TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Anti Lilleaed 206123IAIB

OBJECT DETECTION IN EDUCATIONAL ROBOTS USING ARTIFICIAL NEURAL NETWORKS

Bachelor's Thesis

Supervisor: Gert Kanter
PhD

Tallinn 2023

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Anti Lilleaed 206123IAIB

TEHISNÄRVIVÕRKUDEL PÕHINEV OBJEKTITUVASTUS ÕPPETÖÖS KASUTATAVATELE ROBOTITELE

Bakalaureusetöö

Juhendaja: Gert Kanter
PhD

Tallinn 2023

Autorideklaratsioon

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

Autor: Anti Lilleaed

22.05.2023

Abstract

Object Detection in Educational Robots using Artificial Neural Networks

The robots programming course at Tallinn University of Technology offers students tasks from various fields, including computer vision. Unfortunately, students have not had the opportunity to solve object detection problems on physical robots and have been limited to solving them in a simulation environment. This is because the existing object detection algorithm is not reliable enough for use in the real world.

In this thesis, a new object detection solution was developed for the robots programming course. The thesis begins by providing an overview of the computer vision workflow, main tasks, evaluation metrics, and other related topics, based on literature. Additionally, the process that led to the implementation of the final solution is described, with a primary focus on the selection of a suitable model through comparison. Experiments were carried out to evaluate the final solution's performance in comparison to the existing algorithm.

The main difference compared to the old algorithm is that the new one is based on artificial neural networks. This is beneficial because if the tasks were to be changed in the future, it would be possible to train a new object detection model and just replace it without changing much of the program code.

The thesis is written in Estonian and is 31 pages long, including 7 chapters, 8 figures and 3 tables.

Annotatsioon

Tehisnärvivõrkudel põhinev objektituvastus õppetöös kasutatavatele robotitele

Tallinna Tehnikaülikooli robotite programmeerimise õppeaine pakub üliõpilastele programmeerimisülesandeid mitmetest valdkondadest, üheks neist tehisnägemine. Paraku pole üliõpilastel olnud võimalik füüsilistel robotitel objektituvastuse ülesandeid lahendada, vaid on piirdutud simulatsioonikeskkonnaga. Seda seetõttu, et senine objektituvastuse algoritm polnud füüsilises maailmas piisavalt töökindel.

Käesoleva lõputöö raames loodi selle aine tarbeks uus objektituvastuse lahendus. Lõputöö esimeses osas antakse kirjandusele tuginedes ülevaade tehisnägemise töövoost, põhiülesannetest, hindamismõõdikutest ja muust seonduvast. Lisaks tutvustatakse lahenduseni jõudmise teekonda, mille peamine fookus on võrdluse teel sobiva mudeli valimisel. Eksperimentide abil hinnati lõpliku lahenduse headust võrdluses olemasoleva algoritmiga.

Peamine erinevus võrreldes vana algoritmiga seisneb selles, et uus põhineb tehisnärvivõrkudel. See on kasulik, sest kui ülesannete sisu tulevikus muuta soovitakse, siis on võimalik treenida uus objektituvastuse mudel ning see ilma muud programmikoodi oluliselt muutmata välja vahetada.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 31 leheküljel, 7 peatükki, 8 joonist, 3 tabelit.

Lühendite ja mõistete sõnastik

ANN	*Artificial Neural Network*, tehisnärvivõrk
AP	*Average Precision*, keskmine täpsus
API	*Application Programming Interface*, programmiliides
CNN	*Convolutional Neural Network*, konvolutsiooniline närvivõrk
COCO	*Common Objects in Context*, Microsoft'i tehisnägemise andmestik
F_1	täpsust ja saagist ühendav hindamismõõdik
FOMO	*Faster Objects, More Objects*, rohkemad objektid, kiiremad objektid - objektituvastuse algoritm
FPS	*Frames per Second*, kaadrit sekundis
FN	*False Negative*, väärnegatiivne
FP	*False Positive*, väärpositiivne
IoU	*Intersection over Union*, ühisosa ja ühendi jagatis
JPEG	digitaalpiltide pakkimise standard
mAP	*Mean Average Precision*, keskmiste täpsuste keskmine
MP4	video- ja audioandmete konteinerformaat
R-CNN	*Region-based Convolutional Neural Network*, regioonipõhine konvolutsiooniline närvivõrk - masinõppe mudelite perekond
RGB	*Red Green Blue*, punane, roheline, sinine - liitvärvimudel
RP	robotite programmeerimise õppeaine
RPi	Raspberry Pi, ühest trükkplaadist koosnev arvuti
SDK	Software Development Kit, tarkvaraarenduskomplekt
SSD	*Single Shot MultiBox Detector*, üheetapiline mitmikkasti tuvasti - masinõppe mudelite perekond
TN	*True Negative*, tõsinegatiivne
TP	*True Positive*, tõsipositiivne
VLC	populaarne multimeediamängija
PASCAL VOC	*PASCAL Visual Object Classes*, PASCAL'i märgendatud piltide andmestik
XML	*Extensible Markup Language*, märgistuskeel
YOLO	*You Only Look Once*, sa vaatad vaid kord - masinõppe mudelite perekond

Sisukord

Joonistе loetelu

Tabelite loetelu

1. Sissejuhatus

Tehisnägemise kui valdkonna eesmärk on tõlgendada fotole jäädvustatud maailma, üritades määratleda sellel kujutatu omadusi, nagu näiteks kuju, valgustust ja värvide jaotust [1]. Käesolevas bakalaureusetöös antakse tehisnägemisest teoreetiline ülevaade, kuid töö põhieesmärgist lähtuvalt võetakse fookusesse kitsama skoobiga objektituvastus. Põhieesmärgiks on objektide tuvastamise võimekuse arendamine Tallinna Tehnikaülikooli robotite programmeerimise õppeaine (ainekood ITI0201) tarbeks, et võimaldada ainet sooritavatel üliõpilastel lahendada robotitega ka tehisnägemisele tuginevaid ülesandeid.

Töös testitakse ja võrreldakse erinevaid objektituvastuse algoritme, misjärel valitakse võrdlustulemuste põhjal neist sobivaim ja implementeeritakse see roboti programmiliidesesse. Programmiliides võimaldab üliõpilastel objektituvastuse väljundit pärida, et seda roboti ülesannete lahendamisel sisendinformatsioonina kasutada. Esmajärjekorras määravad lahenduse efektiivsuse objektide tuvastamise kiirus ja täpsus, sest need on töökindla objektituvastuse pakkumisel kriitilise tähtsusega.

Töö peamine keerukus tuleneb robotite riistvaralistest piirangutest. Tagasihoidliku võimsusega Raspberry Pi miniarvutid ei suuda piisavalt kiiresti jooksutada kõrgtasemelisi masinõppe mudeleid, mistõttu on vaja leida lahendus, mis saavutaks parema kiiruse näiteks täpsuse arvelt, kuid oleks siiski piisavalt täpne, et üliõpilased ei peaks raiskama aega ebapädeva algoritmi töökindluse parandamisele ja saaksid keskenduda neile ette nähtud programmeerimisülesannetele.

Lõputöö käigus loodava lahenduse ja juba eksisteeriva lahenduse peal viiakse läbi kolm töökindlusele keskenduvat eksperimenti, et vana lahenduse kui võrdlusbaasi alusel uue lahenduse paremuses veenduda. Lisaks kolmele töökindluse eksperimendile võrreldakse võrdlusbaasi ja loodava lahenduse töökiirust.

Õppeaines pakutavate ülesannete sisu võidakse soovida aja jooksul muuta, näiteks võib tekkida vajadus rohkemate objektiklasside tuvastamiseks. Seda arvesse võttes luuakse lahendus tehisnärvivõrkude baasil, et muudatuste tegemine saaks piirduda vaid fotodest koosneva andmestiku täiendamise ja uue mudeli treenimisega ning et muud programmikoodi oluliselt muutma või täiesti uuesti kirjutama ei peaks. Teatud töö sammud dokumenteeritakse, et aidata huvilistel läbitud protsessi paremini mõista ja läbiviidud samme jäljendada.

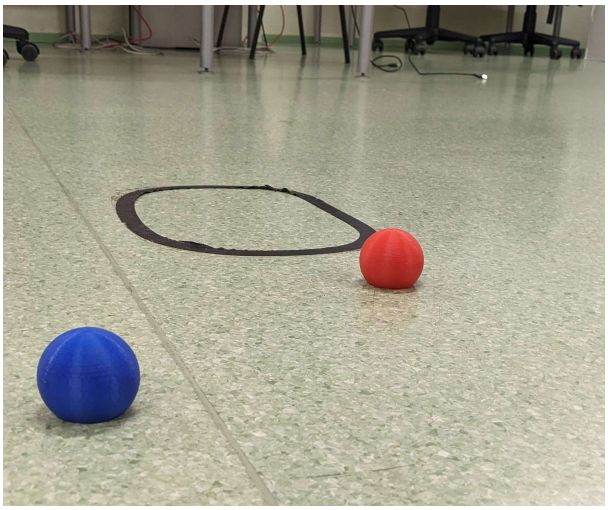
2. Probleemi taust

Töö eesmärk on luua lahendus teatud objektide tuvastamiseks masinõppe abil. See lahendus leiaks kasutust Tallinna Tehnikaülikooli robotite programmeerimise (ainekood ITI0201) õppeaines (edaspidi RP õppeaine). Probleemi tõstatas ja pakkus lahendamiseks välja RP õppeaine õppejõud Gert Kanter, kes on ka käesoleva lõputöö juhendaja.

RP õppeaines lahendavad üliõpilased programmeerimisülesandeid nii simuleeritud robotitel 3D-modelleeritud maailmas kui ka füüsilistel robotitel klassiruumis. Ülesandeid on mitmesuguseid, nende hulgas näiteks ümbritseva keskkonna tajumine ja objektidega manipuleerimine.

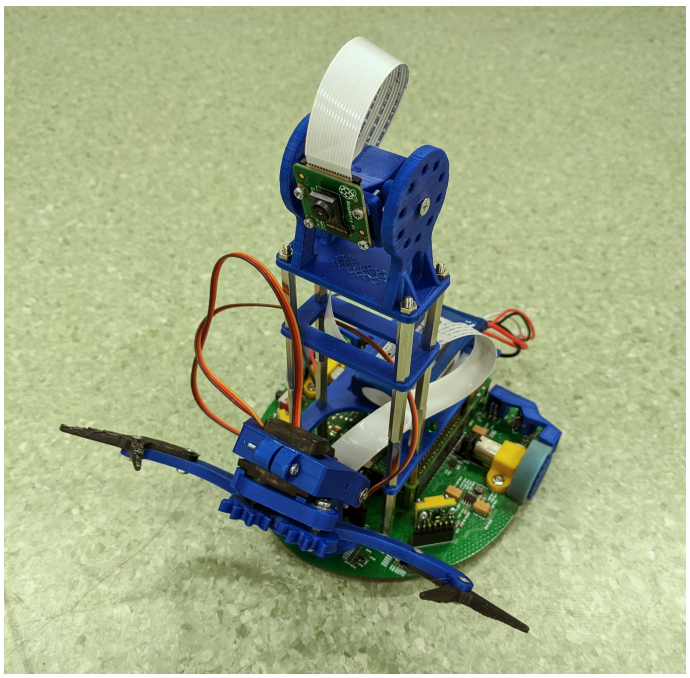
Tehisnägemise alla liigituvaid objektide tuvastamise ülesandeid pole üliõpilastele füüsiliste robotite peal lahendada antud, sest senine objektide tuvastamise lahendus ei ole piisavalt töökindel, et seda õppetöös kasutada. Nimelt vajas see roboti tagasihoidlikult riistvaralt liigset arvutusvõimsust ja andis kohati valeinformatsiooni.

Objektid, mille tuvastamine on vajalik, on punane ja sinine sfäär (vt Joonis 1), mille läbimõõt on ligikaudu 5 cm. Sfääride tuvastamist nõudvateks ülesanneteks on kahe sfääri vahele sõitmine, läbi sfääridest väravate slaalomi sõitmine ning üle põranda laiali olevate sfääride gruppidesse koondamine. Mainitud ülesannetes omab tähtsust ka sfääride värvus - näiteks tuleb sfäärid gruppidesse koondada värvuse järgi. Sellest tulenevalt peab objektituvastuse lahendus suutma eristada sfääride värvust.



