.colorizer()
colorized_depth = np.asanyarray(colorizer.colorize(depth_frame).get_data())
# Sügavusandmete kalkuleerimine
align = rs.align(rs.stream.color)
frameset = align.process(frameset)
# Sügavuspildi kaadrite uuendamine
aligned_depth_frame = frameset.get_depth_frame()
colorized_depth = np.asanyarray(colorizer.colorize(aligned_depth_frame).get_data())
```

Joonis 3.7 Sügavuspildi töötlemine (vt Lisa 1)

3.3.2 Kaamera parameetrite defineerimine ja käivitamine

Välimised parameetrid (Camera extrinsic value) annavad kaamera paiknemise reaalses maailmas, sisemised (Camera intrinsic value) aga konkreetsete suuruste, sealhulgas keskpunkti ja fookuskauguse abil pildi X ja Y koordinaadid (vt Joonis 3.8 ja 3.9). Mõlema abil saadakse moodustada teisendusi vastavalt, kas pikslitest kaamera koordinaatsüsteemi või vastupidi. Autor on kasutanud töös mainitud variantidest esimest.

```
# Kaamera sise,- ja välisparameetride defineerimine
# Intrinsic ja extrinsic
depth_frame = None
color_intrin = None
depth_intrin = None
depth_to_color_extrin = None
```

Joonis 3.8 Kaamera parameetrite tegemine (vt Lisa 1)

```
# Kaamera sise,- ja välisparameetride initsialiseerimine
# Intrinsic ja extrinsic
color_intrin = color_frame.profile.as_video_stream_profile().intrinsics
depth_intrin = aligned_depth_frame.profile.as_video_stream_profile().intrinsics
depth_to_color_extrin = aligned_depth_frame.profile.get_extrinsics_to(color_frame.profile)
```

Joonis 3.9 Kaamera parameetrite initsialiseerimine (vt Lisa 1)

3.4 Maskimine

Maski tegemisel kasutatakse slaiderilt tulevat väärtuste vahemikku, millega sätestatakse kujutatava objekti värvus. Lisaks on pilditöötluse juures oluline selle kvaliteet. Selle saavutamiseks on kaks meetodit (vt Joonis 3.10), mis aitavad pildilt eemaldada väiksemaid avasid, vahesid, pikslivigu või kujutiste väljalõikeid ning muuta seda sujuvamaks. Parema tulemuse saamiseks tehakse mõlema meetodiga kaks iteratsiooni.

Teatud struktuurielement või mask on digitaalne kogumik, mis koosneb ühtedest ja nullidest ning liigub mööda pildil olevaid piksleid. Autori arvates võib seda ette kujutada lihtsa akna või kastina, millega käiakse läbi terve foto. Juhul, kui vähemalt üks nelinurgas olevatest maksimaalväärtustest ehk ühtedest haakub pildil oleva ühega, uuendatakse parasjagu kasti keskpunkti jääva piksli väärtust. Seda ainult eeldusel, et arv on algselt minimaalne ehk null. Vastasel korral, kui haakumist ei toimu või väärtus on juba maksimaalne, muudatusi ei tehta. Sellist protsessi nimetatakse laienemiseks (Dilation). [24] [25]

Võrreldes eeltoodule eristub teine meetod seetõttu, et kõik struktuurielemendi kasti kuuluvad maksimaalväärtused peavad paiknema kohakuti pildi enda ühtedega. Sellises olukorras toimub samuti keskpunkti jääva piksli väärtuse muutmine üheks, kuid olukorra mitte toimumisel aga nulliks. Sellist protsessi nimetatakse vähendamiseks (Erosion). [24] [25]

```
# Maski tegemine eelnevalt defineeritud värvi jaoks (roheline)
# Pildil oleva müra eemaldamine
mask = cv2.inRange(hsv, greenLower, greenUpper)
mask = cv2.erode(mask, None, iterations=2)
mask = cv2.dilate(mask, None, iterations=2)
```

Joonis 3.10 Maski tegemine (vt Lisa 1)

3.5 Kontuuride tegemine

3.5.1 Kontuuride tegemise funktsioon ja nende joonistamine

Objekti tuvastamisel on oluline roll kontuuride joonistamisel ja nende kuvamisel. Autor kasutab selle teostamiseks vastavat funktsiooni Imutils (vt Joonis 3.11). Vaadeldes pildi histogrammi ehk värvide jaotust, leitakse suurim valget värvi ala ning hakatakse seda töötlema (vt Joonis 3.12 ja 3.13). Täpsemalt arvutatakse sellele vastav ringi suurus ning kui saadud raadius on suurem kui kümme ühikut (markeri suurus) kuvatakse see ekraanile. Seda kasutatakse ka sügavuspildilt detaili kauguse saamiseks. Lisaks on katsetatud ka detaili eristamist ligikaudse pindala ja nurkade arvuga, kuid sellega kaasnes pildil liigne müra ning kontuurid ei olnud täpsed – seega ei olnud tulemus piisav.

Saadud pildilt otsitakse kontuuri või ala, mille parameetrid kuuluvad defineeritute hulka. Positiivse olukorra toimumisel on järgnevaks ülesandeks selle ümber ringi joonistamine. Seda tehakse nii värvi- kui ka sügavuspildi juures. Objekti täpsete koordinaatide saamiseks leitakse üles ka selle keskpunkt. Nii pildi analüüsimist, kontuuride joonistamist programmikoodina kui ka lõpliku pildi väljastamist on näha ka joonistel (vt Joonis 3.13 ja 3.14).

```
# Kontuuride tegemine
cnts = cv2.findContours(mask.copy(),
                        cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
center = None
```

Joonis 3.11 Kontuuride tegemise funktsioonid (vt Lisa 1)

```
# Maski pealt suurima kinnise kontuuri otsimine
if len(cnts) > 0:
    c = max(cnts, key = cv2.contourArea)
    ((x, y), radius) = cv2.minEnclosingCircle(c)
    M = cv2.moments(c)
    center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
```

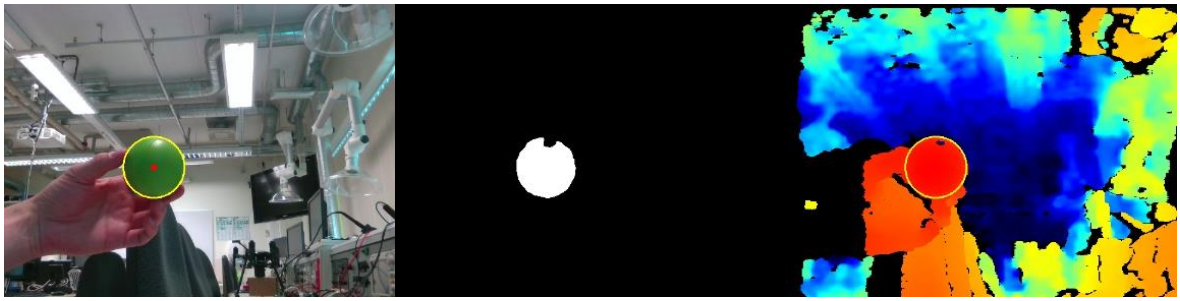
Joonis 3.12 Objetki tuvastasmine (vt Lisa 1)

```
# Kui leitud kontuuri raadius on suurem kui 10 ühikut
# joonistatakse tava- ja sügavuspildile ring ja tema keskkoht
if 10 < radius < 70:
    # Ringi joonestamine
    cv2.circle(frame, (int(x), int(y)), int(radius),(0, 255, 255), 2)
    cv2.circle(colorized_depth, (int(x), int(y)), int(radius),(0, 255, 255), 2)
    # Ringi keskkoha määramine
    cv2.circle(frame, (int(x), int(y)),5,(0, 0, 255), -1)
    cv2.circle(colorized_depth, (int(x), int(y)),5,(0, 0, 255), -1)
```

Joonis 3.13 Kontuuri joonistamine (vt Lisa 1)

3.5.2 Vahekokkuvõte

Selles peatükis toimub objekti nägemine ning tuvastamine tema värvi alusel. Autori arvates ei ole saadud tulemus täiuslik, sest peale langeva valguse liialt suurel tugevusel muutub vaadeldava detaili või markeri värvus (vt Joonis 3.14). See toob kaasa puuduliku või nö auguga kontuuri, mis tööstuslikus kasutamises üldiselt lubatud ei ole. Lisaks võib sellega kaasneda ka saadavate väärate koordinaatide laviin. Probleemi lahenduseks võib tuua, kas slaideri abil parema värvi vahemiku leidmise või ümbritseva keskkonna või ruumi valgustuse reguleerimise.



Joonis 3.14 Lõplik väljakuvatav ekraanitõmmis, kus vasakult paremale lugedes paiknevad vastavalt värvi-, maskitud- ja sügavuspildid.

3.6 Koordinaadid ja nende salvestamine

3.6.1 Distantsi leidmine

Objekti distantsi leidmisel kasutatakse peamiselt sügavuspilti, kus lähemal paiknevad detailid ilmnevad punase ning kaugemad sinise värviga. Samuti on võimalik neid vastupidiseks muuta ehk inverteerida või filtrite abil värvide üleminekuid sujuvamaks korrigeerida. Sealjuures mõõdetakse kaugus vaadeldava markeri keskkohast kaamerani ning seejärel kuvatakse tulemus ekraanile.

Kuna programmikoodis saadavad väärtused on algselt meetrites ja RobotStudio juhtprogrammis kasutatavad pikkusühikud üldiselt millimeetrites, siis sellepärast tehakse ka sellele vastav teisendus (vt Joonis 3.15). Autori arvates on samuti võimalik muuta kaamera enda mõõtühikute suuruseid vastavat skaalat korrigeerides, kuid sellega kaasnes sügavuspildi kvaliteedi märgatav langus ning seetõttu jäädi siiski lihtsa matemaatilise tehte juurde.

