

TALLINNA TEHNIKAÜLIKOOL  
Inseneriteaduskond  
Virumaa kolledž  
Reaal- ja tehnikateaduste keskus

Vladimir Andrianov

**Inimeste voolu automaatne arvestus kasutades  
OpenCV tehisnägemise raamistikku**

Rakendusinfotehnoloogia õppekava lõputöö

Juhendaja: N. Ivleva, lektor

Kohtla-Järve 2019

## AUTORIDEKLARATSIOON

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

---

(kuupäev)

---

(allkiri)

## JOONISTE LOETELU

Joonis 1. Ähmastamise filtri tulemuse näide [5].....	9
Joonis 2. Dilate filtri tulemuse näide [6].....	10
Joonis 3. Erosion filtri tulemuse näide [6].....	10
Joonis 4. Originaalne kaader videost .....	11
Joonis 5. Originaalne kaader pärast MOG2 meetodi kasutamist .....	11
Joonis 6. Originaalne kaader pärast MOG2 ja künnisfiltri kasutamist .....	12
Joonis 7. Kaader kontuuridega ilma künnisfiltrita.....	13
Joonis 8. Kaader kontuuridega kasutades künnisfiltrit .....	14
Joonis 9. Joonistatud kontuur koos väikese punase keskpunktiga.....	15
Joonis 10. Joonistatud kontuur koos trajektooriga.....	15
Joonis 11. Näide inimeste loendurite töö kohta .....	16
Joonis 12. Kontuurid sulasid ühte ning kahel inimesel on üks keskpunkt .....	17
Joonis 13. Kontuurid sulasid ühte.....	18
Joonis 14. Võrdlemise tulemused [13] .....	22
Joonis 15. PiVideoStream meetodi lähtekood [14] .....	22
Joonis 16. Andmete kirjutamine andmebaasile .....	24
Joonis 17. Andmete formaat MongoDB andmebaasis.....	24
Joonis 18. React-Native raamistiku töö erinevate platvormidega [17].....	25
Joonis 19. Mobiilirakenduse avaleht .....	25
Joonis 20. Kuupäeva valiku aken .....	26
Joonis 21. Tulemuste aken.....	26

# SISUKORD

AUTORIDEKLARATSIOON.....	2
JOONISTE LOETELU .....	3
SISSEJUHATUS .....	6
1. NÄGEMISE TEOREETILISED PÕHITÕED .....	7
1.1. Bioloogiline nägemine .....	7
1.2. Arvutinägemine.....	7
1.3. Madalatasemeline ja kõrgetasemeline nägemine .....	7
1.4. OpenCV ja Scikit-image .....	7
2. OPENCV VIDEOVOO TÖÖTLEMINE .....	8
2.1. Pilditöötlus .....	8
2.2. Pildi teisendamine .....	8
2.3. Kasutatud pildi teisendamise OpenCV meetodid .....	8
2.3.1. Gaussian blur meetod.....	9
2.3.2. Dilate ja Erosion meetodid .....	9
2.3.3. MorphologyEx meetod .....	10
2.3.4. CreateBackgroundSubtractorMOG2 meetod.....	10
2.3.5. Threshold meetod .....	12
2.4. Kontuuride leidmine.....	12
2.4.1. Kontuuride joonistamine.....	13
2.4.2. Künnisfiltri kasutamine.....	13
2.5. Tsentroidi leidmine .....	14
2.5.1. Tsentroidi joonistamine .....	14
2.6. Objektide jälgimine.....	15
2.6.1. Liikumispoole määramine .....	16
2.7. Objektide arvestamine.....	16
2.8. Algoritmi piirangud.....	17
3. LAHENDUS.....	18
3.1. Raspberry PI 3.....	18