Joonis 1. *Sfäärid, mida on tarvis tuvastada.*

Füüsilisteks robotiteks on Raspberry Pi 3B+ miniarvutid koos neile paigaldatud komponentidega, mille hulgas plastikust kereosad, rattad ja elektroonilised komponendid nagu kaameramoodul, laser ja infrapuna kaugusandurid, aku jms (vt Joonis 2).



Joonis 2. *Raspberry Pi miniarvuti peale ehitatud robot.*

Loodav lahendus peab töötlema kaameramooduli kaudu saadavat informatsiooni. Kaameramooduliks on Raspberry Pi *Camera Module 2*, mis kasutab Sony IMX219 8-megapikslist sensorit [2]. Töödeldud informatsioon tuleb tagastada roboti programmiliidesest, et üliõpilased seda ülesannete lahendamisel sisendinformatsioonina kasutada saaksid. Tagastatav informatsioon peab sisaldama tuvastatud objektide nimetusi, nende keskkohtade koordinaate ning objektide raadiuseid.

Fookusesse võetakse objektide tuvastamise kiirus ja täpsus, sest need on hea töökindluse pakkumiseks kriitilise tähtsusega. Muuhulgas proovitakse jõuda tulemuseni, mis võimaldaks objekte tuvastada nii hämaramas ruumis kui ka sellistes keskkondades, kus on rohkem segajaid, näiteks sarnaseid objekte, mida tuvastada ei taheta.

RP aine ülesannete sisu võidakse soovida tulevikus muuta, mistõttu peaks lahendus olema paindlik täiendustele. Näiteks võib tekkida vajadus lisaks sfääridele ka muid objekte tuvastada. See ei tohiks vajada terve algoritmi ümberkirjutamist.

3. Teoreetiline taust

3.1 Sissejuhatus tehisnägemisse

Tehisnägemine kui teadusharu on robootika valdkonnas kasutust leidnud vähemalt 1970-ndatest [3], [4]. Olenevalt definitsioonist võidaksegi üleüldise tehisnägemise alguseks lugeda 1970ndaid [1]. Tehisnägemise eesmärk on muuta robotite jaoks võimalikuks ümbritseva keskkonna „nägemine“, sarnaselt inimestele ja suuremale osale loomariigi esindajatele [5]. Algoritmide sisendiks kasutatakse kaamera kaudu omandatud informatsiooni, et määratleda kaadritele jäädvustatu omadusi, nagu näiteks kuju, mõõtmeid, valgustust, värvide jaotust jms, misjärel on nende põhjal võimalik teha arukaid otsuseid [1]. Arukad otsused on robootikas vajalikud, et robotid suudaksid lahendada keerulise iseloomuga ülesandeid, nagu keskkonnas orienteerumine ja liikuvate objektide jälitamine.

Süvaõppe ning ka saadaoleva riistvara kiire areng, mille hulgas arvutusvõimsuse kasv, mälumahu suurenemine ja kaamerasensorite täienemine, on toonud kaasa tehisnägemist kasutavate rakenduste laialdase leviku. Kuna süvaõppes kasutatavad närvivõrgud eeldavad pigem mudelite treenimist kui programmeerimist, siis nõuab nende kasutamine võrreldes varasemate tehisnägemise lahendustega vähem ekspertteadmisi [6]. Tänapäeval leiavad tehisnägemise süsteemid kasutust näiteks autonoomsetes sõidukites, turvasüsteemides ja meditsiiniliste fotode analüüsimisel [1].

3.2 Pilditöötlus

Järgnevates alapeatükkides antakse teoreetiline ülevaade tehisnägemise ülesannete lahendamisele eelnevast pilditöötluse protsessist.

3.2.1 Pilt ja selle omandamine

Pilti saab defineerida kui funktsiooni $f(x, y)$, kus (x, y) on koordinaadid kahedimensioonilises ruumis ning f nendele koordinaatidele vastav intensiivsus. Igat koordinaadipaari pildil kutsutakse piksliks. Ühtlasi on see pildi kõige väiksem mõõde. Iga piksel tähistab värvilise pildi puhul värvust ja mustvalge pildi puhul halltooni vastavas pildi punktis. Digitaalne foto on seega ristkülikukujuline massiiv piksleid [7].

Pildi omandamise eesmärk on muuta optiline kujutis (reaalse maailma informatsioon)

arvuliste väärtustega massiiviks, mida oleks hiljem võimalik arvutiga töödelda. Niisugune digitaalne pilt on võimalik omandada valgustundliku sensori abil. Selline sensor eksisteerib näiteks digitaalses fotokaameras [7].

3.2.2 Eeltöötlus

Andmete eeltöötlus on iga masinõppemudeli oluline osa, sest andmete kvaliteet ja neist omandatava väärtusliku info hulk mõjutab otseselt mudeli õppimisvõimet [8]. Piltide eeltöötlus võib hõlmata värviruumi transformatsioone, normaliseerimist, mõõtmete muutmist, kontrasti suurendamist, müra eemaldamist jms [9].

Kehvades valgustingimustes tehtud pildid võivad suure tõenäosusega vajada müra eemaldamist. Seda on võimalik teha filtreerimisega, näiteks mediaanfiltri abil [9].

Piltide resolutsiooni ehk mõõtmete vähendamine võib olla mõistlik, et vähendada andmete hulka ja seega kiirendada mudeli õppimisprotsessi [9]. Rakendust võib leida ka piltide ühesuunaline venitamine, mis moonutab andmeid [10].

Samuti võib värvilise ehk RGB pildi teisendada halltoonidesse, et vähendada andmete hulka pildis. Seejuures säilib suurem osa olulisest informatsioonist, mis on vajalik tunnuste eraldamiseks [10].

Künnisfiltri rakendamine (ingl *thresholding*) on pikslitel läbiviidav operatsioon, mille tulemusel pannakse piksli väärtuseks 1 või 0, olenevalt sellest, kas piksel on oma väärtuselt üle mingi valitud piiri või alla selle. Tulemuseks saadakse binaarne, rangelt must ja valge pilt. See on kasulik pildilt kindlate osade esile toomiseks [10].

Piltide töötlemiseks leidub veel lugematul hulgal operatsioone, nagu näiteks servade teravdamine ja histogrammi teisendused [10], mille kasutamine võib osutuda vajalikuks ainult väga spetsiifilisi tehisnägemise lahendusi välja arendades.

3.2.3 Tunnuste eraldamine

Kui eelnevas alapeatükis käsitletud eeltöötluse protsesside puhul olid sisendiks pildid ja väljundiks töödeldud pildid, siis tunnuste eraldamise puhul on sisendiks pildid ja väljundiks üks või mitu pildi tunnust. Tunnused on üldjuhul skalaarsed suurused (nagu näiteks pindala) või vektorid (nagu näiteks kujundi koordinaadid või joone parameetrid) [11].

Tunnused saab kategoriseerida regioonideks, joonteks ja huvipunktideks. Regioonid on omavahel ühenduses olevate pikslite hulgad, mis on mingisuguse pikslite omaduse osas homogeensed. Näiteks punane sfäär on tõlgendatav kui sarnase väärtusega pikslite kogum, mis eristub selgelt teist värvi tausta pikslitest. Sirged jooned kui tunnused on väga tavapärane nähtus inimtekkelistel objektidel nagu hoonetel. Kolmas tunnuste klass, huvipunktid, on oma naaberpikslitega võrreldes oluliselt kõrgema intensiivsusega pikslid või objektide nurkade pikslid. Kuna huvipunktid eristuvad muust väga selgelt, on nende töökindel tuvastamine kõige tõenäolisem [11].

3.3 Tehisnägemise ülesanded

Tehisnägemise ülesannete lahendamine toimub pilditöötlusprotsessi järel [8]. Keerukamateks tehisnägemise ülesanneteks võivad olla näiteks inimkeha pooside mõistmine, näoilmetest emotsioonide tuletamine, rahvahulga loendamine, objekti jälitamine, käekirja ja numbrimärkide lugemine [1]. Järgnevalt on toodud tüüpilisemad tegevused või ülesanded, mis leiavad aset eelmainitud ülesannete lahendamisel [12]. Seejuures tuleks tähele panna, et järgnev kujutab vaid ühte võimalikku ülesannete liigitust.

Klassifitseerimine

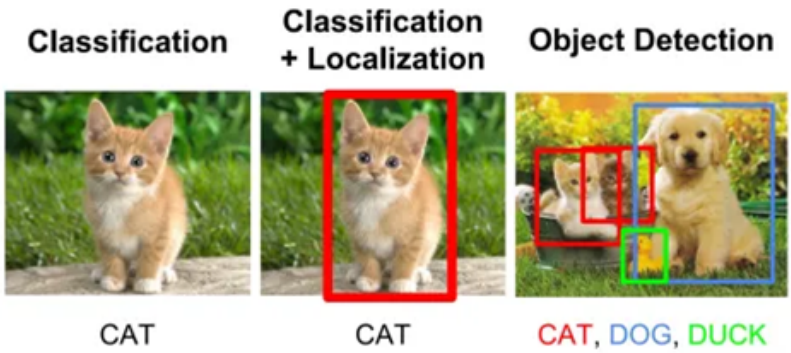
Fotode klassifitseerimise (ingl *image classification*, *image recognition*) eesmärk on ennustada foto langemist mingisse ühte klassi sellel kujutatud objekti põhjal (vt Joonis 3). See tähendab, et klassifitseerimisel on väljundiks objektiklass [12], [13].

Lokaliseerimine

Objektide lokaliseerimisega (ingl object *localization*) püütakse täpsustada ühe või mitme objekti olemasolu ja paiknemine fotol (vt Joonis 3). Lokaliseerimise puhul on väljundiks objekte piiritlevate ristkülikute ehk piiritluskastide koordinaadid. Lokaliseerimine iseseisvalt ei anna teada, mis objektiga on tegu [12].

Objektide tuvastamine

Objektide tuvastamise (ingl *object detection*) ülesanne hõlmab endas kahte eelnevat - fotode klassifitseerimist ja objektide lokaliseerimist (vt Joonis 3). Väljundiks saadakse objektide piirikastide koordinaadid koos klassinimedega [12], [13].



Joonis 3. *Tehisnägemise ülesandeid. Vasakult: foto klassifitseerimine, ühe objekti klassifitseerimine koos lokaliseerimisega, mitme objekti tuvastamine [14].*

Semantiline segmenteerimine

Semantilise segmenteerimisega (ingl *semantic segmentation*) määratakse foto iga piksli kohta, millisesse objektiklassi see piksel kuulub (vt Joonis 4) [15].

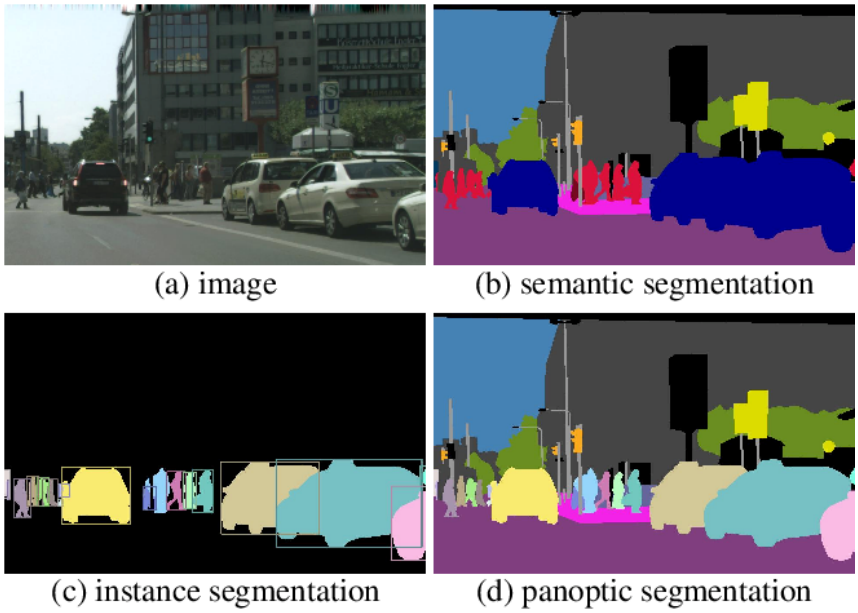
Eksemplaride segmenteerimine

Eksemplaride segmenteerimise (ingl *instance segmentation*) eesmärk on eristada samadesse objektiklassidesse kuuluvaid objekte (vt Joonis 4). Näiteks kahte kõrvuti seisvat inimest ei segmenteerita semantilise segmenteerimise sarnaselt mitte ühe pikslite kogumina, mille klassiks „inimene“, vaid eristataks kahe erineva eksemplarina [16]. Eksemplarideks ei loeta amorfseid piirkondi, nagu näiteks muru, taevast ja sõiduteid, mistõttu eksemplaride segmenteerimine neid ei käsitle [15].