```
# Distants
dpt_frame = pipeline.wait_for_frames().get_depth_frame().as_depth_frame()
pixel_distance_in_meters = depth_frame.get_distance(int(x), int(y))
print("Distants(mm):", "{:.1f}".format(pixel_distance_in_meters*1000), "\n")
```

Joonis 3.15 Distantsi saamine (vt Lisa 1)

3.6.2 Kauguste võrdlemine

Kontrollimise mõttes on programmikoodis ilmnevaid kauguste tulemusi võrreldud nii kaamera enda rakenduse – Intel RealSense Viewer kui ka reaalsete distantside väärtustega (vt Tabel 3.1 ja Tabel 3.2). Kahe katse vältel on detaili liigutatud selliselt, et see paikneb igal ajahetkel kaamera ees ning pildi horisontaal- ja vertikaalmõõtmeid arvestades selle keskosas. Kaamera ise paikneb tasapinnalisel ja mitte kõikuval laual ilma kolmjalgse aluseta, sest sellisel juhul on nii pilti edastav seade kui ka vaadeldav objekt samal tasandil. Nii programmikoodis kui ka Intel RealSense Viewer rakenduses on võrdlemise eesmärgil kasutatud identseid seadeid, sealhulgas ka resolutsiooni, milleks on 640 x 480 pikslit (pikkus x laius). Kõrgema ja kvaliteetsema pildi kuvamine kauguste suurusjärkusid ligikaudselt ei muutnud.

Tabelitest ilmneb, et rakenduse ja reaalsete distantside väärtused ühtivad väga suurel määral, kuid programmikoodi tulemused mitte. On näha, et nende erinevus kasvab kauguse suurendes. Põhjuseks võib tuua selle, et kaamerale lähedal olles nähakse vaadeldavat objekti täielikult ning õiges suuruses, sellest aga eemaldudes muutub pildil kuvatav marker samuti väiksemaks kuni nägemisulatusest väljumiseni. Autori arvates on saadud distantsid siiski täpsuse piirides ning antud ülesande edasi lahendamist ei sega.

Tabel 3.1 Mõõdetud kauguste võrdlus esimese katse korral

Katse nr 1.	**Järjekorra nr.**	1	2	3	4	5	6	7	8
	Reaalne distants, mm	300	400	500	600	700	800	900	1000
	Programmikoodi katsed, mm	301	408	509	610	712	815	925	1035
	Intel RealSense Viewer, mm	300	400	500	600	700	800	900	1000

Tabel 3.2 Mõõdetud kauguste võrdlus teise katse korral

Katse nr 2.	**Järjekorra nr.**	9	10	11	12	13	14	15	16
	Reaalne distants, mm	300	400	500	600	700	800	900	1000
	Programmikoodi katsed, mm	299	404	506	610	719	829	937	1037
	Intel RealSense Viewer, mm	300	400	500	600	700	800	910	1000

3.6.3 Koordinaatide leidmine ja salvestamine

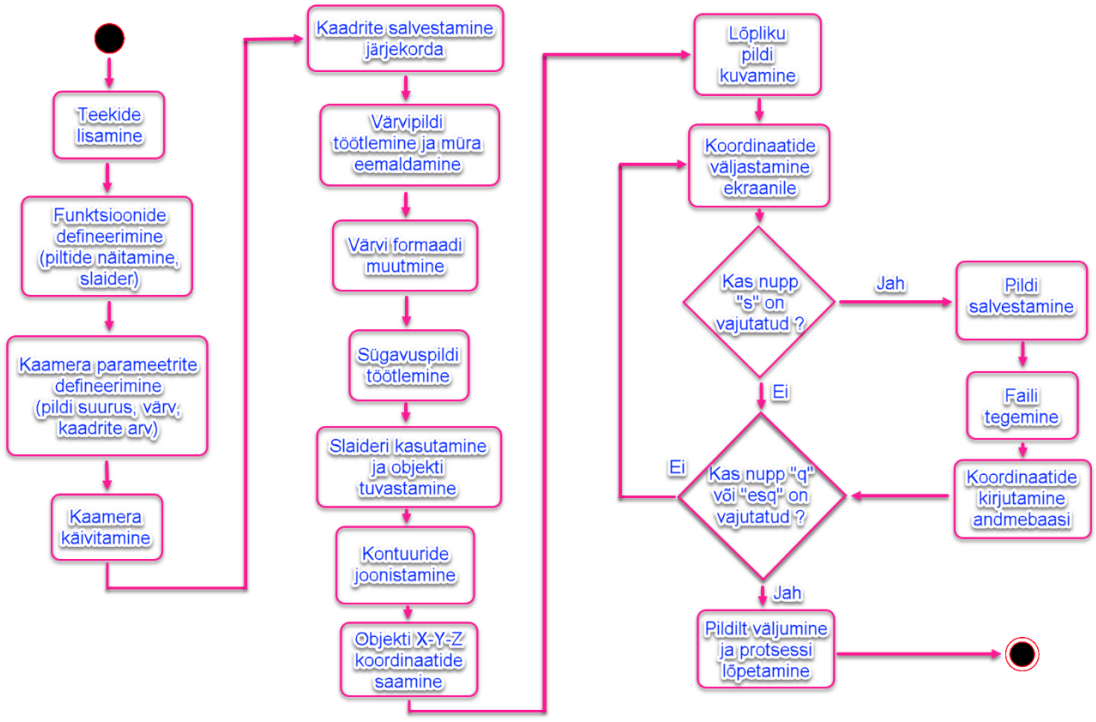
Reaalses maailmas paikneva objekti X ja Y koordinaatide leidmisel on vaja kasutada projektsioone. Teisendus tehakse kaamera distantsi ja sise-, välis- ning sügavusparameetrite abil (vt Joonis 3.16). Vajutades klaviatuuril tähte „s“ salvestatakse hetkeolukorra pilt ja tehakse tekstifail, kuhu salvestatakse eelnevalt määratletud markeri koordinaadid (vt Joonis 3.17). Kui järgevalt detaili asukohta kaamera ees muuta ning uuesti sama nuppu vajutada toimub samalaadne protsess. Sellisel juhul sisestatakse X-Y-Z väärtused ikkagi eelnevalt defineeritud asukohta - ehk uut faili ei tehta ning kõik koordinaadid sisestatakse üksteise alla. Samuti jäävad alles kõik salvestatud pildid.

```
# Objekti koordinaadid
depth_point = rs.rs2_deproject_pixel_to_point(color_intrin,
                                              [x,y],pixel_distance_in_meters*depth_scale)
[x3d,y3d,z3d] = rs.rs2_transform_point_to_point(depth_to_color_extrin, depth_point)
print("Objekti keskpunkti koordinaadid(mm):", "{:.3f}".format(x3d), "{:.3f}".format(y3d))
```

Joonis 3.16 Objekti koordinaadid (vt Lisa 1)