3.2.	Raspberry PI 3 probleemid ja piirangud .....	18
3.3.	Camera V2.1 .....	19
3.4.	Python 3 ja OpenCV 3 .....	19
3.5.	Raspbian operatsioonisüsteem .....	19
3.6.	Rakenduse pilditöötlemise põhialgoritm.....	20
3.7.	Lahenduse probleemid .....	20
3.8.	Rakenduse optimisatsioon.....	20
3.8.1.	Videovoo suuruse muutmine .....	21
3.8.2.	Kaadriseduse muutmine .....	21
3.8.3.	cv2.VideoCapture(0).....	21
3.8.4.	PiVideoStream() .....	21
3.8.5.	Saavutatud rakenduse kiirus .....	22
4.	STATISTIKA TÖÖTLEMINE .....	23
4.1.	Andmebaas .....	23
4.1.1.	Pilvandmebaasid .....	23
4.1.2.	MongoDB Andmebaas .....	23
4.1.3.	Andmebaasiga ühendamine .....	23
4.1.4.	Andmete kirjutamine andmebaasi .....	24
4.2.	Mobiilirakendus statistika kuvamiseks .....	24
4.2.1.	Mobiilirakendus .....	25
	KOKKUVÕTE .....	27
	ABSTRACT.....	28
	KASUTATUD KIRJANDUS.....	29
	LISAD.....	31

## SISSEJUHATUS

Seoses andmeteaduste valdkonna kiire arenemisega on andmetel tänapäeval oluline roll. Andmeteaduste ning automatiseerimise valdkonnad on väga perspektiivsed suunad ja seal on palju võimalusi ja meetodeid. Mõned andmed muutuvad ebatavaliselt kasulikuks, mis varem ei olnud kellelegi huvitavad või mõningaid andmeid oli raske saada või vanad meetodid olid ebaefektiivsed. Seal tulevad appi masinõpetamine, arvutinägemine ja muud meetodid. Arvuti võib analüüsida mõnesid andmeid reaajas, samaaegselt ning paralleelselt andmeid lugeda, analüüsida ja kirjutada failisse.

Kuna masinõppimine teeb olemasolevate omaduste põhjal ennustused, andmekaevandamine keskendub andmetest uute omaduste leidmisele, siis arvutinägemise teadusharu siirdub videovoo analüüsimiseks, mida ka võib kasutada tehisnärvivõrgu objektide tuvastamiseks või ennustuste tegemiseks.

Virumaa kolledž on avalik koht ja selle külastatavus ning populaarsus näitab tema edu. Selle töö eesmärgiks on luua rakendus, mis kasutab arvutinägemist, et automaatselt loendada inimesi Virumaa kolledžis ja salvestab statistika edasiseks analüüsiks. Näiteks mingi analüüsi rakendus, mis töötleb videovoo veebikaamerast. See töö peab andma lugejale ülevaate ning selgitama, kuidas videovoo ülesandeid lahendada, ning selles töös on antud ka videovoo töötlemise teoreetilised põhitõed ja meetodite võrdlemine.

Selle töö peamiseks probleemiks on tõhusa ja odava lahenduse väljatöötamine kasutades OpenCV raamistikku. See tähendab, et rakendus peab olema efektiivne ja tarbima vähe arvutiressursse ning rakendus peab olema optimeeritud, et rakendusi oleks võimalik käivitada nõrkadel arvutitel. Seal on palju erinevaid lahendusi, mis kasutavad erinevaid algoritme ja seoses sellega on saavutatav tulemus alati erinev. On vaja leida optimaalne lahendus.

Tööks valiti Raspberry Pi 3 miniarvuti, mis on päris odav lahendus. Kuna see miniarvuti on nii kompaktne ja odav, siis sellel ei ole väga palju arvutivõimsust. See tähendab, et see arvuti nõuab rakenduse optimeerimist nii, et see rakendus võib töödelda ja analüüsida videot reaajas. Selles töös on analüüsitud algoritmi sammud, toodud optimeerimise lahendused ja realiseerimise probleemid ning edasiarenduse võimalused.

# 1. NÄGEMISE TEOREETILISED PÕHITÕED

Selles osas on lühidalt kirjeldatud nägemise erinevused ja nende lühike kirjeldus. Bioloogilisel nägemisel ja tehisnägemisel on samad eesmärgid, kuid nad on rakendatud täiesti erinevalt.

## 1.1. *Bioloogiline nägemine*

Inimese silm on väga keeruline elund, aga selle peamised komponendid on sarnased kaamera komponentidega. Silmal on olemas lääts, silmaava, kolvikesed ja kepikesed, mis eristavad värve, ja nägemisnärvid, mis ühendavad silmad ajuga. Inimese silmis kolvikesed eristavad värve punase, roheline ja sinise piirkonnas [1].