Väljundiks saadakse eksemplari kontuuridega segmentatsiooni maskid vastavate klassinimedega [15]. Objektidele võidakse lisada ka nelinurksed piirikastid, mistõttu saab selle alamülesandeks lugeda objektide tuvastamist [16].

Panoptiline segmenteerimine

Kolmas segmenteerimise variant on panoptiline segmenteerimine (ingl *panoptic segmentation*), mis ühildab eelnevad kaks (vt Joonis 4). Väljundiks saadakse iga foto piksli kohta klassinimi ja eksemplari identifikaator. Pikslid, mis omavad sama klassinime ja identifikaatorit, kuuluvad samale objektile [15].



Joonis 4. *Segmenteerimise tüübid: (a) sisendfoto, (b) semantiline segmentatsioon, (c) eksemplaride segmentatsioon, (d) panoptiline segmentatsioon [15].*

3.4 Ilma süvaõppeta objektituvastus

Üks tavapäraseid lähenemisi pildilt objektide tuvastamiseks on n-ö „šablooni sobitamine“ (ingl *template matching*) [17]. See on protsess, mille käigus libistatakse huvipakkuva objekti kujutist kui šablooni üle terve pildi ja arvutatakse igal sammul kujutise ja selle poolt kaetud pildiosa sarnasus [18].

See lähenemine on implementeeritud läbi kahedimensioonilise konvolutsiooni, mis tähendab et šablooni iga libistamise puhul arvutatakse väljundpildi vastava piksli väärtus kahe maatriksi kattuvate elementide korrutustulemuste summana. Esimeseks maatriksiks on sisendpilt ja teiseks šabloon, mida kutsutakse ka konvolutsiooni kerneliks. Šablooni sobitamise nõrkuseks on pildid, millel on objekt poolikult, vale nurga all või teistsugustes valgustingimustes [18].

Närvivõrke mitte kasutavaid objektituvastuse lahendusi leidub veel mitmeid, mis pole niivõrd elementaarse tööpõhimõttega, sealhulgas näiteks Viola-Jones'i algoritm, histogramm orienteeritud gradientidest (ingl *histogram of oriented gradients*) ja tugivektor-masinad (ingl *support vector machine*) [19].

3.5 Süvaõppega objektituvastus

Tehisnärvivõrgud (ingl *artificial neural networks*, ANN) on üks mustrite klassifitseerijate liik, mis pakuti välja 1989ndate algul. Sellel on võimekus õppida selgeks mustreid, mida on süvaõppeta meetodite abil keeruline analüüsida [19].

Mõned tüüpilised tehisnärvivõrkude arhitektuurid on mitmekihilised pertseptronid (ingl *multilayer perceptron*), Hopfieldi võrgud (ingl *Hopfield neural networks*) ja konvolutsioonilised närvivõrgud (ingl *convolutional neural networks*, CNN) [19]. Neist viimase arhitektuuri selgitatakse järgnevas alapeatükis.

3.5.1 Konvolutsioonilised närvivõrgud

Konvolutsioonilised närvivõrgud on saanud süvaõppe valdkonnas kõige levinumateks närvivõrkudeks [20]. CNN'id on analoogsed traditsioonilistele ANN'idele, sisaldades läbi õppimise iseennast optimiseerivaid neuroneid. Peamine erinevus tuleneb sellest, et CNN'id on loodud mustrite tuvastamiseks just piltidelt, vähendades märkimisväärselt arvutusvõimsust, mida vajaks ANN'id samade ülesannete lahendamiseks [21].

Arhitektuur

Konvolutsiooniliste närvivõrkude arhitektuur on inspireeritud elusolendite nägemismeelest [20]. CNN'id koosnevad kihtidest, mille vahel on ära jaotatud omavahel ühendatud tehisneuronid ehk sõlmed. Erinevalt teistest ANN'idest on ühe kihi iga sõlm ühendatud vaid väikese piirkonnaga sellele järgnevas kihis [21].

Konvolutsioonilises närvivõrgus on kihte kolme tüüpi: konvolutsioonilised kihid, ahenduskihid ja täissidusad kihid. Esimesele konvolutsioonilisele kihile eelneb sisendkiht, mida CNN'i kihtide hulka otseselt ei loeta. See sisaldab pildi pikslite väärtusi. Konvolutsioonilistes kihtides rakendatakse sisendile teatud filtreid ehk kerneleid, et eraldada arvutuste teel sisendpildilt lihtsaid tunnuseid, nagu servi ja nurki, sarnaselt peatükis 3.4 toodud šablooni sobitamisele.

Sel moel saadud tunnuskaardid saadetakse edasi aktivatsioonifunktsiooni, mis võimendab tunnuste eripärasid. Ahenduskihte kasutatakse tunnuskaartide allasämplimiseks, et vähendada võrgustiku arvutuslikku keerukust ja suurendada vastupidavust sisendpildi varieeruvustele. CNN'i lõpus asuvaid täissidusaid kihte kasutatakse sisendpildi klassifitseerimiseks ühte või mitmesse klassi.

Levinumad mudelid

Konvolutsiooniliste närvivõrkude baasil töötavad objektituvastuse enamlevinud mudelid saab jaotada kaheks: ühe- ja kaheetapilised [22].

Kaheetapilisteks loetakse selliseid võrgustikke, millel on eraldi moodul regioonietteppanekute genereerimiseks [23]. Seega genereeritakse kaheetapilise objektituvastuse puhul esimeses etapis regioonide või objektide ettepanekud ja teises etapis klassifitseeritakse ja lokaliseeritakse need piiritluskastidega [22]. Tipptasemel mudelite näideteks on regioonipõhiste R-CNN mudelite perekonda kuuluvad Faster R-CNN ja Mask R-CNN. Kuna sellistel süsteemidel on vaja sooritada kaks eraldi sammu, siis kipuvad need jääma objektituvastuse kiiruselt ja arhitektuuri keerukuselt alla üheetapilistele mudelitele [23].

Üheetapilised objektituvastajad klassifitseerivad ja lokaliseerivad semantilised objektid ühe korraga. Nende tugevuseks võrreldes kaheetapiliste mudelitega on reaalajas tuvastamine. Levinud tipptaseme mudelid on YOLO (ingl *You Only Look Once*) ja SSD (ingl *Single Shot MultiBox Detector*). SSD ilmudes oli see oluliselt kiirem ja täpsem kaheetapilisest Faster R-CNN'ist kui ka YOLO'st, kuid sellel oli raskusi väikeste objektide tuvastamisega. See probleem kõrvaldati hilisemate täiendustega [23].

YOLO puhul vajab märkimist tõsiasi, et sellel on rohkelt uusi versioone, mis kujutavad endast edasiarendusi inimeste poolt, kes ei ole selle algsed loojad. Põhiversioonide hulgast saab tuua välja YOLOv2, YOLO9000, YOLOv3, YOLOv4 jne [23]. Samuti eksisteerib YOLO baasil loodud mudeleid tagasihoidlikuma riistvaraga seadmetele, sealhulgas Fast-YOLO [24], Tinier-YOLO [25], YOLO v3-Tiny [26], xYOLO [27] jms. Nende mudelite suurem kiirus on kohati saavutatud kehvema tuvastustäpsuse arvelt.

3.6 Hindamismõõdikud

Mõõtühik **kaadrit sekundis** (ingl *frames per second*, FPS) viitab ühe sekundi jooksul töödeldud piltide arvule, andes aimu objektituvastuse mudeli reaalajalisest kiirusest. Üldiselt peetakse vähemalt 20 FPS saavutavat mudelit reaalajaliseks detektoriks [13].

Objektide tuvastamise headuse hindamisel kasutatakse järgnevaid võimalikke tulemusi:

- **Tõsipositiivne** (ingl *true positive*, TP) - pildil eksisteeriva objekti korrektne tuvastamine kattuva piiritluskastiga;
- **Väärpositiivne** (ingl *false positive*, FP) - pildil mitteeksisteeriva objekti ekslik tuvastamine või pildil eksisteeriva objekti tuvastamine valesti paigutatud piiritluskastiga;

- **Väärnegatiivne** (ingl *false negative*, FN) - pildil eksisteeriva objekti mittetuvastamine [28].

Erinevalt piltide klassifitseerimisest ei leia objektituvastuse hindamise puhul kasutust tõsinegatiivsete (ingl *true negative*, TN) tulemuste arv, sest piltidel leidub lõpmatu arv võimalikke piiritluskaste, mida pildilt tuvastada ei soovita ja mis liigituks seega tõsinegatiivsete tulemuste alla [28].

Usalduspiir (ingl *confidence threshold*) on parameeter, mis võimaldab mudeli poolt tagastatud tuvastuse tõenäosuse põhjal otsustada, kas tuvastust tuleks üldse arvesse võtta. Teisisõnu, see määrab, kas olukord loetakse positiivseks või negatiivseks. Seevastu tõese ja väära tuvastuse määramiseks kasutatakse piiritluskastide ühisosa ja ühendi jagatise tulemust [29].

Ühisosa ja ühendi jagatis (ingl *intersection over union*, IoU) on mõõdik lokaliseerimise headuse ehk tuvastatud objekti piiritluskasti paigutamise hindamiseks. Selle arvutamiseks jagatakse mudeli poolt välja pakutud piiritluskasti ja tegeliku piiritluskasti kattuva osa pindala nende ühendi pindalaga (vt Valem 3.1) [13], [28].

$$IoU = \frac{\text{Piiritluskastide ühisosa pindala}}{\text{Piiritluskastide ühendi pindala}} \ . \tag{3.1}$$

Võrreldes IoU väärtust mingi valitud läviväärtusega *t*, nagu näiteks 0,5, saab määrata objektituvastuse korrektseks või ebakorrektseks. Teisisõnu, tulemused, kus $IoU \geq t$, on tõsipositiivsed ja tulemused, kus $IoU < t$, on väärpositiivsed [28].

Täpsus (ingl *precision*) annab aimu mudeli võimekusest tuvastada ainult asjakohaseid objekte (vt Valem 3.2) [28].

$$\text{Täpsus} = \frac{TP}{TP + FP} \ . \tag{3.2}$$

Täpsus üksinda pole piisavalt usaldusväärne mõõdik, sest see ei võta arvesse väärnegatiivseid tulemusi. Seetõttu on võimalik saavutada maksimaalne täpsus tuvastades korrektselt kasvõi ainult ühe objekti ja jättes kõik ülejäänud objektid üldse tuvastamata [30].

Saagis (ingl *recall*) annab aimu mudeli võimekusest tuvastada kõik objektid, mis tuvastamiseks ette nähtud olid (vt Valem 3.3) [28].

$$\text{Saagis} = \frac{TP}{TP + FN} \ . \tag{3.3}$$

Ka näiliselt head saagise väärtust on võimalik saavutada valelikult, kui klassifitseerida pildil absoluutselt kõik objektiks [30].

$\mathbf{F}_1$ skoor ühildab täpsuse ja saagise ühtseks mõõdikuks, et lahendada eelnimetatud puudujäägid (vt Valem 3.4) [30]:

$$F_1 = 2 \cdot \frac{\text{Täpsus} \cdot \text{Saagis}}{\text{Täpsus} + \text{Saagis}} \ . \tag{3.4}$$

Skoori kõrgeim võimalik väärtus on 1, viidates täiuslikule täpsusele ja saagisele. Väikseim võimalik väärtus 0 viitab täpsuse või saagise osas kehvimale võimalikule tulemusele.

Täpsuse-saagise kõver (ingl *precision* × *recall curve*) koondab samuti täpsuse ja saagise. Täpsus paigutatakse diagrammi y-teljele ning saagis x-teljele. Kõvera punktid väljendavad täpsuse ja saagise väärtusi erinevate usalduspiiride juures. Kahte mudelit võrreldes on parim see, mille kõver paikneb kõrgemal [29].

Keskmine täpsus (ingl *average precision*, AP) on täpsuse-saagise diagrammi kõveraaluse osa pindala. Selle saab arvutada iga objektiklassi kohta [31].