```
# Nupu "s" vajutamisel tehakse pilt ja fail ning salvestatakse objekti koordinaadid
if key == ord("s"):
    cv2.imwrite("Pilt" + str(num) +".jpg", frame)
    num+=1
    f = open("Koordinaadid.txt", "a")
    f.write((str("{:.8f}".format(x3d))) + (" ") + (str("{:.8f}".format(y3d))) + (" ")
            + (str("{:.1f}".format(pixel_distance_in_meters*1000))) + ("\n"))
    f.close()
```

Joonis 3.17 Pildi tegemine ja koordinaatide salvestamine (vt Lisa 1)



Joonis 3.18 Programmi struktuuri kirjeldav voodiagramm

4. ÜHENDUSE LOOMINE

4.1 Võimalused

Markeri X-Y-Z koordinaate sisaldava faili saatmine või üle kandmine mikrokontrollerilt arvutisse võib toimuda mitmel võimalikul moel. Esimesel juhul ühendatakse vastavad seadmed interneti kaablitega ühte kindlasse võrku, milleks näitena võib tuua Tallinna Tehnikaülikooli internetivõrgu. Ühenduse olemasolul ja kindlal toimumisel saab andmeid vastavate otspunktide vahel turvaliselt saata. Teisena võib kasutada lokaalset kohtvõrku (LAN), kus tekitatakse privaatne seadmete vaheline ühendus informatsiooni saatmiseks. Sealjuures on alati oluline meeles pidada turvalisuse põhimõtteid, sest vastavate ühenduste loomisel lubatakse kahe seadme vahel vabalt informatsiooni edastada ning vastu võtta. Seega korrigeeritakse viirusetõrje või muude ühenduste blokeerimise rakenduste seadeid ning võrk võib muutuda kergesti rünnatavaks.

Lisa variandina saadakse katsetada ka olukorda, kus otsest kommunikatsiooni kokkupuudet seadmete vahel ei ole ning vajaliku faili üle kandmine uude keskkonda toimub USB-mälupulga abil. Autori arvates on see kõige kergemini teostatav ja turvalisem meetod, kuid sellega kaasnevad nii töö tegemiseks minev ajakulu kui ka nö vauefekti kadumine. Viimast iseloomustab mõttekäik, et tegevusi, mida on võimalik automatiseerida, teostatakse siiski inimeste poolt käsitsi.

4.2 Socket Server

Serverit kasutatakse andmete ja sõnumite saatmiseks või edasi kandmiseks läbi võrgu ühenduse. Suhtluse toimumisel on peamiselt kaks osapoolt: klient, mis võtab andmeid vastu, ja server, mis neid ühendatud seadmele saadab. Sealjuures võib vastu võtvaid seadmeid olla ka mitmeid. [26]

Autori arvates on mainitud meetod käesoleva töö käigus heaks võimaluseks andmete vahetamiseks. Sealjuures on saatvaks osapooleks koordinaatide failiga mikrokontroller ja vastuvõtvaks personaalne arvuti. Edastamisel järgitakse TCP/IP protokolli, mis toimib nimetatud otspunktide vahel. See on usaldusväärne, sest mõlemad osapooled peavad nõustuma nii saatmise kui vastu võtmise olukorraga, seega saadetavate andmehulkade liikumised on kontrollitavad. Sealjuures on oluline saatmise ja vastuvõtmise järjekorral, sest need ühtivad. [26] Lisaks toimub ühendus ainult juhul, kui nii saatja kui ka vastu võtja teavad sama IP-aadressi nime ja TCP-pordi numbrit. Vastasel korral andmete edastamine võimalik ei ole.

5. ROBOT JA TEMA JUHTIMISPROGRAMM

5.1 ABB YuMi

5.1.1 Andmed

IRB 14400 ehk YuMi on automatiseerimise lihtsustamiseks loodud inimest kaasav robot, mis on võimeline ohu tekkimisel lühikese aja jooksul süsteemi seiskama. Masina modifikatsioone esineb nö ühe kui ka kahe käeliste versioonidena. Sealjuures on mõlemal jäsemel ohutuse tagamiseks korpuse pehmendused ning seitse liikumistelge, mis muudavad töö tegemise kiiremaks ja multifunktsionaalsemaks. Samuti puudub masinat ümbritsev võre, mis tavaliselt turvalisuse mõttes vajalik on. [27] [28]

Autori arvates on YuMi on kergesti kasutatav, sest omab standardset IRC5 kontrolleri süsteemi, mida kasutavad ka teised ABB robotid. Lisaks on masinat võimalik programmeerida ja juhtida ka läbi personaalse arvuti. Samuti on robot võimeline läbi viima korduvaid ja igapäevaseid protsesse, võimaldades töötada 24 tundi ööpäevas. Sealjuures on lisafunktsioonide hulgas haaratsi sisseehitatud kaameral põhinev objekti tuvastus, mida on võimalik kasutada nii detailide liigutamiseks kui ka nende visuaalseks uurimiseks ja sealsete defektide leidmiseks. [28] [29]

Lisaks eelnevale on inimest justkui jäljendaval robotil IP30 kaitseaste, mis peatab töö nii suurte kui ka väiksemate esemete, sealhulgas käte, sõrmede, tööriistade või juhtmete puute toimumisel. Negatiivse poole pealt puudub kaitse vedelike vastu, mis tänapäeva tehnoloogiat vaadeldes üsna tavapäraseks on saanud. Samuti töötab robot peamiselt toatemperatuuri juures või vahemikus 5°C...40°C. [28] [30]

Masina aluse mõõtmed on vastavalt 496 x 399 x 593 mm (pikkus x laius x kõrgus). Sealjuures on maksimaalseks ulatuskauguseks 559 mm [31]. Lisaks on robot kiire, täpne ning kerge kaaluga, kus ühe ja kahe käega versioonid kaaluvad vastavalt 9,5 ja 38 kg [31] [32]. Üldiselt toimub seadme kinnitamine laua külge, kuid roboti eri variante saab võimalusel isegi seina või lae külge paigutada. Laia kasutusvaldkonna tõttu võib seda kasutada nii erinevate kergekaaluliste detailide või kaupade (kuni 500 g) tõstmisel kui ka elektroonika komponentide lisamisel või seadmete komplekteerimisel. [32]

5.1.2 Kasutamine

Autor kasutab valitud robotit seetõttu, et see on eelkõige töö tegemise asukohas olemas ning uute lahenduste testimisel võimeline neid ka teostama. Lisaks võib kasutada ka teisi masinaid, mida on võimalik vastavaks ülesandeks programmeerida. Sinna hulka kuulub ka näiteks ABB IRB 1600, mis on samuti Tallinna Tehnikaülikooli territooriumil olemas. Viimast kasutades tuleb arvestada, et robotid on oma liikumiste ja tehniliste suuruste tõttu erinevad ning seetõttu ei ole võimalik alati kasutada sama programmikoodi. Sealjuures jääb töö loogika ikkagi muutumatuks. Roboti simulatsioonide läbiviimine toimub ohutuse mõttes peamiselt virtuaalses keskkonnas RobotStudio programmi kasutades. Lisaks on kontrolleri RobotWare versioon 6,02.

5.2 RobotStudio

RobotStudio on virtuaalse roboti programmeerimiseks ja simulatsioonide tegemiseks loodud rakendus, mis võimaldab tõsta töökindlust, protsesside produktiivsust ja efektiivsust. Süsteemi programmeerimine toimub üldjuhul arvuti või muu seadme abil ilma, et peaks tootmist välja lülitama. Peale simulatsioonide toimumist on võimalik programmeerimist teostada ka reaalselt, päris robotit kasutades. [33]

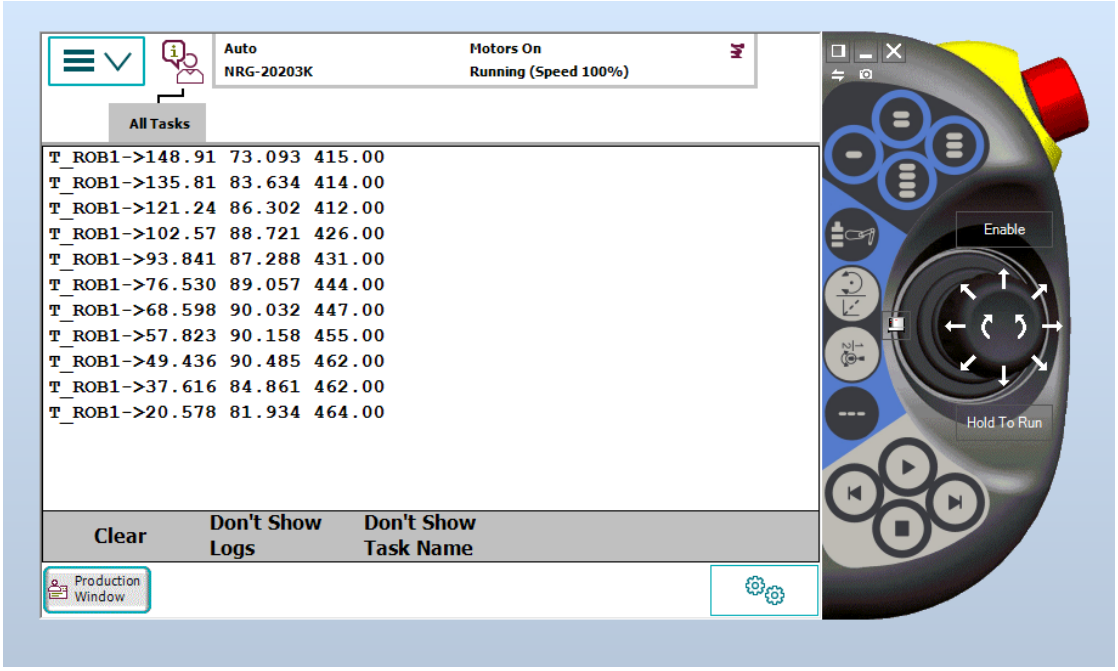
Programmis on palju erinevaid roboteid, mille peamisteks erinevusteks on kasutatav tööala, tõstetav kogumass ja ulatusraadius. Samuti on RobotStudio abil võimalik masinat kergesti seadistada erinevatele tööülesannetele, milleks võivad olla pakkide ja muude detailide tõstmised või liikumised ühest asukohast teise. Autor valis RobotStudio rakenduse eelnevate kasutusoskuste tõttu. Lisaks aitab programm kaasa mõningatele töö ajal tekkivate ohtude, vigade või probleemide ennetamisele. RobotStudio üheks eripäraks on enda poolt valmistatud ruumiliste detailide kasutamise võimalus. Nendeks võivad olla toolid, lauad, alusplaadid, paberiplokid, kapid jne. Lisaks on neid ka lihtne importida. Samuti on võimalik saada igal ajahetkel roboti lülide X-Y-Z koordinaate ja pöördenurkasid ümber telgede. [34]

Lisaks eelnevale on RobotStudios palju erinevaid funktsioone, mille abil toimub kasutaja tegevuste lihtsustamine. Üheks nendest on roboti haaratsi asendi kontroll, mis analüüsib automaatselt lüli asukoha muutust ning maksimaalväärtust ületama hakates katkestab liikumise. Teiseks on kokkupõrke tuvastus, mis üritab programmi toimumise ajal ennetada detailide omavahelisse vastastikmõjusse sattumist ning selle võimalikul juhtumisel lisab veateate. Lisaks kuulub nende hulka programmikoodis olevate vigade otsimine, süsteemi liikumiskiiruse muutmine ja ekraani lindistamise võimalus. [35]

6. ROBOTI JUHTIMINE

6.1.1 Koordinaatide faili lugemine