## 1.2. *Arvutinägemine*

Arvutinägemine on tehisnägemine, mis tähendab, et see on süsteem, mis emuleerib inimese silma ja aju koostööd. Arvuti kasutab kaamerat silmade asendamiseks, mis koosneb läätsedest ja maatriksist. Kaameral on diafragma, mis töötab nagu silmaava, diafragma avaneb ja sulgub ja kui diafragma on avatud, siseneb tuli kaamerasse. Läätsed keskenduvad valguse maatriksile. Maatriks on suur fotodetektorite kogum, mis muundab valguskiirgust elektrisignaalsiks ja valguskiirgus ei tähenda seda, et see on alati nähtav valgus, võib olla ka infrapunakiirgus [2].

## 1.3. *Madalatasemeline ja kõrgetasemeline nägemine*

Arvutinägemine jaguneb kaheks kihiks: kõrgetasemeline nägemine ja madalatasemeline nägemine. Madalatasemeline nägemine vastutab pilditöötluse, kontuuritöötluse, tagaplaani eemaldamise, muutuste detekteerimise, objekti jälgimise, suuruse muutmise eest jne. Kõrgetasemeline nägemine vastab objektide klassifitseerimisele, objektide asukoha või selle suuruse hindamisele.

## 1.4. *OpenCV ja Scikit-image*

OpenCV (Open Source Computer Vision) ja Scikit-image on avatud raamistikud, mis siirduvad arvutinägemiseks. Need raamistikud sisaldavad palju algoritme ja meetodeid piltide töötlemiseks, näiteks geomeetrilised transformatsioonid, värvide manipuleerimine.

## **2. OPENCV VIDEOVOO TÖÖTLEMINE**

OpenCV (Open Source Computer Vision) – see on avatud lähtekoodiga arvutinägemise ja masinaõpetamise raamistik, mis sisaldab 2500 arvutinägemise ja masinaõpetamise optimeeritud algoritmi [3]. Need algoritmid aitavad tuvastada ja jälgida objekte jne. See on väga populaarne raamistik ja raamistiku loojatelt saadud teabe kohaselt on see raamatukogu juba 14 miljonit korda alla laaditud [3]. OpenCV on saadaval kolmes keeles: C++, Python, ja Java programmiliidese kaudu.

### **2.1. Pilditöötlus**

Arvutinägemine ehk tehisnägemine vajab pilditöötlust. Tehisnägemine ei oma inimintuitsiooni ja inimesed mõnikord alahindavad sellise süsteemi tegelikku keerukust. Pilditöötluse eesmärk on müra ja mittevajaliku informatsiooni eemaldamine piltidest ja videost, et vähendada informatsiooni hulka, mis tuleb analüüsimiseks ja kasutab palju arvutiressursse, näiteks objektide loendamiseks ja jälgimiseks on vaja jälgida ainult objektide kontuuri, mitte tervet pilti. Pilditöötlus sisaldab palju erinevaid meetodeid, nagu tagaplaani kustutamine, muutuste jälgimine, värvide ja kontuuride leidmine ja teisendamine jne.

### **2.2. Pildi teisendamine**

Kui süsteemi eesmärk on piltidelt objektide leidmine, arvestamine ja jälgimine, siis alati tekivad probleemid objektide tuvastamisega. Valgustus, objekti poos, müra, suurus ja muud tegurid mõjuvad tulemustele, näiteks, kui pildis on halb valgustus või objekt on liiga väike või eraldusvõime on liiga väike, siis võib juhtuda nii, et seda objekti arvuti tuvastada ei saa ja see on arvutinägemise süsteemide piirang [4]. Seega saadud tulemus ei ole sajaprotsendiliselt õige ja see tulemus alati omab õigsuse ja vea tõenäosust, see on teine arvutinägemise süsteemide piirang.

### **2.3. Kasutatud pildi teisendamise OpenCV meetodid**

OpenCV raamistik sisaldab palju pildi teisendamise meetodeid, igal meetodil on oma kasutusala ja tööalgoritm, ning saavutatav tulemus on alati erinev.