Keskmiste täpsuste keskmine (ingl *mean average precision*, mAP) on enim kasutatud mõõdik, mida kasutatakse erinevate mudelite võrdlemisel [23]. Sellega mõõdetakse mudeli täpsust üle kõigi andmestikus sisalduvate objektiklasside. Seega on see üle kõigi klasside omandatud keskmiste täpsuste aritmeetiline keskmine (vt Valem 3.5) [13], [28]:

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \ , \tag{3.5}$$

kus AP_i on keskmine täpsus i-ndas objektiklassis ja N on vaadeldavate klasside koguarv [28].

4. Mudelite võrdlus

Järgnevalt antakse ülevaade katsetatud algoritmidest ning tuuakse välja eristavad leiud, mis viisid lõplikus lahenduses kasutatava algoritmi valimiseni. Olukordades, kus klassiruumi robotitele polnud ligipääsu või tekkisid muud ettenägematud takistused, katsetas autor mudeleid oma isikliku Raspberry Pi 4B peal. Seega võib kõikide tulemuste puhul, kus on katsevahendina mainitud RPi 4B miniarvutit, arvestada RPi 3 B+ kontekstis veidi kehvemate tulemustega, kui on välja toodud.

4.1 SSDLite-MobileNet-v1 mudel

Kõige esimese katsetusena Raspberry Pi kasutamisest objektituvastuse tarbeks järgiti algajatele suunatud SSDLite-MobileNet-v1 mudeli juhendmaterjali. Eelmainitud mudel on üks Google näidismudelitest, mis on treenitud Microsoft COCO andmestiku peal ja suudab tuvastada 80 erinevat objekti, sealhulgas inimesi, autosid, kruuse [32].

Mudeli jooksutamiseks paigaldati RPi süsteemi TensorFlow Lite [33], mis on masinõppe tarkvarateek tagasihoidlikuma riistvaraga seadmete jaoks. Mudel käivitus edukalt ja tuvastas suurepäraselt objekte, kuid jäi kesiseks kiiruse poolest, jäädes 2-5 FPS vahemikku.

4.2 YoloX mudel

Järgmise sammuna püüti leida mudel, mille välja pakutud kiirus piiratud riistvaraga süsteemidele oleks suurem kui SSDLite-MobileNet-v1 peal saavutatu. Leiti kogumik [34] suure hulga populaarsete mudelitega, mis on Q-Engineering nimelise ettevõtte poolt edasi arendatud ja koos erinevate juhenditega välja pandud [35]. Ühtlasi on neid just Raspberry Pi sarnaste seadmete ja mikrokontrollerite peal kasutamiseks muudetud.

Mudelite hulgas pakkusid kõige paljulubavamaid tulemusi NanoDet [36] ja YoloFastestV2 [37]. Raspberry Pi 4 64-bit operatsioonisüsteemi peal ja 1950 MHz protsessori ülekiirenduse juures on Q-Engineering tiim NanoDet kiiruseks mõõdetud 13,0 FPS ja YoloFastestV2 kiiruseks 18,8 FPS. Headele tulemustele vaatamata on nende mudelite tööle saamine keerukas protsess. Protsessi pole paraku nende kahe mudeli jaoks dokumenteeritud. Küll aga eksisteerib juhendmaterjal [38] ühe teise mudeli, YoloX [39], [40] jaoks. YoloX kiiruseks on eelmainitud riistvara peal mõõdetud 7,0 FPS.

Autor otsustas esmalt läbida põhjaliku YoloX juhendmaterjali ja selle tööle saada, et saada aimu, kas NanoDet ja YoloFastestV2 käivitamine hõlmaks samasugust protsessi. Samuti tuli YoloX käivitamise järel otsustada, kas sellised mudelid võiksid olla roboti liidesesse mõistlikul kombel implementeeritavad.

YoloX tööle saamine osutus suure sammude hulga tõttu väga keerukaks ja ajakulukaks. Kõrgemalt tasemelt vaadatuna koosnes protsess järgnevatest põhisammudest:

1. Multimeedia raamistiku GStreamer paigaldamine [41];
2. Tehisnägemise algoritmide teegi OpenCV (C++) paigaldamine [42];
3. Code::Blocks IDE paigaldamine projekti kasutamiseks [38];
4. Süvaõppe raamistiku ncnn paigaldamine [43];
5. YoloX allalaadimine ja kompileeritavale kujule viimine [38], [44];
6. YoloX ja GStreamer projektide kokku ühendamine Code::Blocks'is, et kasutada RPi kaameramoodulit [38].

Lisaks esines palju väikeseid takistusi, mis tuli käsitsi lahendada - näiteks RPi *swap* partitsiooni ajutiselt suurendada ja korrigeerida GStreamer *pipeline*'i parameetreid.

Objektituvastuse akent üritati panna ka FPS näidikut kuvama, et veenduda selle kiiruses, kuid see osutus arvatust keerulisemaks. Silmaga hinnates jäi kiirus siiski samale tasemele SSDLite-MobileNet-v1 algoritmiga, mida oligi oodata, sest autorite mõõdetud 7,0 FPS oli saavutatud ülekiirendatud RPi 4 protsessoriga. Märkimisväärne erinevus on aga lahenduse käivitamisele kulunud aeg. Kui SSDLite-MobileNet-v1 saadi tööle 30 minutiga, siis YoloX tööle saaamisele kulus 24 tundi.

Kuigi antud mudelid kompileeritakse riistvaralähedasel kombel C++ keelest ja on tänu sellele kiiremad, ei andnud protsessi keerukus alust arvata, et selle implementeerimine Python'is kirjutatud roboti API'sse oleks mõistlik. Eriti seetõttu, et pea olematu võit kiiruses nõuaks kordades suuremat ajakulu. Samuti erinesid NanoDet ja YoloFastestV2 programmikoodid näidiseks toodud YoloX'i omast niivõrd palju, et autor kahtles oma võimetes C++ koodiga sellisel tasemel ümber käia.

4.3 EfficientDet0 ja EfficientDet4 mudelid

Järgmiseks sihiks võeti mudelite testimine kasutades enda fotodest andmestikku, mis tuli esmalt kokku koguda ja seejärel märgendada. Juhendmaterjalina kasutati TensorFlow YouTube kanali videot [45], milles käiakse läbi mudelite treenimise protsess Google Colaboratory pilvekeskkonnas.

4.3.1 Andmestiku kogumine

Andmestik koguti esialgu vaid ühes keskkonnas jäädvustatud fotodest. Selleks keskkonnaks olid hämarama valgustusega ruumid autori kodus. Hiljem planeeriti andmestikku täiendada hästi valgustatud klassiruumis jäädvustatud fotodega. Varieeruvate valgustingimustega keskkondade kasutamine on üks moodus, kuidas treenida mudel, mis töötaks tõhusalt erinevates oludes. Antud lahenduse puhul on see ka oluline, sest õppetöö läbiviimise ruum võib ajas muutuda.

Kõik andmestikku jõudnud fotod sfääridest omandati läbi Raspberry Pi kaameramooduli, et andmestik saaks võimalikult ligilähedane reaalse maailma tingimustega. Seejuures pärinevad fotod sarnaselt kõrguselt ja kaldenurgalt, jäljendamaks kaameramooduli paiknemist robotil.

Kuna ükshaaval piltide jäädvustamine otse Raspberry Pi peale olnuks üleliia tülikas ja ajakulukas tegevus, leiti selleks alternatiivne moodus. Nimelt filmiti kaameramooduli kaudu videofail ja eraldati sellest teatud intervallide tagant kaadreid. Seejuures ei salvestatud videot otse Raspberry Pi failisüsteemi, vaid striimiti üle võrgu tööjaama järgnevatel põhjustel:

- Raspberry Pi'de sees kasutatavad mälukaardid ei ole eriti mahukad ja olenevalt salvestatava video pikkusest ja kvaliteedist võib see osutuda takistuseks;
- Video tulnuks mugavamaks kaadrite eraldamiseks ja nende märgendamiseks paratamatult tööjaama teisaldada;
- Raspberry Pi peal kaamerapildi videona salvestamiseks tulnuks kirjutada skript, sest ühtegi eksisteerivat lahendust selleks ei leitud.

Video striimimine RPi'st tööjaama viidi läbi kasutades VLC multimeedia mängimise rakendust [46], mis paigaldati mõlemasse süsteemi. RPi poolses VLC kasutajaliideses algatati striimi edastamine ning tööjaamas selle vastuvõtmine. VLC võimaldas striimi salvestada .ts laiendiga faili. Seejärel sai samas rakenduses selle MP4 formaati ümber teisendada, et sellest kaadreid eraldada.

Kaadrite eraldamiseks kirjutati Python'i skript, mis võimaldas määrata, kui mitme kaadri tagant kaader JPEG pildifailina eraldi kausta salvestada. Piisavalt erinevad järjestikkused kaadrid saadi salvestades iga 30. kaader, sest selle aja jooksul liigutati robotit piisavalt palju ringi. Kaadrite omavaheline erinemine on oluline, et treenitav mudel ei saaks kallutatud suure arvu ühetaoliste piltide poolt. Ülejäänud liiga sarnased pildid eemaldati andmestikust käsitsi.

Selle protsessi läbimise tulemusena omandati esmane 191-st fotost koosnev andmestik.

4.3.2 Andmestiku märgendamine

Fotod otsustati märgendada Label Studio [47] tööriistaga, mis kujutab endast käsurea kaudu käivitatavat keskkonda. Punase ja sinise sfääri märgendite nimedeks seati vastavalt „*Red sphere*“ ja „*Blue sphere*“. Et tagada märgendatud andmete ühene tõlgendamine mudelite poolt, peeti silmas järgmisi põhimõtteid:

- Kustuta andmestikust foto, kui see pole märgendamiseks sobilik, s.t on mudeli treenimiseks ebaoluline või eksitav, mõne teisega liiga sarnane või duplikaat;
- Märgendikast tuleb paigutada võimalikult tihedalt objekti ümber, kuid veel olulisem on jätta objekt terves ulatuses märgendikasti sisse;
- Kui objekt jääb osaliselt fotolt välja või on varjatud mingi teise objekti poolt, siis märgenda objekt ainult sel juhul, kui 40% sellest on nähtaval;
- Kui üks märgendatav objekt on osaliselt teise märgendatava objekti ees, siis märgendite nelinurgad peavad kattuma;
- Olukordades, kus osa objektist on varjatud millegi muu poolt, jäta märgendikasti sisse ainult see osa objektist, mis on nähtaval, mitte ära venita nelinurka edasi sinna, kuhu selle objekti varjatud osa tegelikult ulatuda võib, sest mudelile võib selline informatsioon olla eksitav.

Märgendatud fotod eksporditi EfficientDet mudelite peal kasutamiseks PASCAL VOC XML formaadis.

4.3.3 Andmestiku kasutamine ja tulemused

Andmestik jagati käsitsi treeningandmestikuks ja valideerimisandmestikuks, pakiti kokku ning laeti Colaboratory keskkonda üles. Ära muudeti märgendiklasside nimetused ja kirjutati Colaboratory’s andmeid sisse lugev moodul ümber, et veebiaadressite sisselugemise asemel oleks võimalik sisse lugeda enda failisüsteemis asuvat andmestikku. Enda andmestikku lugedes ilmnes probleem - andmete lugeja eeldas PASCAL VOC XML formaadis märgendeid, kuid ilma esimese metaandmete reata, mille Label Studio nendele lisanud oli. Kõigist failidest esimese rea eemaldamiseks kirjutati Python’i skript.

Eduka andmete sisselaadimise järel treeniti mudel ja eksporditi see Raspberry Pi’sse käivitamiseks. Google Colaboratory’s olev kood võimaldas treenida erinevaid EfficientDet mudeleid. Miniarvutitele sobivam EfficientDet0 mudel saavutas RPi 4B peal keskmise

kiiruse 5 FPS ja tuvastas sfääre üsna töökindlalt. Kuna sinine sfäär paistis selles etapis kasutatud esmase andmestiku fotodel välja palju tumedamalt, isegi mustana, siis tuvastas see musti objekte kohati siniste sfääridena.

Huvi pärast katsetatud EfficientDet4 mudel osutus hoopis teiseks äärmuseks – suurepärane täpsus kiiruse arvelt. See tuvastas sfääre väga täpselt, mis tähendab, et märgendikastid kuvati ideaalselt sfääride ümber, ilma teisi objekte sfäärideks klassifitseerimata. Kiirus oli see-eest äärmiselt aeglane - vaid 0,6 FPS.