Esimeseks tegevuseks on arvutis oleva koordinaate hõlmava faili avamine RobotStudio keskkonnas. Seejärel tehakse suur andmemassiiv, kuhu hakatakse andmeid teksti kujul sisse kirjutama. Massiivi suurus sõltub varem salvestatud koordinaatide hulgast. Oluline on mõista, et rohkete andmete läbitöötamine on aja poolest kulukas. Seega on autor valinud suuruse selliselt, et see poleks liialt kohmakas, kuid teisest küljest mahutab piisaval hulgal materjali. Seejärel teisendatakse koordinaatide väärtused teksti kujult tagasi numbriteks (vt Lisa 4). Saadud tulemuse kuvamiseks kasutatakse virtuaalse keskkonna IRC5 kontrolleri FlexPendant pulti, mis on varustatud ekraaniga (vt Joonis 6.1). Kuna koordinaadid on juba algselt õiges formaadis, ei ole vaja neid eraldi roboti koordinaatsüsteemi teisendada.

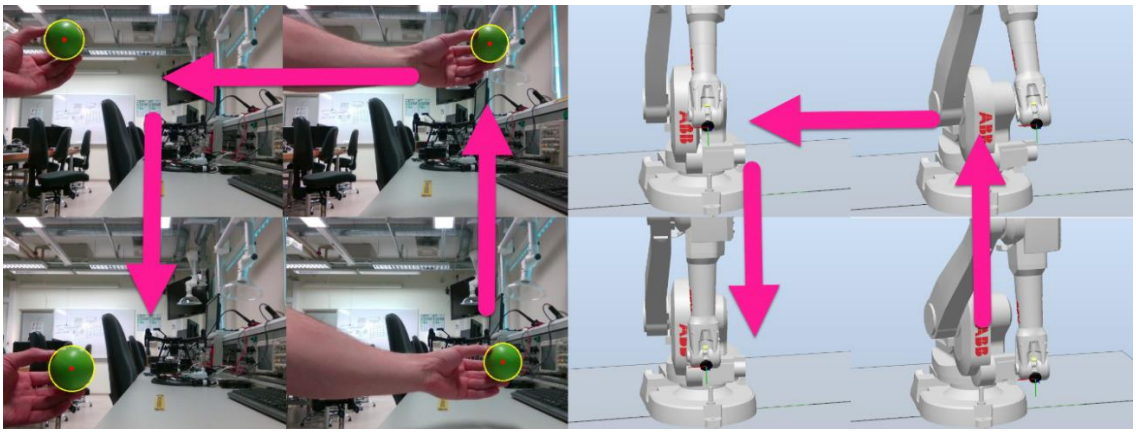


Joonis 6.1 RobotStudio programmis koordinaatide väljastamine

6.1.2 Roboti liikumise genereerimine

Roboti liikuma panemine toimub üldjuhul varem defineeritud sihtpunkte kasutades (vt Joonis 6.2). Selleks koostatakse muutuja, kus paiknevad nii liikumiseks vajalikud koordinaadid kui ka masina lülide orientatsioonid jm parameetrid. Sealjuures asendatakse muutujasse iga haaratsi liigutuse jaoks uuesti massiivis paiknevad X-Y-Z väärtused. Samuti on defineeritud roboti kalibreeritud algasukoht, kuhu minnakse pärast iga koordinaadi vahetust (vt Lisa 4).

Lisaks on võimalik sihtpunktidesse liikumise kiirust kontrollida selle jaoks vastavat parameetrit muutes. Olulisel kohal on haaratsi või tööriista asend või paiknemine liigutatava koordinaatteljestiku suhtes, sest nende ühitimisel liigub haarats igal ajahetkel paralleelselt pinnaga.



Joonis 6.2 Roboti liigutamine IRB 1600 näitel

6.1.3 Vahekokkuvõte

Defineeritud koordinaatide abil on võimalik robotit nendesse liigutada, kuid seda ainult juhul kui masin seda võimaldab. Vastasel korral saadakse veateade ja liikumist ei toimu. Seega tuleb arvestada roboti enda lülide liikumiskauguste ja pöördenurkadega. Lisaks tuleb välja mõelda lahendus, kuidas kõik kaamera abil defineeritud koordinaatide väärtused mahuksid masina liikumisulatusse. Autori arvates on kõige lihtsamaks võimaluseks sellise süsteemi koostamine, kus ilmneb nii kaamera kui ka roboti vahel konkreetne seos. Sellisel juhul tuleb teada saada roboti X-Y-Z telgede suunas liikumise ning kaamera pildi peal paiknevate koordinaatide maksimaalsed väärtused.

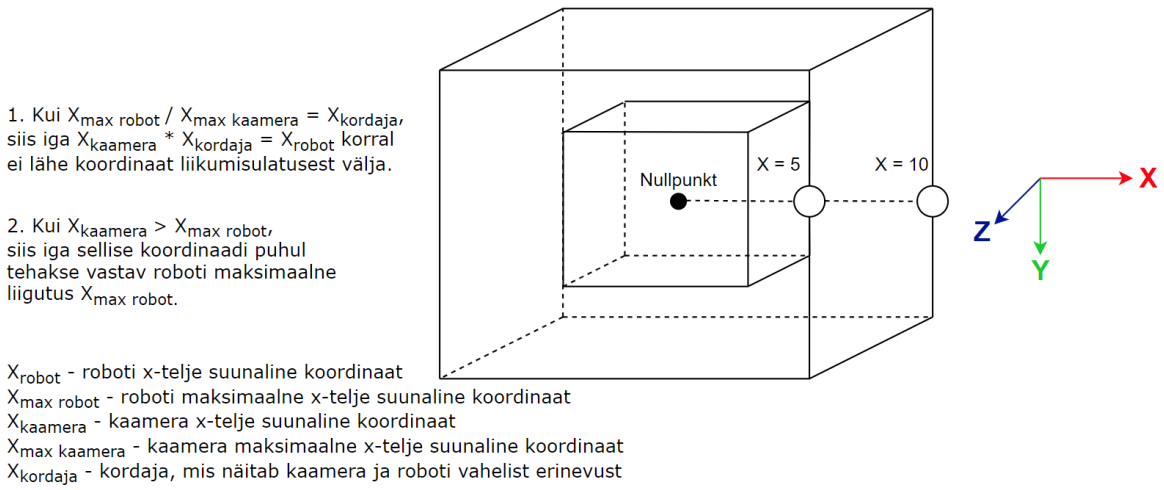
6.1.4 Võimalikud lahendused

Nii programmi RobotStudio kui ka kaamera korral paiknevad punktid selliselt, et keskkohas asub koordinaat, mille väärtused on (0 x 0) ning vastavaid telgi (X-Y-Z) mööda liikudes muutuvad need ühel juhul positiivseteks, teisel negatiivseteks. Seega koordinaatide ühikud ühtivad ja nende lugemine toimub samamoodi. See annab võimaluse kaamera ja roboti maksimaalseid väärtuseid omavahel jagada. Iga telje jaoks saadud kordajaid saab kasutada varasemalt mainitud seose tekitamiseks.

Näitena võib tuua olukorra, kus kaameras ees oleva objekti maksimaalne X-telje suunaline komponent on viis ja robotil kümme ühikut (vt Joonis 6.3). Lihtsa matemaatilise tehte abil saadakse kordaja väärtuseks kaks, mis seletab, et iga markeri koordinaat erineb masina omast vastavalt kaks korda. See toimub juhul, kui vastavate seadmete parameetrid on omavahel lineaarses seoses. Samuti on võimalik ka teiste telgedega sama protsess läbi viia. Igale kaamera koordinaadile leitakse talle vastav roboti X-Y-Z suurus. Autor soovib seejuures korrata süsteemi põhimõtet, et kõik varem defineeritud koordinaadid jääksid roboti liikumisulatusse.

Teisel juhul võib kasutada võrdus- ja võrratusmeetodit, kus esialgselt leitakse sarnaselt eelnevale maksimaalsed telgede suunalised asendid. Eraldi vaadeldakse kahte olukorda. Esmalt olukorda, kus sisse loetavad väärtused jäävad normaalvahemikku. Seejuures toimub liikumine tavapäraselt. Kui aga väärtused jäävad roboti liikumisulatusest välja, arvutatakse need ümber, et tulemus jääks liikumisulatusse. Sellisel juhul tehakse programmis ikkagi vastav roboti maksimaalne liigutus ning piiridest välja ei minda.

Näitena võib tuua olukorra, kus saadud markeri X-telje suunaline koordinaat on vastavalt kümme ja roboti maksimaalne viis ühikut (vt Joonis 6.3). On näha, et kaamera kujutise suurus ei mahu masina liikumisulatusse. Kasutatav tehe näeb välja selline, et iga markeri koordinaadi puhul, mis on suurem, kui teatud konkreetne arv, praegusel hetkel viis ühikut, sooritatakse robotil tema maksimaalne liikumine, milleks on siiski viis ühikut. Seega jäädakse alati liikumisalasse.



Joonis 6.3 Esimesel juhul iseloomustavad suur ja väike kuup vastavalt kaamera ning roboti liikumisulatusi, teisel juhul on olukord vastupidine.

7. EDASISED TEGEVUSED

Algselt toimub koordinaatide leidmine ja salvestamine igal ajahetkel ühe ja sama konkreetse punkti kohta. Hilisemates plaanides on mõistlik aga kasutada kas mitut vaadeldavat objekti või ühte, mis on jaotatud erinevateks osadeks. Viimasest saab luua justkui virtuaalse inimskeleti, kus iga lüli moodustab eraldi vaadeldava tüki. Nendeks võivad olla ranne, küünarnukk ja õlg ning selle teostamine võib toimuda andmete valdkonna närvivõrke kasutades. Sellisel juhul ei pea roboti kõiki lülisid automaatselt koos ühte konkreetsesse punkti liigutama, vaid saab võimalusel lülisid liigutada ka eraldi. Selline teostus muudab süsteemi multifunktsionaalsemaks.

Eeltöödeldud süsteemi eeliseks on suur andmemassiiv, kuhu on koondatud paljudest kihtidest koosnevad protsessid matemaatiliste tehetena, kusjuures igal kihil on erinev ülesanne ja eesmärk. Nendeks võivad olla nii kahe dimensioonilise pildi kui ka reaalajas jooksva video abil inimese, või liikuva objekti tuvastamine ning töötlemine selliselt, et iga skeleti lüli kohta vastab tema reaalse paiknemise tõenäosus. Selle järgi ühendatakse hilisemalt omavahel kokkusobivad punktid ning saadakse skelett. Üheks selliseks närvivõrguks on OpenPose. [36]

Samuti on võimalik üles ehitada süsteem, kus robot enda liikumiste põhiselt õpib [37]. Seda saab kasutada näiteks kaardistamise eesmärgil. Saates robotile erinevaid koordinaate ning proovides masinat liikuma panna ühest asukohast teise, võib toimuda nende meelde jätmine ning salvestamine ruumiliste punktidena faili keskkonda [38]. Selle järgi ehitatakse tema liikumisulatuse vahemik. Parema tulemuse saamiseks ja täpsuse suurendamiseks tuleb tegevust korrata.

KOKKUVÕTE

Töö põhieesmärkideks on robotite õpetamise uurimine ja nende liikuma panemine kasutades selleks masinnägemise abi. Kasutavateks seadmeteks on nii värvi- kui ka sügavuspilti edastav Intel RealSense D415 kaamera ja võimekas NVIDIA Jetson Nano mikrokontroller.

Võrreldes eelnevate samalaadsete projektidega, on töös kasutatav kaamera kõige uuem tehnoloogia, mistõttu on pildi või video peal olevate andmete töötlemine muudetud kasutajale lihtsamaks ja mugavamaks. Samuti on mikrokontroller Jetson informatsiooni töötlemise osas võimekas, sest jõuab seda piisavalt kiirelt hallata. Seega sobivad mõlemad seadmed robootika ja näo- või objekti tuvastamise valdkonda.

Nende omavahelise liidestamise järel toimub varem defineeritud detaili või markeri tuvastamine. Eristamiseks on loodud slaider, mis aitab korrigeerida pildil ilmuvate pikslite värve. Nii on leitud rohelist värvi markeri nägemise asetsemisvahemik, mis sõltub peamiselt peale langeva valguse kogusest ja tugevusest. Eelnevat arvestades ja pildi värvide jaotust vaadeldes joonistatakse markeri ümber selle paiknemise kontuurjooned ning keskpunkt. Järgnevalt leitakse detaili X-Y-Z koordinaadid, mille salvestamiseks luuakse eraldi andmebaas. Sealjuures saadakse ümbritsevate esemete kaugused nii värvi- kui ka sügavuspilti kasutades.