### ***2.3.1. Gaussian blur meetod***

Gaussian Blur on Gaussi ähmastamise filter, mis on väga populaarne ja mida kasutatakse paljudes rakendustes, mis töötlevad pilte või videoid. Tavaliselt kasutatakse seda filtrit müra vähendamiseks ja see aitab kontuure ühendada üheks kontuuriks. OpenCV Gaussian Blur funktsiooni on lihtne kasutada. Seda on võimalik konfigureerida, sest see võtab mõned parameetrid, näiteks filtri intensiivsus. Kontuuride leidmine on arvuti ressursimahukas protsess ja selle filtri kasutamine kiirendab rakendusi, sest see vähendab müra ning informatsiooni hulka on vähendatud. Selle filtri poolt töödeldud pilt näeb välja nii nagu see pilt oleks tehtud halva teravusega kaameral. Alljärgnev joonis (vt Joonis 1) iseloomustab selle filtri saavutatavat tulemust, paremal poolel on töödeldud pilt, vasakul on originaalpilt.



**Joonis 1. Ähmastamise filtri tulemuse näide [5]**

### ***2.3.2. Dilate ja Erosion meetodid***

Dilate on operatsioon, mis avardab kontuurid ja muudab neid suuremaks (vt Joonis 2). See operatsioon tuleb appi, kui kontuurid on liiga väikesed ja mõnikord ühel inimesel on kaks või rohkem väikeseid kontuure ja selle operatsiooni abil on võimalik kontuure ühendada üheks kontuuriks, ja see näitab, kuidas kontuurid „kasvavad“ [6] ja vastupidine meetod on Erosion, mida kasutatakse müra eemaldamiseks (vt Joonis 3).



**Joonis 2. Dilate filtri tulemuse näide [6]**



**Joonis 3. Erosion filtri tulemuse näide [6]**

### ***2.3.3. MorphologyEx meetod***

MorphologyEx teostab arenenud morfoloogilisi muutusi. See funktsioon võtab järgmised parameetrid: pilt, sihtkoht, struktureeriv element ja operatsiooni tüüp. Kontuuride töötlemiseks on olemas kaks väga kasulikku operatsiooni, need on MORPH\_OPEN ja MORPH\_CLOSE. Need operatsioonid kasutavad Dilate ja Erosion meetodit. Erosion meetod toimib sarnaselt Dilate meetodile, aga see töötleb pilti vastupidi, see vahendab objekte. Pildi töötlemiseks kasutatakse MORPH\_OPEN, et kustutada kogu müra ja MORPH\_CLOSE operatsioon, et täita kontuurid, mis on tühjad.

### ***2.3.4. CreateBackgroundSubtractorMOG2 meetod***

See on algoritm, mis segmenteerib pilte. See meetod rakendab Gaussi segu mudeli tausta lahutamist [7]. Selline meetod säilitab videovoo kaadrite ajalugu ja loob uut pilti, mis sisaldab

ainult muutuseid, näiteks, kui me kasutame sellist meetodit kaamera video analüüsimiseks, siis see meetod genereerib pilte, milles ei ole tagaplaani, ehk seal on ainult liikuvad objektid, mille positsioon on muutunud. See on adaptiivne ja kõige mugavam meetod tagaplaani eemaldamiseks ja liikuvate objektide kontuuride leidmiseks. See meetod genereerib originaalvideost (vt Joonis 4) uut pilti, milles on ainult liikuvate objektide kontuurid (vt Joonis 5).



**Joonis 4. Originaalne kaader videost**



**Joonis 5. Originaalne kaader pärast MOG2 meetodi kasutamist**

Seal on kohe nähtav, et müra on suur probleem ja pärast filtri rakendamist on pildil üleliigsed kontuurid, mida ka arvestatakse süsteemis nagu reaalseid inimesi.

### ***2.3.5. Threshold meetod***

Threshold on künnisfilter, mis on arvutinägemises kõige populaarsem ja kasutatavam. See filter filtreerib objektid, mis tulevad edasi pildisse. Künnisfiltrit kasutatakse SubtractorMOG2 mudelis. See filter takistab väikeste kontuuride ja objektide sissepääsu programmi ja müra kogu väheneb (vt Joonis 6).