Neid tulemusi arvesse võttes saadi aru, et EfficientDet mudelid poleks lahenduse jaoks piisavalt kiired, kuid see-eest osutusid kasulikuks õppimisvahendiks, et saada märgendatud andmestiku loomise ja kasutamise kogemus.

4.4 FOMO mudel

Viimase võimaliku mudelina leiti kõigi eelnevatega võrreldes kordades kõrgemaid kiirusi lubav lahendus – FOMO (ingl *Faster Objects*, *More Objects*) [48]. Nimelt lubavad FOMO autorid Raspberry Pi 4 seadmetel saavutada kiirust kuni 60 FPS, olles järjekindlalt tuvastanud objekte 40-60 kaadrit sekundis. Seejuures mainivad nad 30 korda väiksemat võimsustarvet ja mälu kasutust kui MobileNet SSD ja YOLOv5 mudelid [49].

FOMO mudeleid on võimalik luua läbi Edge Impulse nimelise veebirakenduse [50]. See pakub kõiki vajalikke tööriistu, loomaks tehisnägemise rakendusi väiksema jõudlusega seadmetele. Ühtlasi märgendati kõik 191 andmestiku pilti Edge Impulse keskkonnas uuesti, sest see ei toetanud niisugustes formaatides märgendifailide importimist, mida varasemalt kasutatud LabelImg eksportimiseks pakkus.

FOMO mudeli treenimise järgselt prooviti seda lokaalselt Raspberry Pi 3B+ peal jooksutada, ilma selle faile püsivalt seadmesse paigaldamata. Selleks kasutati *Edge Impulse for Linux* tööriista [51]. Iga kaadri töötlemisele kulus 18-21 ms, mis on ümber teisendatuna ligikaudu 47,6-55,6 FPS.

4.5 Võrdluse tulemused

Mudeleid testides ja uurides võrreldi neid nelja kriteeriumi alusel: kiirus, käivitamisele kulunud aeg, täpsus ja implementeerimise keerukus. Mudelite puhul, mille kiirust polnud võimalik mõõta, on toodud mudeli autorite või demonstreerijate väljapakutud kiirus.

Tööle saamiseks kulunud aeg tähistab aega, mis kulus mudeli juhendamaterjali läbimise algusest kuni mudeli käivitamiseni. Mudelite käivitamiseks kasutati juhendmaterjalis leiduvat andmestikku, selle puudumisel enda andmestikku, mille loomise aega arvesse ei võetud. Mudel loeti käivitatuks siis, kui oli võimalik näha objektituvastuse eelvaadet ehk piiritluskastide paigutusi kaamera videopildil.

Täpsust ühegi mudeli puhul ei mõõdetud, mistõttu kujutavad need endast autori subjektiivset hinnangut nähtule. Nimelt oli mudeleid käivitades võimalik näha piiritluskastide paigutamist ja selle põhjal saada hea arusaam mudeli töökindlusest ja piiritluskastide paigutamise täpsusest, mis loetakse ühiselt kriteeriumi "täpsus" alla.

Implementeerimise keerukus on samuti subjektiivse loomuga. See kujutab endast autori hinnangut mudeli ja sellega kaasneva tarkvara või raamistike paigaldamise keerukusele, kui seda peaks lõpplahendusena paigaldama roboti API'sse või API'ga muul moel ühildama. Arvesse võeti nähtud dokumentatsiooni ja juhendmaterjale ning autori tehnilisi oskusi.

Võrdluse tulemused on toodud tabelis 1. Tööle saamiseks kulunud aega ega hinnangulist täpsust pole NanoDet ja YoloFastestV2 mudelite puhul toodud, sest neid mudeleid otsustati mitte testida.

Tabel 1. *Mudelid koos võrdlustulemustega.*

Kriteerium	**Mudel**						
	SSDLite-MobileNet-v1	YoloX	NanoDet	Yolo-FastestV2	Efficient-Det0	Efficient-Det4	FOMO
Kiirus	3,5 FPS	7 FPS	13 FPS	18,8 FPS	5 FPS	0,6 FPS	51,6 FPS
Tööle saamiseks kulunud aeg	0,5 h	24 h	-	-	2 h	2 h	4,7 h
Hinnanguline täpsus	Hea	Keskmine	-	-	Kehv	Väga hea	Hea
Implementeerimise keerukus	Keskmine	Keeruline	Väga keeruline	Väga keeruline	Keskmine	Keskmine	Kerge

Kriteeriumite põhjal kõige sobilikuma mudeli väljaselgitamiseks kasutati kaalutud skooride meetodit. Igale kriteeriumile seati kaal, mis väljendab selle kriteeriumi olulisust kõigi kriteerimite seas. Näiteks kiirusele seati kaal 40%, sest see on hea lõpplahenduse puhul olulisim tegur. Kui lahendus peaks osutuma tudengitele kasutamiseks liiga aeglaseks, siis selle osas ei saaks nad midagi muuta. Kehva täpsust võiks aga olla võimalik teatud võtetega parendada, näiteks mudeli väljastatavatele andmetele filtrit rakendades, et üksikud eksimused välja "siluda". Seejuures on oluline märkida, et sellist olukorda saavutada ei soovita, vaid sellest saab mõelda kui halvimast juhust.

Tabelis 1 toodud tulemused said teisendatud ümber skoorideks, milles iga skoor jääb vahemikku 0-10 punkti. Tulemused on toodud tabelis 2.

Kiirust teisendati 2,5 FPS vahemike kaupa. See tähendab, et tulemus 0-2,5 FPS võrdsustati nulli punktiga, 2,5-5 FPS ühe punktiga, 5-7,5 kahe punktiga jne. Seejuures iga vahemiku osaks loeti vahemiku otspunktidest väiksem arv. Käivitamisele kulunud aeg teisendati järgnevalt: kui kulus kuni 1 h, siis 10 punkti; kui kuni 2h, siis 9 punkti; kui kuni 3h, siis 8 punkti jne. Täpsuse hinnangud teisendati kümne punktisele skaalale eelnevatest jäigemalt: väga kehv - 0 punkti, kehv - 2,5 punkti, keskmine - 5 punkti, hea - 7,5 punkti, väga hea - 10 punkti. Sarnaselt tehti ka implementeerimise keerukusega: väga keeruline - 0 punkti, keeruline - 2,5 punkti, keskmine - 5 punkti, kerge - 7,5 punkti, väga kerge - 10 punkti.

Tabel 2. *Mudelite võrdlustulemused ümber teisendatuna skoorideks.*

Kriteerium	**Kaal**	**Mudel**						
		SSDLite-MobileNet-v1	YoloX	NanoDet	Yolo-FastestV2	Efficient-Det0	Efficient-Det4	FOMO
Kiirus	40%	1	2	5	7	2	0	10
Tööle saamiseks kulunud aeg	15%	10	0	-	-	9	9	6
Hinnanguline täpsus	25%	7,5	5	-	-	2,5	10	7,5
Implementeerimise keerukus	20%	5	2,5	0	0	5	5	7,5
Kaalutud lõppskoor	100%	4,78	2,55	-	-	3,78	4,85	**8,28**

Teistest oluliselt kõrgema kaalutud lõppskoori saavutas FOMO. Seejuures on oluline ära märkida, et FOMO oleks saavutanud veelgi suurema skoori, kui kiiruse skooriks teisendamise skaala oleks valitud laiaulatuslikumalt. Praegusel juhul oli võimalik saavutada maksimaalne skoor juba kiirusega 25 FPS. FOMO saavutas aga keskmise kiiruse 51,6 FPS. Skaala valiti niisugune, sest juba 20 FPS loetakse reaalajaliste objektituvastuse lahenduste puhul adekvaatseks tulemuseks [13].

Kuna paremaid tulemusi lubavaid mudeleid ei leitud ja kõrgeim kaalutud lõppskoor kinnitas FOMO tulemuste üleüldist paremust, siis otsustati lõpplahendusse implementeerida FOMO objektituvastuse mudel.

5. Tehniline lahendus

5.1 Ettevalmistused

Kuigi ettevalmistused said tehtud juba enne mudelite võrdlemist, tuuakse need käesolevas peatükis välja kui töö põhieesmärgi täitmisel vajalikuks osutunud tegevused.

Kuna lõpplahenduse implementeerimine ja katsetamine hõlmasid robotite failisüsteemis muudatuste tegemist, siis oleks otse robotite mälukaartidele muudatuste tegemine olnud riskantne. Seda arvesse võttes kirjutas autor RP õppeaine robotites kasutusel oleva failisüsteemi koopia ehk süsteemikujutise oma isiklikule mälukaardile ümber ja kasutas seda kõigi tegevuste tarbeks.

Kaugligipääsu jaoks paigaldati RPi operatsioonisüsteemi vajalik tarkvara. Vastasel juhul oleks autor pidanud iga süsteemidevahelise tegevuse jaoks klaviatuuri, monitori ja hiirt arvuti küljest RPi külge ühendama ja sealt jälle tagasi.

Tegevused, mis ei olnud ettevalmistuse mõttes märkimisväärsed, kuid mida tingimata ette planeerida ei osatud ja mis seega rohkelt ajakulu kaasa tõid, olid:

- roboti komponentide lahti kruvimine kaameramooduli eemaldamiseks, et seda isikliku RPi 4 külge ühendada;
- võrguühendusele seatud parameetrite eemaldamine, et robot ka väljaspool ülikooli võrku suudaks ühenduda;
- ülikooli laboris töötades võrguparameetrite tagasi lisamine;
- regulaarne roboti aku vahetamine ja sellega kaasnevad taaskäivitused;
- RPi mäluruumi tühjendamine, kui seda tarkvara paigaldamiseks väheks jäi;
- operatsioonisüsteemi versiooni ja arhitektuuri, kettaruumi, muutmälu, võrguühenduste jms informatsiooni pärimine terminali kaudu;
- koodimuudatuste tegemine otse Raspberry Pi peal, terminali või Thonny programmeerimiskeskkonna kaudu;
- programmikoodi ja muude failide käsitsi teisaldamine süsteemide vahel;
- tarkvara versioonikonfliktide lahendamine ning uuenduste läbiviimine;
- kataloogide ja failide õiguste haldamine.

5.2 Valitud mudeli tööpõhimõte

FOMO (ingl *Faster Objects*, *More Objects*) kui mikrokontrolleritele ja teistele vähem võimekatele seadmetele loodud masinõppe algoritm üritab saavutada suurepärast jõudlust koheldes objektituvastust kui klassifitseerimisülesannet. Nimelt töötleb FOMO algoritm sisendiks saadud pilti kui ruudustikku, mille iga ruudu osas viiakse eraldi läbi pildi klassifitseerimine [49].

FOMO erineb teistest objektituvastuse algoritmidest seetõttu, et see ei anna väljundiks objektide piiritluskaste, vaid nende keskpunktide asukohad. Küll aga võimaldab FOMO neist tuletada ligilähedased piiritluskastid. Kuna peamine eesmärk on tuvastada sfääre või tulevikus ka muid väikeseid kompaktse iseloomuga objekte, siis on ligikaudsed piiritluskastid aktsepteeritavad. FOMO aktsepteerib sisendina nii mustvalgeid kui ka RGB pilte [49].

FOMO kasutab tavapärast klassifitseerimise konvolutsiooniliste närvivõrkude arhitektuuri, kuid asendab sellest viimased kihid nii-öelda piirkondliku klasside tõenäosuse kaardiga. Olgugi, et saadud kaart on mõõtmetelt väiksem kui sisendpilt, on võimalik viimase kihi peal tuvastatud paiknemised viia vastavusse sisendpildga. Sisuliselt saadakse selle tulemusel kuumakaart (ingl *heat map*) objektide asukohtadest [49].

Sisendpilti osadeks tükeldava ruudustiku ühe ruudu mõõtmete vaikeväärtuseks on 8*8 pikslit. Seega koheldaks pilti mõõtmetega 96*96 pikslit kui ruudustikku mõõtmetega 12*12 pikslit. Sellest tuleneb üks FOMO piiranguid - see ei ole võimeline eristama kahte lähestikku paiknevat objekti, kui nende keskpunktid satuvad klassifitseerimisel sama ruudu sisse. Küll aga saab seda parendada, kui valida kuumakaardile suuremad mõõtmed [49].