Parema ja kvaliteetsema pildi saavutamiseks kasutatakse mitut erinevat võimalust. Nendest esimene on kaamera kalibreerimine, millega korrigeeritakse selle sisemisi ehk objektiivi ja sensorite parameetreid. Tulemuseks on tekkinud moonutuste vähendamine. Lisaks kasutatakse pildi töötlemise etapis filtreid, kus ühel juhul ühtlustatakse lähedal olevate pikslite intensiivsust ja teisel eemaldatakse väiksemaid avasid või kujutisi. Nende eesmärgid on vastavalt värvi üleminekute sujuvamaks muutmine ning pildi kvaliteedi parandamine.

Mikrokontrolleri ja arvuti vahel loodud ühendus on oluline andmete vahetamiseks ja koordinaatidega sisustatud tekstifaili edastamiseks. Samuti on mõlemal osapoolel eraldi rollid, millest üks peab andmeid ühendatud seadmele saatma ning teine vastu võtma. Sealjuures on ühendus turvaline, sest otspunktide vahel peab nii saatmise kui vastu võtmise olukorra toimumisel olema vastav nõusolek. Lisaks muudab see andmehulkade liikumised kontrollitavateks.

Viimases staadiumis kasutatakse virtuaalset robotkeskkonda, mis võimaldab programmeerimise ja simulatsioonide tegemise abil töökindlust ning protsesside produktiivsust tõsta. Programmi abil toimub saadud fail sisselugemine ja masinale arusaadavaks muutmine. Selleks tehakse suur andmemassiiv, kuhu hakatakse vastavaid andmeid teksti kujul sisse kirjutama. Seejärel muudetakse saadud tekst tagasi numbriteks, eesmärgiga neid sihtpunktide defineerimisel kasutada.

Masina haaratsi liikumisskeemi tekitamiseks luuakse iga koordinaadi muutumise korral uus ruumiline punkt. Lisaks algab ja lõpeb roboti liikumine samast asukohast ehk sihtpunkti jõudes minnakse tagasi algpunkti. Sealjuures on võimalik roboti liikumiskiirust muuta, selle jaoks vastavat parameetrit muutes.

Autor on loonud lahenduse, kuidas kaamera abil koordinaate defineerida ja neid robotile edastada. Liikumine toimub ainult juhul, kui masina lülid seda ise võimaldavad. Vastasel korral saadakse veateade ja tegevust ei toimu. Seega ei ole mõlema osapoole koordinaatteljestike vahel konkreetset seost. Hilisemates plaanides on eelkõige vastavate probleemide lahendamine ning soov süsteemi edasi arendada selliselt, et sinna lisatakse juurde vaadeldavaid markereid. Nende abil saadakse virtuaalne inimskelett täpsema liikumise genereerimiseks. Seejärel võib toimuda roboti õpetamine närvivõrke kasutades, mille järel suureneb multifunktsionaalsus, jättes süsteemi algse põhimõtte muutumatuks.

SUMMARY

The main objectives of this thesis are to study the teaching of robots and to control their movements using machine vision. The author uses an Intel RealSense D415 camera that can transmit both color and depth picture and an NVIDIA Jetson Nano microcontroller.

Compared to previous similar projects the camera, which is used in this work uses the latest technology, which makes the processing of data on an image or video easier and more comfortable for user interface. The Jetson microcontroller is also capable of processing information quickly enough due to its own capability. As a result the mentioned devices are suitable for robotics and a face or an object detection.

After connection of the devices the object identification process occurs. A slider is used to distinguish a prevously defined marker from other visible details. It helps to adjust the colors of the pixels that appear in the image. This supports to find the range of vision of the green marker, which mainly depends on the amount and intensity of the light. In view of the above and observing the color distribution of the image, if the obtained result remains in the respective parameters the center and contour lines are drawn around the marker. The next operation is to find the X-Y-Z coordinates of the detail and store these in a database. The distances of the surrounding objects are obtained using both color and depth images.

There are several ways to get a better and higher quality image. The first of these is the calibration of the camera, which adjusts its internal parameters included the parameters of the lens and other sensors. The result of this is a reduction in distortions. In addition some filters are used in the image processing phase, where in one case the intensity of nearby pixels are equalized and in other case smaller apertures or openings in the contours of images and pixel errors are removed. The objectives of these methods are to smooth the color transitions and improve image quality.

The connection between the microcontroller and computer is important for data exchange, more specifically to transmit a text file with coordinates. Both devices need to have their own role in this system, where one has to send the data to the connected device and the other receive all of it. The connection is secure, because there must be agreement between the endpoints in both ways for sending and receiving situations. In addition it also makes the movement of the datasets controllable.

In the last stage a virtual robot environment is used, which allows to increase reliability and process productivity through programming and simulation performing. With this programm it is possible to read the received file and make it understandable for the machine itself.

For this purpose a large data array is created in which the corresponding data will be written in the form of text. Nextly the resulting text need to be converted back to numbers for defining movement destinations. A new targetpoint is created each time the coordinate changes. In addition the robot starts and stops moving from the same location, in example when it reaches the destination it goes back to the starting point. It is also possible to change the speed of the movement by changing the specific robot parameter.

The author has created a solution for defining coordinates using a camera and transmitting them to the robot. The movement only takes place if the machine's links allow it, otherwise an error message is received and no action occurs. Therefore, there is no specific link between the coordinate systems of a camera and a robot. Further plans are to solve the current problems and to develop the system by adding more visible markers. These will be used to get the virtual human skeleton for more accurate movement. The robot can then be taught using neural networks by which the multifunctionality of the system increases leaving the original principle unchanged.

KASUTATUD KIRJANDUS

[1] M. Weinberger, „The rise and fall of Kinect: Why Microsoft gave up on its most promising product,“ Business Insider, 3 Jaanuar 2018. [Võrgumaterjal]. Saadaval: https://finance.yahoo.com/news/downfall-kinect-why-microsoft-gave-183900710.html. [Kasutatud 18 Mai 2020].

[2] „Intel RealSense Depth Camera D415,“ Intel Corporation, [Võrgumaterjal]. Available: https://www.intelrealsense.com/depth-camera-d415/. [Kasutatud 20 Märts 2020].

[3] „Intel® RealSense™ Depth Camera D415 Store,“ Intel Corporation, [Võrgumaterjal]. Saadaval: https://store.intelrealsense.com/buy-intel-realsense-depth-camera-d415.html. [Kasutatud 24 Märts 2020].

[4] „Intel RealSense D400 Series Product Family Datasheet,“ Jaanuar 2019. [Võrgumaterjal]. Saadaval: https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/Intel-RealSense-D400-Series-Datasheet.pdf. [Kasutatud 20 Märts 2020].

[5] „What Is Camera Calibration?,“ The MathWorks, [Võrgumaterjal]. Saadaval: https://www.mathworks.com/help/vision/ug/camera-calibration.html. [Kasutatud 22 Aprill 2020].