**Joonis 6. Originaalne kaader pärast MOG2 ja künnisfiltri kasutamist**

### ***2.4. Kontuuride leidmine***

OpenCV abil on pildis võimalik leida kontuurid, see on funktsioon `cv2.findContours()`, mis tagastab kontuurid andmete massiivina ja neid saab kasutada tsüklis. Funktsioon tagastab kontuurid nagu XY koordinaadi paarid. Need koordinaadid on ristküliku vastaste nurkade koordinaadid ja nad sobivad kontuuride joonistamiseks, sest ristküliku joonistamiseks on vaja kahte punkti.

### 2.4.1. Kontuuride joonistamine

Rakendus tagastab video reaajas, millel on kujutatud objektide kontuurid, et mõista, kuidas programm toimib, et oleks võimalik seda parandada ja vigu märgata. OpenCV funktsioon `cv2.rectangle` tõmbab pildile ristküliku, ja nende joonistamiseks kasutab kahte punkti, mida on võimalik saada kontuuri koordinaatidest.

### 2.4.2. Künnisfiltri kasutamine

Künnisfiltrit kasutatakse kontuuride sorteerimiseks, see võimaldab kasutada rakenduses ainult suuri kontuure, mille pindala on suurem kui mingi minimaalne suurus, see kiirendab rakenduse kiirust ja salvestab arvutiressursse, sest kontuuride edasitöötlemine on ressursimahukas protsess. Näiteks on toodud kaader ilma künnisfiltrita (vt Joonis 7) ja kaader künnisfiltriga (vt Joonis 8).



Joonis 7. Kaader kontuuridega ilma künnisfiltrita



**Joonis 8. Kaader kontuuridega kasutades künnisfiltrit**

## ***2.5. Tsentroidi leidmine***

Tsentroid on objekti kontuuri keskpunkt, mida kasutatakse objekti jälgimiseks. Objekti jälgimiseks on vaja teada, kus asub kontuuri keskpunkt, et oleks mingi punkt, mida on vaja jälgida.

### ***2.5.1. Tsentroidi joonistamine***

OpenCV raamistikus on veel üks funktsioon kontuuride joonistamiseks, `cv2.circle` funktsiooni abil on võimalik võtta tsentroidi koordinaati ja seal kohal joonistada ring, et rakenduse kasutajale oleks arusaadav, kuidas rakendus arvestab inimesi. Joonistatud keskpunkt on nähtav kaadris (vt Joonis 9) .

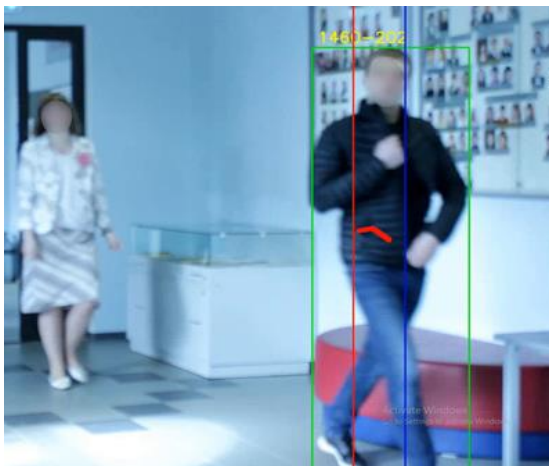


**Joonis 9. Joonistatud kontuur koos väikese punase keskpunktiga**

## **2.6. Objektide jälgimine**

Objektide arvutamiseks on vaja neid jälgida. Kasutades objektide jälgimist, on võimalik seda korrata ilma duplikaatideta ja seejärel jätkata selle jälgimist ning õige number on loenduris.

Objekti jälgimiseks kontuuride keskpunktide koordinaadid salvestatakse massiivisse. Neid koordinaate kasutatakse joonise joonistamiseks, mis iseloomustab objekti liikumise trajektoori (vt Joonis 10).