5.3 Mudeli treenimine

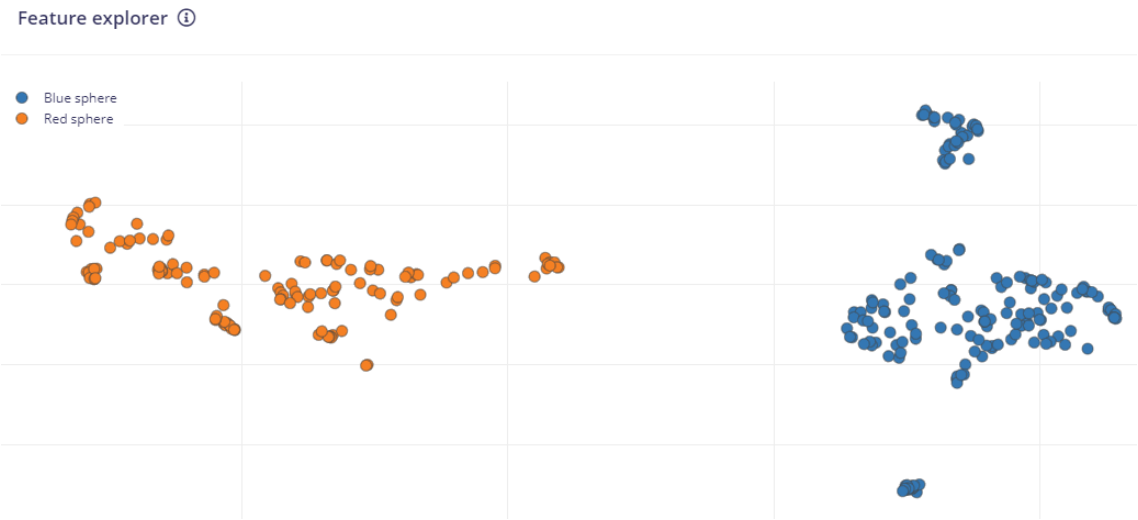
Lisaks eelnevalt märgendatud 191-le pildile lisati Edge Impulse keskkonda veel 241 klassiruumis jäädvustatud fotot, mis samuti keskkonna tööriistadega ära märgendati. Nii omandati lõpuks 432-st pildist koosnev andmestik, mis jaotati keskkonna soovituslike vaikeväärtuste kohaselt 80% ulatuses treeningandmeteks ja 20% ulatuses testandmeteks.

Üle kõigi piltide esineb sinist sfääri eksemplarina 340 korral, punast aga 333 korral. Punaste ja siniste eksemplarid jaotati testandmete vahel nii, et mõlemat esineks neis 67 korda. Seitsme eksemplari võrra jäi seega erinevus treeningandmetesse. Kokku eksisteerib andmestikus 58 fotot, millel ei esine ühtegi sfääri või on need nii väikeses osas nähtaval, et

need otsustati mudeli mitte eksitamise huvides jätta märgendamata.

Objektituvastus eeldab antud keskkonnas ruudukujulist sisendpilti. Võimalikult paljude tunnuste säilimiseks valiti mudeli treenimisel piltide mõõtmeteks esialgu 320*320 pikslit, mis on FOMO mõistes üsna suur. Piltide suurust muudeti valiku "*fit longest axis*" kohaselt, mis mahutab pildi pikema telje valitud mõõtmete sisse ning lisab lühemale teljele nii-öelda "polsterduse". Seda tehti lootuses mitte kaotada osa märgenduste informatsioonist.

Sfääre sisaldavate piltide osas genereeriti tunnuste kaart, mis võimaldab leida andmestikust ebaharilikke, märgendamata ja valesti märgendatud andmeid (vt Joonis 5) [52]. Tunnuskaart esitas kõik sfääride esinemised punktidena, kusjuures see viitas suurepärasele võimekusele antud andmestikust punast ja sinist sfääri eristada. Nimelt paiknevad eri värvi punktid kaardil teineteisest selgelt eemal. Küll aga võivad punktide liigsed kuhjumised viidata vastavate andmete liigsele omavahelisele sarnasusele.



Joonis 5. *Tunnuste kaart.*

Lahenduseks valitud FOMO variant baseerub MobileNetV2 mudelil [49]. FOMO treeniti õppimiskiirusega 0,001 ja üle 30 treeningtsükli. Andmete rikastamise (ingl *data augmentation*) valik lülitati sisse, mis tähendab, et olemasolevaid andmestiku fotosid transformeeriti erinevatel viisidel. Teisisõnu suurendati andmestikku tehislikult.

Treening saavutas valideerimisandmetel F_1 skoori 99,1%. Valideerimisandmeid oli seejuures 20%, kuid erinevalt testandmetest ei olnud täpsustatud, mis andmete hulgast need võeti. Väljundiks antud segadusmaatriksilt võis välja lugeda, et ühel korral tuvastati punast sfääri seal, kuhu seda tegelikult märgitud polnud. Pole välistatud, et see oli hoopis positiivne olukord, sest mudel võis tuvastada mõnd osaliselt pildi serval paiknevat ja meelega

märgendamata jäänud sfääri. Maatriksi kohaselt leidus ka üks olukord, kus sinine sfäär jäi märgendamata.

Niiviisi saavutati üks esimesi kasutatavaid mudeleid, kuid sellel esines probleeme. Peamise probleemina oli selle tuvastamine aeglasem, kui eksisteeriva lahenduse oma, mida muuhulgas aegluse tõttu õppetöös kasutusse ei võetud. Samuti jättis mudel vahepeal sfääre tuvastamata, kui need selgelt kaadris olid. Sellele järgnevalt treeniti veel mitmeid mudeleid, et leida parim treenimise konfiguratsioon.

Olulisema muudatustena vähendati sisendpildi mõõtmed 320*320 piksli pealt 192*192 piksli peale, mis osutus mõistlikuks keskteeks kiiruse ja täpsuse vahel. Teise olulise muudatusena loobuti treenimisel kõigist autori korteriruumides jäädvustatud fotodest, sest need erinesid liigselt klassiruumis tehtutest ja tõid kaasa rohkem kahju kui kasu.

Parim ja ühtlasi viimane mudel treeniti niisiis valikutega "*fit shortest axis*", õppimiskiirus 0,001, 60 treeningtsüklit, andmete rikastamine sisse lülitatud. Valideerimisandmetel saavutati F_1 skoor 99,2%, mis tingimata palju mudeli headuse kohta ei ütle, sest korteris tehtud fotode eemaldamisega vähenes valideerimisandmete rikastatus ehk valideerimine muutus lihtsamaks. Küll aga on see kõrgeim tulemus, mida üle kõikvõimalike konfiguratsioonide saavutada suudeti. Väiksemad pildimõõtmed parandasid kiirust märkimisväärselt ning korteris tehtud fotode eemaldamine elimineeris peamised tuvastamise töökindluse probleemid.

5.4 Lahenduse paigaldamine roboti programmiliidesesse

Roboti programmiliides koosneb Python failidest [53]. Nendes asendati senine närvivõrkudeta objektituvastuse kood uuega ning lisati juurde objektituvastuse mudeli kasutamiseks vajalikud failid.

Autor sai teatud määral programmikoodi mallina kasutada Edge Impulse pakutava Python SDK [54] näidiste hulgas leiduvaid faile. Neile tehtud muudatused sisaldasid muuhulgas rohke ebavajaliku koodi eemaldamist. Esmalt asendati vanema algoritmi kaameramoodulit aktiveeriv ja sellelt sisendit päriv programmikood uuega. Kui vana kood eeldas Raspberry Pi *Camera Module 2* kasutamist, siis uus kood aktsepteerib ka suvalise veebikaamera ühendamist roboti külge ja selle kohest kasutamist objektide tuvastamiseks.

Kui kaamera kasutamine saadi tööle, kirjutati kood, mis kogu vajaliku informatsiooni FOMO väljundist välja loeb ja nõutud kujul programmiliidese väljundisse tagastab. Küll aga tuli objektide raadiused omandada piiritluskasti ligikaudsete mõõtmete põhjal, nimelt

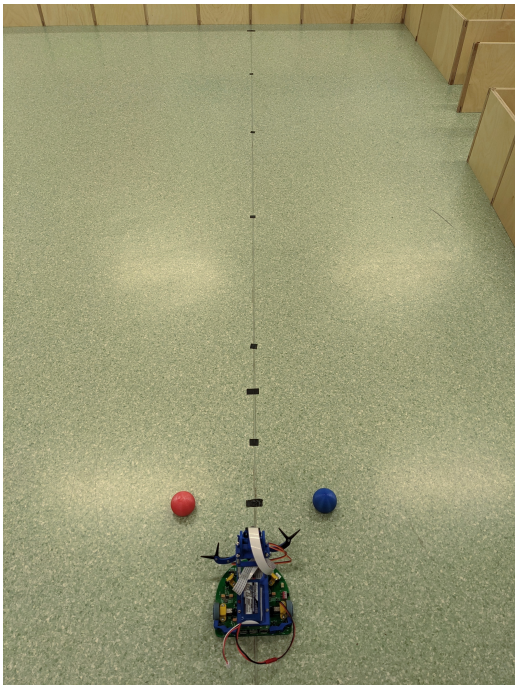
kõrguse ja laiuse aritmeetilist keskmist kahega jagades. See tuleneb FOMO mudeli piirikastide leidmise piirangutest.

6. Tulemused

Loodud lahenduse hindamiseks kavandati kolm eksperimenti. Esimene eksperiment on objektide tuvastamine erinevatelt kaugustelt, teine objektide tuvastamine sarnaste objektide juuresolekul ning kolmas sfääride tuvastamine roboti pöördliikumise ajal. Eksperimentides võrreldakse loodud lahendust seni eksisteerinud lahendusega. Teisisõnu on vana lahendus kui seni parim mudel uue mudeli headuse hindamisel võrdlusbaasiks. Eksperimentidega seotud fotod ja tekstifailid on leitavad lõputöö repositooriumis (vt Lisa 2).

6.1 Tuvastamise kaugus

Kaugete objektide tuvastamine on oluline, et robotitega oleks võimalik lahendada ülesandeid, kus sfäärid paiknevad klassiruumi põrandal laiali ning mitte tingimata roboti vahetus läheduses. Kauguse mõõtmiseks paigutatakse sfäärid robotist erinevatele kaugustele (vt Joonis 6) ja leitakse maksimaalne kaugus, millel tuvastamine edukalt toimub. Tuvastamise edukust hinnatakse väljundfotode põhjal, millel on näha tuvastatud objektide piiritluskaste. Kui piiritluskast asub vähemalt mingil määral sfääri peal, siis loetakse tuvastus edukaks.



Joonis 6. *Tuvastamise kauguse eksperiment.*

Sinine ja punane sfäär paigutati teineteisest 30 cm kaugusele. Teisisõnu jäeti nende vahele

piisavalt ruumi, et robot sinna ära mahuks. Sellest võib mõelda kui tavapärasest sfääride paiknemise olukorrast, sest niisugusele kaugusele võidakse sfääre hakata teineteisest paigutama näiteks slaalomi sõitmise ülesande puhul. Sfäärid paigutati roboti kaamerast aga 30 cm, 45 cm, 1 m, 2 m, 3 m, 4 m, ja 5 m kaugusele. Joonisel 6 nähtavad sfäärid asusid 15 sentimeetri kaugusel ning jäid valitud kaamera kaldenurga tõttu roboti vaateväljast välja, mistõttu alustatigi 30 sentimeetri pealt. Iga vahemaa puhul tehti kolm järjestikkust fotot, et veenduda ka tuvastuste järjepidevuses. Kuna kolm järjest jäädvustatud fotot oleksid pea identsed, siis viitaks erinevuste olemasolu kehvale töökindlusele.

Tulemused on toodud tabelis 3, milles "SP" tähistab nii sinise kui punase sfääri edukat tuvastamist, "S" tähistab ainult sinise sfääri tuvastamist. Olukordi, kus tuvastataks punast sfääri, kuid mitte sinist, ei esinenud. Ülejäänud lahtrid märgivad olukordi, kus tuvastusi ei olnud.