[6] „Camera Calibration,“ doxygen, 29 August 2018. [Võrgumaterjal]. Saadaval: https://docs.opencv.org/3.4.3/dc/dbb/tutorial_py_calibration.html. [Kasutatud 22 Aprill 2020].

[7] „Intel RealSense SDK,“ Intel Corporation, 2018. [Võrgumaterjal]. Saadaval: https://github.com/IntelRealSense/librealsense. [Kasutatud 8 Aprill 2020].

[8] „pyrealsense2,“ Intel Realsense Team, [Võrgumaterjal]. Saadaval: https://intelrealsense.github.io/librealsense/python_docs/_generated/pyrealsense2.html. [Kasutatud 30 Aprill 2020].

[9] „NumPy,“ NumPy developers, 2020. [Võrgumaterjal]. Saadaval: https://numpy.org/. [Kasutatud 8 Aprill 2020].

[10] „NumPy in Python | Set 1 (Introduction),“ GeeksforGeeks, 5th & 6th Floor, Royal Kapsons, A- 118, Sector- 136, Noida, Uttar Pradesh (201305), [Võrgumaterjal]. Saadaval: https://www.geeksforgeeks.org/numpy-in-python-set-1-introduction/. [Kasutatud 30 Aprill 2020].

[11] A. Mordvintsev ja A. K, „OpenCV-Python Tutorials,“ 2013. [Võrgumaterjal]. Saadaval: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html. [Kasutatud 8 Aprill 2020].

[12] „OpenCV – Overview,“ GeeksforGeeks, [Võrgumaterjal]. Saadaval: https://www.geeksforgeeks.org/opencv-overview/. [Kasutatud 30 Aprill 2020].

[13] A. Rosenbrock, „Imutils,“ 18 august 2019. [Võrgumaterjal]. Saadaval: https://github.com/jrosebr1/imutils. [Kasutatud 8 Aprill 2020].

[14] „Frame Buffering Management in RealSense SDK 2.0,“ Intel Corporation, [Võrgumaterjal]. Saadaval: https://github.com/IntelRealSense/librealsense/wiki/Frame-Buffering-Management-in-RealSense-SDK-2.0. [Kasutatud 9 Aprill 2020].

[15] „NVIDIA Jetson Nano Developer Kit,“ NVIDIA Corporation, Märts 2019. [Võrgumaterjal]. Saadaval: https://siliconhighway.com/wp-content/gallery/jetson-nano-devkit-datasheet-936542-US-hr.pdf. [Kasutatud 24 Märts 2020].

[16] „NVIDIA Jetson Nano Developer kit description,“ Seeed Technology Co.,Ltd., [Võrgumaterjal]. Saadaval: https://www.seeedstudio.com/NVIDIAr-Jetson-Nanotm-Developer-Kit-p-2916.html. [Kasutatud 24 Märts 2020].

[17] „Jetson Nano - Use More Power!,“ JetsonHacks, 10 Aprill 2019. [Võrgumaterjal]. Saadaval: https://www.jetsonhacks.com/2019/04/10/jetson-nano-use-more-power/. [Kasutatud 24 Märts 2020].

[18] „Getting Started With Jetson Nano Developer Kit,“ NVIDIA Corporation, 2020. [Võrgumaterjal]. Saadaval: https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit. [Kasutatud 24 Märts 2020].

[19] R. Böhringer, „HSV Color Space,“ 16 Aprill 2019. [Võrgumaterjal]. Saadaval: https://www.ronja-tutorials.com/2019/04/16/hsv-colorspace.html. [Kasutatud 30 Aprill 2020].

[20] „Image Processing with Python,“ The Carpentries, Data Carpentry, 16 Veebruar 2020. [Võrgumaterjal]. Saadaval: https://datacarpentry.org/image-processing/06-blurring/. [Kasutatud 27 Märts 2020].

[21] R. Collins, „Lecture 4: Smoothing,“ [Võrgumaterjal]. Saadaval: http://www.cse.psu.edu/~rtc12/CSE486/lecture04.pdf. [Kasutatud 21 Aprill 2020].

[22] „rs-align-advanced,“ Intel Corporation, [Võrgumaterjal]. Saadaval: https://dev.intelrealsense.com/docs/rs-align-advanced. [Kasutatud 1 Mai 2020].

[23] A. Grunnet-Jepsen, J. N. Sweetser ja J. Woodfill, „Tuning depth cameras for best performance,“ [Võrgumaterjal]. Saadaval: https://dev.intelrealsense.com/docs/tuning-depth-cameras-for-best-performance. [Kasutatud 1 Mai 2020].

[24] N. Efford, „Digital Image Processing: A Practical Introduction Using JavaTM, Chapter 11 "Morphological Image Processing",“ 2000. [Võrgumaterjal]. Saadaval: https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm. [Kasutatud 21 Aprill 2020].

[25] „Types of Morphological Operations,“ The MathWorks, [Võrgumaterjal]. Saadaval: https://www.mathworks.com/help/images/morphological-dilation-and-erosion.html. [Kasutatud 21 Aprill 2020].

[26] N. Jennings, „Socket Programming in Python (Guide),“ Real Python, [Võrgumaterjal]. Saadaval: https://realpython.com/python-sockets/. [Kasutatud 22 Aprill 2020].

[27] „YuMi® - IRB 14000 | Collaborative Robot,“ ABB, 2020. [Võrgumaterjal]. Saadaval: https://new.abb.com/products/robotics/industrial-robots/irb-14000-yumi. [Kasutatud 19 Aprill 2020].

[28] „YuMi® – IRB 14000 Overview,“ ABB, 6 Detsember 2018. [Võrgumaterjal]. Saadaval: https://search.abb.com/library/Download.aspx?DocumentID=9AKK106354A3256&LanguageCode=en&DocumentPartId=&Action=Launch. [Kasutatud 19 Aprill 2020].

[29] „ABB plaanib tuua haiglatesse mobiilsed laborirobotid,“ AM, Arvutimaailm, 10 Oktoober 2019. [Võrgumaterjal]. Saadaval: https://www.am.ee/ABB-YuMi. [Kasutatud 20 Aprill 2020].

[30] „IP Rating Chart,“ DSM&T, [Võrgumaterjal]. Saadaval: http://www.dsmt.com/resources/ip-rating-chart/. [Kasutatud 19 Aprill 2020].

[31] „YuMi® creating an automated future together.," ABB, Mai 2019. [Võrgumaterjal]. Saadaval: https://search.abb.com/library/Download.aspx?DocumentID=9AKK106354A3254&LanguageCode=en&DocumentPartId=&Action=Launch. [Kasutatud 19 Aprill 2020].

[32] „IRB 14050 Single-arm YuMi® Collaborative Robot," ABB, [Võrgumaterjal]. Saadaval: https://new.abb.com/products/robotics/industrial-robots/irb-14050-single-arm-yumi. [Kasutatud 19 Aprill 2020].

[33] M. Segerstrom, „What is RobotStudio? I bet you're guessing it has something to do with robots.," 24 Aprill 2013. [Võrgumaterjal]. Saadaval: https://www.abb-conversations.com/2013/04/what-is-robotstudio-something-to-do-with-robots/. [Kasutatud 16 Aprill 2020].

[34] P. Neto, „A Guide for ABB RobotStudio," Jaanuar 2014. [Võrgumaterjal]. Saadaval: http://www2.dem.uc.pt/pedro.neto/PUB/BC/M_1.pdf. [Kasutatud 16 Aprill 2020].

[35] L. Eitel, „RobotStudio software from ABB optimizes robot setups (with videos)," 13 Aprill 2017. [Võrgumaterjal]. Saadaval: https://www.therobotreport.com/abb-robotstudio-software-optimizes-robot-setups-with-videos/. [Kasutatud 16 Aprill 2020].

[36] V. Gupta, „Deep Learning based Human Pose Estimation using OpenCV (C++ / Python)," BIG VISION LLC, 29 Mai 2018. [Võrgumaterjal]. Saadaval: https://www.learnopencv.com/deep-learning-based-human-pose-estimation-using-opencv-cpp-python/. [Kasutatud 17 Mai 2020].

[37] B. Li, T. Lu, X. Li, Y. Cai ja S. Wang, „An Automatic Robot Skills Learning System from Robot's Real-World Demonstrations," 12 September 2019. [Võrgumaterjal]. Saadaval: https://ieeexplore.ieee.org/document/8833143. [Kasutatud 18 Mai 2020].

[38] P.-J. Hwang, C.-C. Hsu ja W.-Y. Wang, „Development of a Mimic Robot: Learning from Human Demonstration to Manipulate a Coffee Maker as an Example," Department of Electrical Engineering, National Taiwan Normal University, 14 November 2019. [Võrgumaterjal]. Saadaval: https://ieeexplore.ieee.org/document/8901025. [Kasutatud 18 Mai 2020].

[39] A. Rosebrock, „Ball Tracking with OpenCV," 14 September 2015. [Võrgumaterjal]. Saadaval: https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/. [Kasutatud 2020 Aprill 22].

[40] „Open CV viewer example,“ Intel Corporation, oktoober 2018. [Võrgumaterjal]. Saadaval: https://github.com/IntelRealSense/librealsense/blob/master/wrappers/python/examples/opencv_viewer_example.py. [Kasutatud 7 Aprill 2020].

[41] D. Bader, „Socket Programming in Python, Echo-server,“ 10 Oktoober 2018. [Võrgumaterjal]. Saadaval: https://github.com/realpython/materials/blob/master/python-sockets-tutorial/echo-server.py. [Kasutatud 22 Aprill 2020].

[42] D. Bader, „Socket Programming in Python, Echo-client,“ 10 Oktoober 2018. [Võrgumaterjal]. Saadaval: https://github.com/realpython/materials/blob/master/python-sockets-tutorial/echo-client.py. [Kasutatud 22 Aprill 2020].

LISAD

Lisa 1 Objekti tuvastamine ja koordinaatide salvestamine [13] [39] [40]