Tabel 3. *Tuvastamise kauguse eksperimendi tulemused.*

Algoritm	**Sfääride kaugus kaamerast**						
	30 cm	45 cm	1 m	2 m	3 m	4 m	5 m
Vana	SP	S	S	-	-	-	-
	SP	S	S	-	-	-	-
	SP	S	S	-	-	-	-
Uus	SP	SP	SP	SP	S	-	-
	SP	SP	SP	SP	S	-	-
	SP	SP	SP	SP	S	-	-

Eksperimendi tulemusel selgus, et loodud lahendus on tuvastamise kauguse poolest vanast lahendusest oluliselt etem, suutes tuvastada mõlemat sfääri kuni kahe meetri kauguselt. Vana lahendus pakkus sama tulemust vaid kuni 30 sentimeetri kauguselt. Mõlema algoritmi puhul tuli ilmsiks, et punase sfääri tuvastamine on probleemsem kui sinise sfääri tuvastamine, sest selle tuvastamine lakkas väiksemalt kauguselt. Pole välistatud, et selle võis tingida miski, mida inimsilmaga ei taju - näiteks valgustatuse erinemine sfääride asukohtade vahel. Mõlemad algoritmid olid tuvastustega iga vahemaa kolme iteratsiooni osas järjepidevad.

6.2 Tuvastamine sarnaste objektide juuresolekul

Objektide tuvastamist sarnaste objektide juuresolekul hinnatakse väljundi vaatlusel. Selleks pannakse robot tuvastusi piiritluskastidena väljundpiltidele salvestama. Eksperimendi jaoks valitakse hulk värvuselt ja kujult sfääridele sarnanevaid klassiruumis leiduvaid objekte, mis ringikujuliselt põrandale paigutatakse. Nende hulka paigutatakse ka sfäärid. Robot paigutatakse ringi keskele, objektidest umbes ühe meetri kaugusele. Seejärel pöörleb robot ühe täisringi, salvestades väljundpilte nii kiiresti kui vastav algoritm suudab. Täispöörde

tegemiseks kulub valitud kiirusel umbes 11 sekundit. Kui selgub, et võõraid objekte loetakse sfäärideks, siis saab sellest järeldada, et nende paiknemine roboti vaateväljas võib reaalses kasutuses probleeme tekitada ja nõuda vastavate segajate eemaldamist enne objektituvastuse ülesannete lahendamist.

Väljavalitud objektide hulgas on kuhja laotud sinised akud, must teibirull, punane papist karp, siniste ja punaste plastikosadega robotid, punaste ja mustade juhtmete kimp, punane ja sinine marker, arvutitool (ennekõike selle mustad ümarad rattad) ning roboti sinine käpp (vt Joonis 7).

Joonis 7. *Sarnaste objektide eksperiment.*

Tulemustest selgus, et loodud lahendus on väga tundlik sarnast värvi objektidele, tuvastades enamusi ringis asetsevaid siniseid või punaseid objekte vähemalt ühe korra. Muuhulgas tuvastas see punaseid sfääre ka kandilise pappkarbi ja taustale jääva kapiukse keskmes. Sellest võib järeldada, et mudel õpetati pigem tuvastama sinist ja punast värvust, kui sfääri ümarat kuju. Teisalt on see tulemus loogiline, sest andmestikus esines teisi punaseid ja siniseid objekte minimaalselt, mistõttu sama värvi mittesfääriliste objektide eiramise vajadust närvivõrgule ei teadvustatudki.

Märkimist väärib see, et musta värvi arvutitooli rattaid, musta teibirulli ning tumedat juhtmekimpu kordagi sfäärina ei tuvastatud. Musta värvi objekte esines andmestikus aga palju, mis võiski nende eduka vältimise tagada. Kuna suuremat osa eksperimendi jaoks väljavalitud objektidest tavapäraselt põrandal ei paiknegi, siis võib eesmärgi lugeda saavutatuks, sest toolirattad on põrandal tavapärane nähtus ja just nendega mudel ootuspäraselt toime tulebki.

Täiendavalt salvestati uue algoritmi igale väljundpildile piiritluskastide kõrvale tuvastuste tõenäosus ehk mudeli enesekindlus. Kui mitmeid objekte pakuti kõrge tõenäosusega sfäärideks vaid vahepeal, siis on selle juures positiivne, et sfääre pakuti väga kõrge tõenäosusega sfäärideks iga kaadri puhul. Kuna algoritmile lisati moodus usalduspiiri muuta, siis võib probleemide ilmnedes olla võimalik paljusid valetuvastusi elimineerida usalduspiiri tõstmise kaudu.

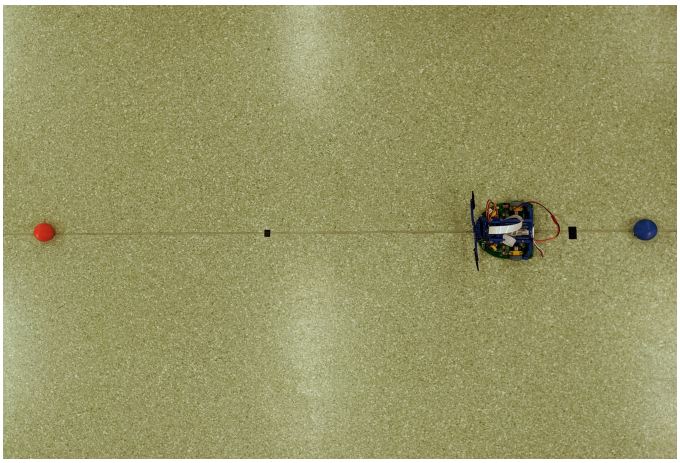
Vana algoritm tegi palju vähem valetuvastusi, kuid see-eest ei suutnud punast sfääri ühelgi kaadril tuvastada. Samuti tuvastas see ühel kaadril sinise sfääri peal lausa kahte sinist sfääri. FOMO algoritmi eripärast tulenevalt ei saaks sellist asja uue lahenduse puhul juhtuda.

Vana algoritmi tugevuseks on objekti täpsete mõõtmete omandamine. See paigutab piiritluskastide asemel piiritlusringid ning saab seda tänu täpsete mõõtmete omandamisele teha suure detailsusega. Uus lahendus on võimeline tagastama vaid objekti ligikaudsed mõõtmed ning jääb selles osas vanale alla. Kokkuvõtvalt on uus selle eksperimendi osas siiski asjalikum, sest täpsemate mõõtmete tagastamine on teisejärguline aspekt ning ennekõike peaks mudel sfääride tuvastamisega toime tulema.

6.3 Tuvastamine pöördliikumisel

Pöördliikumise eksperimendiga kontrollitakse roboti võimet tuvastada sfääre samal ajal, kui robot ühe koha peal maksimaalsel kiirusel pöörleb. Nimelt saavad pildid roboti liikumise ajal jäädvustatud udusemalt kui kohapeal seistes, moonutades seeläbi piltidel olevaid objekte. Tavapäraselt kasutatavat pöörlemiskiirust peegeldab paremini eelnev eksperiment, milles pöörlemine märgatavat udusust ei tekitanud. Seega on maksimaalse pöörlemiskiiruse kasutamise eksperiment ennekõike kasulik algoritmide võimete piiride mõistmiseks.

Punane sfäär paigutatakse robotist kahe meetri kaugusele (vt Joonis 8), nagu osutus uue algoritmi võimete piiriks kauguse mõõtmise eksperimendis. Sinine sfäär paigutatakse vastupidises suunas robotist 20 sentimeetri kaugusele. Robot pöörleb ühe koha peal kuus täisringi. Ühe täispöörde tegemiseks kulub umbes neli sekundit. Sarnaselt eelnevatele eksperimentidele salvestatakse väljundpilte koos piiritluskastidega. Tulemusi hinnatakse väljundi vaatlusel.



Joonis 8. *Pöördliikumise eksperiment.*

Uus algoritm tuvastas mõlemat sfääri edukalt kõigil kaadritel, kus need selgelt vaateväljas oli. Samuti mõnel korral, kus vaid osa sfäärist vaateväljas oli.

Vana algoritm tuvastas sinist sfääri selle kõigil esinemistel edukalt. Ka siis, kui see osaliselt vaateväljas oli. Punast sfääri ei tuvastanud see mitte ühelgi kaadril, kuid see on ootuspärane, olles kooskõlas kauguse eksperimendi tulemustega.

Uue algoritmi puhul leidus kõikide kaadrite peale vaid üks valetuvastus, kus taustal olevat punast kapiust kõrge tõenäosusega sfääriks pakuti. Vana algoritm tegi aga ohtralt valetuvastusi. Nimelt pidas see kõike punast, mis taustale jäi, punaseks sfääriks.

Vana algoritmi peal korrati eksperimenti nii, et ka punane sfäär asus robotist 20 sentimeetri kaugusel. Selle põhjal sai taaskord kinnitust tõsiasi, et vanal algoritmil on raskusi just punase sfääri tuvastamisel, sest see tuvastas punast sfääri vaid üksikutel kaadritel, kusjuures piiritlevad ringjooned joonistas see samuti ebakorrektselt.

Kokkuvõtvalt saadi eksperimendi tulemusel teada, et piltide udusus ei põhjusta täiendavaid probleeme mitte kummagi algoritmi jaoks. Kuna uue algoritmi töökindlus säilib ka suurimal pöörlemiskiirusel, siis võiks see sobida ülesannete lahendamiseks, ilma et roboti liikumise pärast muret tundma peaks.

6.4 Tuvastamise kiirus

Objektituvastus osutus kiiremaks kui varasema algoritmi puhul võimalik oli. Nimelt kulub terve ühe tuvastustsükli täitmisele vanal algoritmil keskmiselt 0,337 sekundit ja uuel algoritmil keskmiselt 0,207 sekundit. Keskmised arvutati 150 järjestikkuse tuvastuse

ajakulude põhjal.

Olukordades, kus soovitakse veelgi paremat kiirust, saab 192*192 pikslise sisendiga mudelifaili vahetada välja juba autori poolt treenitud 160*160 pikslise mudelifaili vastu, millel kulub ühele tuvastustsüklile vaid ligikaudu 0,1 sekundit. Sel juhul võib aga olla vajalik klassiruumist võimalikke segajaid eemaldada ja programmikoodis usalduspiiri väärtust muuta, et töökindlus 192*192 pikslise mudeliga samaväärne oleks.

7. Kokkuvõte

Antud bakalaureusetöö eesmärk oli luua uus objektide tuvastamise lahendus Tallinna Tehnikaülikooli robotite programmeerimise õppeaines (ainekood ITI0201) kasutamiseks, sest senine algoritm ei olnud piisavalt töökindel reaalses maailmas kasutamiseks. Tänu loodud lahendusele saavad ainet sooritavad üliõpilased edaspidiselt lisaks simulatsioonikeskkonnale ka füüsiliste robotite peal tehisnägemisele tuginevaid ülesandeid lahendada.

Töö peamine keerukus tulenes robotite riistvaralistest piirangutest. Nimelt ei suuda tagasihoidliku võimsusega Raspberry Pi miniarvutid piisavalt kiiresti kõrgtasemelisi masinõppe mudeleid jooksutada. Seetõttu testiti ja võrreldi töös erinevaid objektituvastuse algoritme ja valiti võrdlustulemuste põhjal neist sobivaim. Parimaks osutus FOMO algoritm, millel treeniti objektituvastuse mudel. Algoritm implementeeriti roboti programmiliidesesse. Programmiliides võimaldab üliõpilastel objektituvastuse väljundandmeid pärida, et seda ülesannete lahendamisel sisendinformatsioonina kasutada. Lahendust luues keskenduti peamiselt hea kiiruse ja täpsuse saavutamisele, sest need on töökindla objektituvastuse pakkumisel kriitilise tähtsusega.

Loodud lahenduse peal viidi läbi kolm töökindlusele keskenduvat eksperimenti, mille käigus selgus uue mudeli üleüldine paremus seni eksisteerinud lahendusega võrreldes. Seejuures tulid ilmsiks üksikud miinused, mis loodud lahendusel eksisteerivad, kuid mis selle õppetöösse kaasamist ei takista. Lisaks kolmele töökindluse eksperimendile mõõdeti vana ja uue lahenduse töökiirust, mille tulemused kinnitasid, et uus lahendus on oma eellasest ka kiirem.

Õppeaines pakutavate ülesannete sisu võidakse soovida aja jooksul muuta, näiteks võib tekkida vajadus rohkemate objektiklasside tuvastamiseks. Seda arvesse võttes loodi lahendus tehisnärvivõrkude baasil, et muudatuste tegemine saaks piirduda vaid fotodest koosneva andmestiku täiendamise ja uue mudeli treenimisega ning et muud programmikoodi oluliselt muutma või täiesti uuesti kirjutama ei peaks. Teatud töö sammud dokumenteeriti, et aidata huvilistel läbitud protsessi paremini mõista ja läbiviidud samme jäljendada.

Viited

[1] R. Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022. DOI: https://doi.org/10.1007/978-3-030-34372-9.

[2] Raspberry Pi. *Raspberry Pi Camera Module 2*. URL: https://www.raspberrypi.com/products/camera-module-v2/. Kasutatud 02.05.2023.

[3] Carnegie Mellon University. *Shakey*. The Robot Hall of Fame. URL: http://www.robothalloffame.org/inductees/04inductees/shakey.html. Kasutatud 18.04.2023.

[4] Computer History Museum. *Shakey*. URL: https://www.computerhistory.org/revolution/artificial-intelligence-robotics/13/289. Kasutatud 18.04.2023.

[5] D. M.-K. Lam. *Evolution of Sight in the Animal Kingdom*. URL: https://webvision.med.utah.edu/2014/07/evolution-of-sight-in-the-animal-kingdom/. Kasutatud 15.04.2023.

[6] N. O' Mahony et al. "Deep Learning vs. Traditional Computer Vision". In: *arXiv e-prints* (2019). DOI: 10.48550/arXiv.1910.13796.

[7] V. K. Mishra, S. Kumar and N. Shukla. "Image acquisition and techniques to perform image acquisition". In: *SAMRIDDHI: A Journal of Physical Sciences, Engineering and Technology* 9.01 (2017), pp. 21–24. DOI: https://doi.org/10.18090/samriddhi.v9i01.8333.

[8] K. Maharana, S. Mondal and B. Nemade. "A review: Data pre-processing and data augmentation techniques". In: *Global Transitions Proceedings* 3.1 (2022). International Conference on Intelligent Engineering Approach (ICIEA-2022), pp. 91–99. DOI: https://doi.org/10.1016/j.gltp.2022.04.020.

[9] A. Wang, W. Zhang and X. Wei. "A review on weed detection using ground-based machine vision and image processing techniques". In: *Computers and Electronics in Agriculture* 158 (2019), pp. 226–240. DOI: https://doi.org/10.1016/j.compag.2019.02.005.

[10] C. Solomon and T. Breckon. *Fundamentals of Digital Image Processing: A practical approach with examples in Matlab*. John Wiley & Sons, 2011.

[11] P. Corke. “Image Feature Extraction”. In: *Robotics, Vision and Control: Fundamental Algorithms In MATLAB® Second, Completely Revised, Extended And Updated Edition*. Cham: Springer International Publishing, 2017, pp. 413–458. DOI: https://doi.org/10.1007/978-3-319-54413-7_13.

[12] J. Brownlee. *A Gentle Introduction to Object Recognition With Deep Learning*. URL: https://machinelearningmastery.com/object-recognition-with-deep-learning/. Kasutatud 15.04.2023.

[13] X. Wu, D. Sahoo and S. C. H. Hoi. “Recent advances in deep learning for object detection”. In: *Neurocomputing* 396 (2020), pp. 39–64. DOI: https://doi.org/10.1016/j.neucom.2020.01.085.

[14] A. Bornstein. *Using Object Detection for Complex Image Classification Scenarios Part 4*. Towards Data Science. 2019. URL: https://towardsdatascience.com/using-object-detection-for-complex-image-classification-scenarios-part-4-3e5da160d272. Kasutatud 19.04.2023.

[15] A. Kirillov, K. He, R. Girshick, C. Rother and P. Dollár. “Panoptic Segmentation”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 9396–9405. DOI: 10.1109/CVPR.2019.00963.

[16] Hasty GmbH. *Instance Segmentation*. URL: https://hasty.ai/docs/mp-wiki/model-families/instance-segmentor. Kasutatud 15.04.2023.

[17] N. S. Hashemi, R. B. Aghdam, A. S. B. Ghiasi and P. Fatemi. “Template Matching Advances and Applications in Image Analysis”. In: (2016). DOI: 10.48550/arXiv.1610.07231.

[18] X. Binjie and J. Hu. “6 - Fabric appearance testing”. In: *Fabric Testing*. Woodhead Publishing, 2008, pp. 148–188. DOI: https://doi.org/10.1533/9781845695064.148.

[19] G. Cheng and J. Han. “A survey on object detection in optical remote sensing images”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 117 (2016), pp. 11–28. DOI: https://doi.org/10.1016/j.isprsjprs.2016.03.014.

[20] Z. Li, F. Liu, W. Yang, S. Peng and J. Zhou. “A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects”. In: *IEEE Transactions on Neural Networks and Learning Systems* 33.12 (2022), pp. 6999–7019. DOI: 10.1109/TNNLS.2021.3084827.

[21] K. O’Shea and R. Nash. “An Introduction to Convolutional Neural Networks”. In: *arXiv e-prints* (2015). DOI: 10.48550/arXiv.1511.08458.

[22] F. Sultana, A. Sufian and P. Dutta. “A Review of Object Detection Models Based on Convolutional Neural Network”. In: *Intelligent Computing: Image Processing Based Applications*. Singapore: Springer Singapore, 2020, pp. 1–16. DOI: 10.1007/978-981-15-4288-6_1.

[23] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar and B. Lee. “A survey of modern deep learning based object detection models”. In: *Digital Signal Processing* 126 (2022). DOI: https://doi.org/10.1016/j.dsp.2022.103514.

[24] M. J. Shafiee, B. Chywl, F. Li and A. Wong. “Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video”. In: *arXiv e-prints* (2017). DOI: 10.48550/arXiv.1709.05943.

[25] W. Fang, L. Wang and P. Ren. “Tinier-YOLO: A Real-Time Object Detection Method for Constrained Environments”. In: *IEEE Access* 8 (2020), pp. 1935–1944. DOI: 10.1109/ACCESS.2019.2961959.

[26] P. Adarsh, P. Rathi and M. Kumar. “YOLO v3-Tiny: Object Detection and Recognition using one stage improved model”. In: *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*. 2020, pp. 687–694. DOI: 10.1109/ICACCS48705.2020.9074315.

[27] D. Barry, M. Shah, M. Keijsers, H. Khan and B. Hopman. “xYOLO: A Model For Real-Time Object Detection In Humanoid Soccer On Low-End Hardware”. In: *2019 International Conference on Image and Vision Computing New Zealand (IVCNZ)*. 2019, pp. 1–6. DOI: 10.1109/IVCNZ48456.2019.8960963.

[28] R. Padilla, S. L. Netto, and E. A. B. da Silva. “A Survey on Performance Metrics for Object-Detection Algorithms”. In: *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*. 2020, pp. 237–242. DOI: 10.1109/IWSSIP48289.2020.9145130.

[29] M. El Aidouni. *Evaluating Object Detection Models: Guide to Performance Metrics*. 2019. URL: https://manalelaidouni.github.io/Evaluating-Object-Detection-Models-Guide-to-Performance-Metrics.html#precision---recall-and-the-confidence-threshold. Kasutatud 16.04.2023.

[30] *Precision, Recall, & F1 Score Intuitively Explained*. Youtube. 2020. URL: https://youtu.be/8d3JbbSj-I8. Kasutatud 16.04.2023.

[31] A. Anwar. *What is Average Precision in Object Detection & Localization Algorithms and how to calculate it?* 2022. URL: https://towardsdatascience.com/what-is-average-precision-in-object-detection-

localization-algorithms-and-how-to-calculate-it-3f330efe697b. Kasutatud 16.04.2023.

[32] EdjeElectronics. *How to Run TensorFlow Lite Object Detection Models on the Raspberry Pi (with Optional Coral USB Accelerator)*. 2022. URL: https://github.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/blob/master/deploy_guides/Raspberry_Pi_Guide.md. Kasutatud 16.04.2023.

[33] TensorFlow. *TensorFlow Lite*. URL: https://github.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/blob/master/deploy_guides/Raspberry_Pi_Guide.md. Kasutatud 16.04.2023.

[34] Q-engineering. *Q-engineering GitHub*. URL: https://github.com/Qengineering. Kasutatud 18.04.2023.

[35] Q-engineering. *Q-engineering Homepage*. URL: https://qengineering.eu/. Kasutatud 18.04.2023.

[36] Q-engineering. *NanoDet with the ncnn framework*. URL: https://github.com/Qengineering/NanoDet-ncnn-Raspberry-Pi-4. Kasutatud 18.04.2023.

[37] Q-engineering. *YoloFastest V2 with the ncnn framework*. URL: https://github.com/Qengineering/YoloFastestV2-ncnn-Raspberry-Pi-4. Kasutatud 18.04.2023.

[38] Q-engineering. *Deep learning examples on Raspberry 32/64 OS*. 2023. URL: https://qengineering.eu/deep-learning-examples-on-raspberry-32-64-os.html. Kasutatud 18.04.2023.

[39] Q-engineering. *YoloX with the ncnn framework*. URL: https://github.com/Qengineering/YoloX-ncnn-Raspberry-Pi-4. Kasutatud 18.04.2023.

[40] Z. Ge, S. Liu, F. Wang, Z. Li and J. Sun. "YOLOX: Exceeding YOLO Series in 2021". In: *arXiv e-prints* (2021). DOI: 10.48550/arXiv.2107.08430.

[41] Q-engineering. *Install GStreamer 1.18 on Raspberry Pi 4*. 2023. URL: https://qengineering.eu/install-gstreamer-1.18-on-raspberry-pi-4.html. Kasutatud 18.04.2023.

[42] Q-engineering. *Install OpenCV 4.5 on Raspberry Pi 4*. 2022. URL: https://qengineering.eu/install-opencv-4.5-on-raspberry-pi-4.html. Kasutatud 18.04.2023.

[43] Q-engineering. *Install ncnn deep learning framework on a Raspberry Pi 4*. 2023. URL: https://qengineering.eu/install-ncnn-on-raspberry-pi-4.html. Kasutatud 18.04.2023.

[44] Q-engineering. *OpenCV C++ examples on Raspberry Pi 4*. URL: https://qengineering.eu/opencv-c-examples-on-raspberry-pi.html. Kasutatud 18.04.2023.

[45] TensorFlow. *Train a custom object detection model using your data*. Youtube. 2021. URL: https://youtu.be/-ZyFYniGUsw. Kasutatud 18.04.2023.

[46] VideoLAN. *VLC media player*. URL: https://www.videolan.org/vlc/. Kasutatud 18.04.2023.

[47] Heartex Inc. *Label Studio Open Source Data Labeling Platform*. URL: https://labelstud.io/. Kasutatud 18.04.2023.

[48] L. Moreau and M. Kelcey. *Announcing FOMO (Faster Objects, More Objects)*. Edge Impulse Inc. 2022. URL: https://www.edgeimpulse.com/blog/announcing-fomo-faster-objects-more-objects. Kasutatud 18.04.2023.

[49] Edge Impulse Inc. *FOMO: Object detection for constrained devices*. 2022. URL: https://docs.edgeimpulse.com/docs/edge-impulse-studio/learning-blocks/object-detection/fomo-object-detection-for-constrained-devices. Kasutatud 18.04.2023.

[50] Edge Impulse Inc. *Edge Impulse*. URL: https://www.edgeimpulse.com/. Kasutatud 18.04.2023.

[51] Edge Impulse Inc. *Edge Impulse for Linux*. 2023. URL: https://docs.edgeimpulse.com/docs/edge-impulse-for-linux/edge-impulse-for-linux. Kasutatud 18.04.2023.

[52] Edge Impulse Inc. *Data Explorer*. 2022. URL: https://docs.edgeimpulse.com/docs/edge-impulse-studio/data-explorer. Kasutatud 18.04.2023.

[53] *TalTech Robot programming course (ITI0201) robot API*. URL: https://github.com/iti0201/robot. Kasutatud 18.04.2023.

[54] Edge Impulse Inc. *Use Edge Impulse for Linux models from Python*. URL: https://github.com/edgeimpulse/linux-sdk-python. Kasutatud 18.04.2023.

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Anti Lilleaed

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Tehisnärvivõrkudel põhinev objektituvastus õppetöös kasutatavatele robotitele", mille juhendaja on Gert Kanter
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

22.05.2023

[1] Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 - Lõputöö repositoorium

Järgnev GitLab'i repositoorium sisaldab loodud objektituvastuse lahenduse programmikoodi, fotode andmestikke, eksperimentide tulemusi ja muud lõputöö tegemisega seotud informatsiooni:

https://gitlab.cs.ttu.ee/iti0201/object-detection

Lõputöö raames kirjutatud dokumentatsioon on leitav sama repositooriumi Wiki osas:

https://gitlab.cs.ttu.ee/iti0201/object-detection/-/wikis/home