```
# Nimetus: Objetki tuvastamine, koordinaatide kirjutamine
# Kirjeldus:
# Avatakse kaamera ning seejärel tuvastatakse nähtavaid objekte. Kui leitakse
sobiv/parameetritesse mahtuv, joonistatakse sellele ümber ring.
# Vajutusel nuppu "s" salvestatakse pilt ning kirjutatakse selle objekti koordinaadid teksti faili.
# Kuupäev: 11.04.2020

# Teekide lisamine
import pyrealsense2 as rs # Kaamera käivitamine
import numpy as np # Andme massiivide kasutamine
import cv2 # Pildi töötlemine
import imutils # Kontuuride tegemine

key = cv2.waitKey(1) # Nupu defineerimine
num = 1 # Salvestatud piltide number algab 1-st

# Piltide koosnäitamise funktsioon
def stackImages(scale, imgArray):
   rows = len(imgArray)
   cols = len(imgArray[0])
   rowsAvailable = isinstance(imgArray[0], list)
   width = imgArray[0][0].shape[1]
   height = imgArray[0][0].shape[0]
   if rowsAvailable:
      for x in range ( 0, rows):
         for y in range(0, cols):
            if imgArray[x][y].shape[:2] == imgArray[0][0].shape [:2]:
               imgArray[x][y] = cv2.resize(imgArray[x][y], (0, 0), None, scale, scale)
            else:
               imgArray[x][y] = cv2.resize(imgArray[x][y], (imgArray[0][0].shape[1],
imgArray[0][0].shape[0]), None, scale, scale)
            if len(imgArray[x][y].shape) == 2: imgArray[x][y]= cv2.cvtColor( imgArray[x][y],
cv2.COLOR_GRAY2BGR)
```

```
        imageBlank = np.zeros((height, width, 3), np.uint8)
        hor = [imageBlank]*rows
        hor_con = [imageBlank]*rows
        for x in range(0, rows):
            hor[x] = np.hstack(imgArray[x])
        ver = np.vstack(hor)
    else:
        for x in range(0, rows):
            if imgArray[x].shape[:2] == imgArray[0].shape[:2]:
                imgArray[x] = cv2.resize(imgArray[x], (0, 0), None, scale, scale)
            else:
                imgArray[x] = cv2.resize(imgArray[x], (imgArray[0].shape[1], imgArray[0].shape[0]),
None,scale, scale)
            if len(imgArray[x].shape) == 2: imgArray[x] = cv2.cvtColor(imgArray[x],
cv2.COLOR_GRAY2BGR)
        hor= np.hstack(imgArray)
        ver = hor
    return ver

# Slaideri tegemine värvi korrigeerimiseks
def empty(a):
    pass
cv2.namedWindow("Slaider")
cv2.resizeWindow("Slaider", 300, 400)
cv2.createTrackbar("L-H", "Slaider", 29, 255, empty)
cv2.createTrackbar("L-S", "Slaider", 86, 255, empty)
cv2.createTrackbar("L-V", "Slaider", 6, 255, empty)
cv2.createTrackbar("U-H", "Slaider", 64, 255, empty)
cv2.createTrackbar("U-S", "Slaider", 255, 255, empty)
cv2.createTrackbar("U-V", "Slaider", 255, 255, empty)
#greenLower = (29, 86, 6)
#greenUpper = (64, 255, 255)

# Kaamera seadete(suurus, värv, kaadrite arv) defineerimine
pipeline = rs.pipeline()
```

```
config = rs.config()
config.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)
config.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)

# Kaamera käivitus
profile = pipeline.start(config)

# Kaamera sise- ja välisparameetride defineerimine
# Intrinsic ja extrinsic
depth_frame = None
color_intrin = None
depth_intrin = None
depth_to_color_extrin = None

# Sügavussensori käivitamine
depth_sensor = profile.get_device().first_depth_sensor()
depth_scale = depth_sensor.get_depth_scale()
print("Sügavusparameeter on: " , depth_scale)

try:
   while True:
      # Kaadrite järjekorda salvestamine
      # Värvi- ja sügavuspilt
      frameset = pipeline.wait_for_frames()
      color_frame = frameset.get_color_frame()
      depth_frame = frameset.get_depth_frame()
      if not depth_frame or not color_frame:
         continue

      # Värvipildi töötlemine, massiviks tegemine
      color_image = np.asanyarray(color_frame.get_data())
      frame = color_image.copy()
      blurred = cv2.GaussianBlur(frame, (11, 11), 0)
      hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
```

```
# Sügavus pildi töötlemine, massiviks tegemine
colorizer = rs.colorizer()
colorized_depth = np.asanyarray(colorizer.colorize(depth_frame).get_data())
# Sügavusandmete kalkuleerimine
align = rs.align(rs.stream.color)
frameset = align.process(frameset)
# Sügavuspildi kaadrite uuendamine
aligned_depth_frame = frameset.get_depth_frame()
colorized_depth = np.asanyarray(colorizer.colorize(aligned_depth_frame).get_data())

# Kaamera sise- ja välisparameetride initsialiseerimine
# Intrinsic ja extrinsic
color_intrin = color_frame.profile.as_video_stream_profile().intrinsics
depth_intrin = aligned_depth_frame.profile.as_video_stream_profile().intrinsics
depth_to_color_extrin = aligned_depth_frame.profile.get_extrinsics_to(color_frame.profile)

# Initsialiseerime värvislaideri, anname slaideri ridadele väärtused
l_h = cv2.getTrackbarPos("L-H", "Slaider")
l_s = cv2.getTrackbarPos("L-S", "Slaider")
l_v = cv2.getTrackbarPos("L-V", "Slaider")
u_h = cv2.getTrackbarPos("U-H", "Slaider")
u_s = cv2.getTrackbarPos("U-S", "Slaider")
u_v = cv2.getTrackbarPos("U-V", "Slaider")
greenLower = (l_h, l_s, l_v)
greenUpper = (u_h, u_s, u_v)

# Maski tegemine eelnevalt defineeritud värvi jaoks (roheline)
# Pildil oleva müra eemaldamine
mask = cv2.inRange(hsv, greenLower, greenUpper)
mask = cv2.erode(mask, None, iterations=2)
mask = cv2.dilate(mask, None, iterations=2)

# Kontuuride tegemine
cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
```

```
        center = None

        # Maski pealt suurima kinnise kontuuri otsimine
        if len(cnts) > 0:
            c = max(cnts, key = cv2.contourArea)
            ((x, y), radius) = cv2.minEnclosingCircle(c)
            M = cv2.moments(c)
            center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
            # Kui leitud kontuuri raadius on suurem kui 10 ühikut, joonistatakse tava- ja sügavuspildile
ring ja tema keskkoht
            if 10 < radius < 70:
                # Ringi joonestamine
                cv2.circle(frame, (int(x), int(y)), int(radius),(0, 255, 255), 2)
                cv2.circle(colorized_depth, (int(x), int(y)), int(radius),(0, 255, 255), 2)
                # Ringi keskkoha määramine
                cv2.circle(frame, (int(x), int(y)),5,(0, 0, 255), -1)
                cv2.circle(colorized_depth, (int(x), int(y)),5,(0, 0, 255), -1)

                # Distants
                dpt_frame = pipeline.wait_for_frames().get_depth_frame().as_depth_frame()
                pixel_distance_in_meters = depth_frame.get_distance(int(x), int(y))
                print("Distants(mm):", "{:.2f}".format(pixel_distance_in_meters*1000), "\n")

                # Objekti koordinaadid
                depth_point =
rs.rs2_deproject_pixel_to_point(color_intrin,[x,y],pixel_distance_in_meters*depth_scale)
                [x3d,y3d,z3d] = rs.rs2_transform_point_to_point(depth_to_color_extrin, depth_point)
                # Vahemik -10...10 (ei ole kaasaarvatud)
                if -10 < (x3d *1000000) < 10:
                    print("Objekti keskpunkti koordinaadid(mm):", "{:.3f}".format(x3d *1000000)+ "0",
"{:.3f}".format(y3d *1000000))
                elif -10 < (y3d *1000000) < 10:
                    print("Objekti keskpunkti koordinaadid(mm):", "{:.3f}".format(x3d *1000000),
"{:.3f}".format(y3d *1000000)+ "0")
                elif -10 < (x3d *1000000) < 10 & -10 < (y3d *1000000) < 10:
```

```
            print("Objekti keskpunkti koordinaadid(mm):", "{:.3f}".format(x3d *1000000)+ "0",
"{:.3f}".format(y3d *1000000)+ "0")
        # Vahemik <= -100 või >= 100 (kaasaarvatud)
        elif (x3d *1000000) <= -100 or (x3d *1000000) >= 100:
            print("Objekti keskpunkti koordinaadid(mm):", "{:.2f}".format(x3d *1000000),
"{:.3f}".format(y3d *1000000))
        elif (y3d *1000000) <= -100 or (y3d *1000000) >= 100:
            print("Objekti keskpunkti koordinaadid(mm):", "{:.3f}".format(x3d *1000000),
"{:.2f}".format(y3d *1000000))
        elif (x3d *1000000) <= -100 or (x3d *1000000) >= 100 and (y3d *1000000) <= -100 or
(y3d *1000000) >= 100:
            print("Objekti keskpunkti koordinaadid(mm):", "{:.2f}".format(x3d *100000),
"{:.2f}".format(y3d *100000))
        # Vahemik -10...-100 ja 10...100 (-10 ja 10 kaasaarvatud)
        else:
            print("Objekti keskpunkti koordinaadid(mm):", "{:.3f}".format(x3d *1000000),
"{:.3f}".format(y3d *1000000))



        # Nupu "s" vajutamisel tehakse pilt ja fail ning salvestatakse objekti koordinaadid
        if key == ord("s"):
            cv2.imwrite("Pilt" + str(num) +".jpg", frame)
            num+=1
            f = open("Koordinaadid.txt", "a")
            # Vahemik -10...10 (ei ole kaasaarvatud)
            if -10 < (x3d *1000000) < 10:
                f.write((str("{:.3f}".format(x3d*1000000))) + "0" + (" ") + (str("{:.3f}".format(y3d
*1000000))) + (" ") + (str("{:.2f}".format(pixel_distance_in_meters*1000))) + ("\n"))
            elif -10 < (y3d *1000000) < 10:
                f.write((str("{:.3f}".format(x3d*1000000))) + (" ") + (str("{:.3f}".format(y3d
*1000000))) + "0" + (" ") + (str("{:.2f}".format(pixel_distance_in_meters*1000))) + ("\n"))
            elif -10 < (x3d *1000000) < 10 & -10 < (y3d *1000000) < 10:
                f.write((str("{:.3f}".format(x3d*1000000))) + "0" + (" ") + (str("{:.3f}".format(y3d
*1000000))) + "0" + (" ") + (str("{:.2f}".format(pixel_distance_in_meters*1000))) + ("\n"))
            # Vahemik <= -100 või >= 100 (kaasaarvatud)
```

```
                elif (x3d *1000000) <= -100 or (x3d *1000000) >= 100:
                    f.write((str("{:.2f}".format(x3d*1000000))) + (" ") + (str("{:.3f}".format(y3d
*1000000))) + (" ") + (str("{:.2f}".format(pixel_distance_in_meters*1000))) + ("\n"))
                elif (y3d *1000000) <= -100 or (y3d *1000000) >= 100:
                    f.write((str("{:.3f}".format(x3d*1000000))) + (" ") + (str("{:.2f}".format(y3d
*1000000))) + (" ") + (str("{:.2f}".format(pixel_distance_in_meters*1000))) + ("\n"))
                elif (x3d *1000000) <= -100 or (x3d *1000000) >= 100 and (y3d *1000000) <= -100 or
(y3d *1000000) >= 100:
                    f.write((str("{:.2f}".format(x3d*1000000))) + (" ") + (str("{:.2f}".format(y3d
*1000000))) + (" ") + (str("{:.2f}".format(pixel_distance_in_meters*1000))) + ("\n"))
                # Vahemik -10...-100 ja 10...100 (-10 ja 10 kaasaarvatud)
                else:
                    f.write((str("{:.3f}".format(x3d*1000000))) + (" ") + (str("{:.3f}".format(y3d
*1000000))) + (" ") + (str("{:.2f}".format(pixel_distance_in_meters*1000))) + ("\n"))
                f.close()

        # Lõpliku pildi kuvamine
        imgStack = stackImages(0.6,([frame, mask, colorized_depth]))
        cv2.imshow("Frame, Mask, Depth", imgStack)
        key = cv2.waitKey(1)

        #Pildilt väljumiseks tuleb vajutada "q" tähte või "esc" nuppu
        if key & 0xFF == ord("q") or key == 27:
            cv2.destroyAllWindows()
            break
finally:
    pipeline.stop()
```

Lisa 2 Mikrokontrolleri ja arvuti vaheline ühendus, server [41]

```
# Nimetus: Socket server(server)
# Kirjeldus: Tekitatakse ühendus serveri ja kliendi vahel andmete saatmiseks.
# Kuupäev: 05.05.2020

# Ühenduse loomine
import socket

HOST = "172.20.82.100"  # Saatja IP aadress
PORT = 65432  # Vabalt valitud pordi nr (peab ühtima klient/serveri vahel)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
   s.bind((HOST, PORT))
   s.listen()
   print("Saatja nimi: ", HOST)
   print("Oodatakse sissetulevaid ühendusi ..")
   conn, addr = s.accept()
   with conn:
      print("\n" + "Serveriga on ühendatud: ", addr)
      while True:
         filename = input(str("\n" + "Sisesta saadetava faili nimi: "))
         # Faili avamine ja kodeerimine
         file = open(filename, "rb")
         file_data = file.read(8192)
         # Faili saatmine
         conn.send(file_data)
         print("\n" + "Fail on saadetud.")
         if not file_data:
            break
```

Lisa 3 Mikrokontrolleri ja arvuti vaheline ühendus, klient [42]

```
# Nimetus: Socket server(klient)
# Kirjeldus: Tekitatakse ühendus serveri ja kliendi vahel andmete vastuvõtmiseks.
# Kuupäev: 05.05.2020

# Ühenduse loomine
import socket

HOST = "172.20.82.138"  # Saaja IP aadress
PORT = 65432  # Vabalt valitud pordi nr (peab ühtima klient/serveri vahel)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
   s.connect((HOST, PORT))
   print("\n" + "Ühendus on loodud.")
   filename = input(str("\n" + "Sisesta vastuvõetava faili nimi: "))
   # Faili avamine ja kodeerimine
   file = open(filename, "wb")
   # Faili vastuvõtmine
   file_data = s.recv(8192)
   file.write(file_data)
   file.close
print("\n" + "Fail on vastuvõetud.")
```

```
MODULE Module1
! Nimetus: Koordinaatide sisselugemine ja liikumise genereerimine
! Kirjeldus: Kõigepealt toimub Desktopil asetseva faili avamine ning koordinaatide
lugemine massiivi. Seejärel toimub teisendus tekstilt numbrite kujule. Viimasena
sisestatakse vastavad X-Y-Z parameetrid muutujasse ning liigutatakse robotit uude
asukohta.
! Kuupäev: 10.05.2020

    !Defineerimine
    VAR string Array{80}; !Andmemassiv
    VAR num stringsAmount; !Sõnadearv
    VAR string Data; !Andmed tektsti kujul
    VAR num i; !Töötlemine
    VAR num j; !Töötlemine
    VAR iodev file; !Failiga töötlemine
    PERS bool EOF; !Failiga töötlemine
    VAR string sx; !X-koordinaat tekstina
    VAR string sy; !Y-koordinaat tekstina
    VAR string sz; !Z-koordinaat tekstina
    VAR bool ok; !Teisendus tekstist numbri kujule
    PERS num ex; !X-koordinaat numbrina
    PERS num ey; !Y-koordinaat numbrina
    PERS num ez; !Z-koordinaat numbrina

    !Roboti algpositsioon
    CONST jointtarget calib_pos := [[0, 0, 0, 0, 0, 0], [0, 9E9, 9E9, 9E9, 9E9, 9E9]];
    !Roboti liikumispositsioon
    PERS robtarget Target_a:=[[49.436,90.485,462.00],[1,0,0,0],[0,-
1,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    PROC main()
        CountStringsInFile; !Massiivi jaoks vajaliku informatsiooni saamine
        WriteDataAndMove; !Massiivi tekitamine, koordinaatide lugemine, liikumine
    ENDPROC

    PROC CountStringsInFile()
        Open "C:/Users/jkikoj/Desktop/Koordinaadid.txt",file\Read;
        Rewind file;
        stringsAmount:=0;
        EOF:=FALSE;

        WHILE EOF=FALSE AND Data<>"EOF" DO
            Data:=ReadStr(file);
            Incr stringsAmount;
        ENDWHILE

        Decr stringsAmount;
        Close file;
    ENDPROC
```

```
  PROC WriteDataAndMove()
    Open "C:/Users/jkikoj/Desktop/Koordinaadid.txt",file\Read;
    Rewind file;
    EOF:=FALSE;
    i:=1;
    j:=1;
    WHILE EOF=FALSE AND i<stringsAmount+1 DO
      Data:=ReadStr(file);
      Array{i}:=Data;
      WHILE j=i DO
        !Hetke koordinaatide väljastus
        TPWrite Data;

        !Massiivist koordinaatide lugemine tekstina
        sx:= StrPart(Array{j}, 1, 6);
        sy:= StrPart (Array{j}, 8, 6);
        sz:= StrPart (Array{j}, 15, 6);

        !Tekstist numbrite saamine
        ok := StrToVal(sx, ex);
        ok := StrToVal (sy, ey);
        ok := StrToVal (sz, ez);

        !Koordinaatide asendamine liikumispositsiooni kohale
        Target_a.trans.x:= ex;
        Target_a.trans.y:= ey;
        Target_a.trans.z:= ez;
        MoveJ Target_a,v200,z0,tool1\WObj:=wobj1;
        WaitTime 1;

        !Liikumine tagasi algpunkti
        MoveAbsJ calib_pos,v1000,z0,tool0;
        WaitTime 1;
        Incr j;
      ENDWHILE
      Incr i;
    ENDWHILE
    Close file;
    Decr i;
    Decr j;
    EOF:=FALSE;
  ENDPROC
ENDMODULE
```