**Joonis 10. Joonistatud kontuur koos trajektooriga**

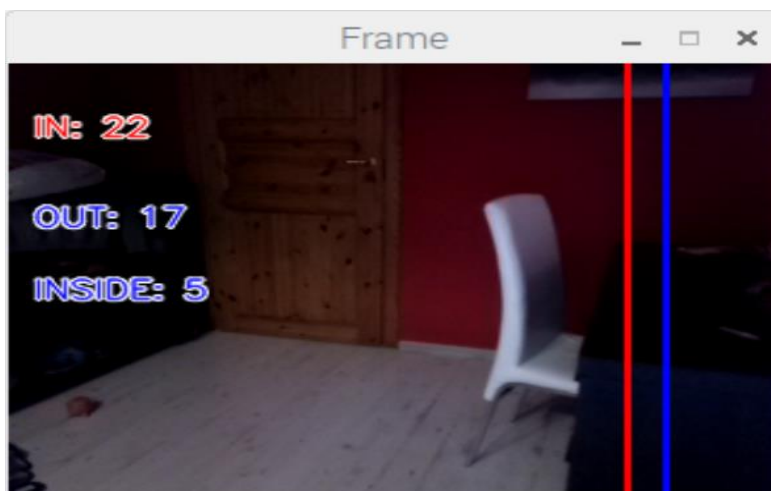
### ***2.6.1. Liikumispoole määramine***

Kuna kontuuri keskpunktide koordinaatide ajalugu on võimalik salvestada, siis on võimalik arvutada, millises suunas inimene liikus, näiteks seda, kas ta läks vasakule või paremale, näiteks kui on olemas kaks piiri, sisenemis- ja väljumispiir ning on võimalik jälgida, millise koordinaadiga tsoonisse sisenen objekt või kui objekt oli juba tsoonist lahkunud, ja nende koordinaatide võrdlemisel on arusaadav, mis suunas objekt liikus, näiteks isik sisenen X koordinaadiga 240 ja väljus X koordinaatiga 290, siis läheb ta paremale või vastupidi.

Rakenduses on määratud kaks piiri ja nad kuvatakse ekraanile vertikaalsete joontena. Kui objekt sisenen lugemispiirkonnasse, salvestatakse iga kaadri keskpunkti koordinaadid ja kui objekt on lugemispiirkonnast lahkunud, võrreldakse esimest ja viimast koordinaati.

### ***2.7. Objektide arvestamine***

Objekti arvestamisel on vaja teada, millal lugemispiirkond on tühi. Rakenduse koodis on olemas globaalne muutuja, milles salvestatakse staatus, kas objekt on tsoonist lahkunud või mitte. Kui objekt veel ei lahkunud tsoonist, ning keskpunkt asub veel lugemispiirkonnas, siis loendurid ei muutu. Kui objekt lahkub tsoonist, siis lähtestatakse objekti koordinaatide ajalugu, tsooni staatus, joon kustutakse jne. Sel hetkel lisab programm loenduritele ühiku, sõltuvalt objekti liikumise suunast, näiteks kui objekt liigub paremale, siis lisatakse ühik esimesele loendurile ja informatsioon kuvatakse ekraanile (vt Joonis 10).



**Joonis 11. Näide inimeste loendurite töö kohta**



## 2.8. Algoritmi piirangud

See objektide jälgimise algoritm on väga efektiivne, kui on vaja loendada objekte, mis liiguvad eraldi. Kui objektid liiguvad üksteisega paarikaupa, siis nende kontuurid sulasid ühte (vt Joonis 11). Kui lugemispiirkonnas üheaegselt on rohkem kui üks objekt, siis arvutatakse need valesti. Seda probleemi saab vältida nurga või asukoha muutmisel.



**Joonis 12. Kontuurid sulasid ühte ning kahel inimesel on üks keskpunkt**

Filtreeritud pildil on näha, et kontuurid sulasid ühte ja see on MOG2 mudeli piirang (vt Joonis 12). Järgnev joonis iseloomustab seda probleemi taustamudelis, milles on nähtavad kõik kontuurid ja on näha, et inimeste kontuurid sulasid ühte ja sellisel juhul ei ole see mudel kõige tõhusam lahendus.



**Joonis 13. Kontuurid sulasid ühte**

### **3. LAHENDUS**

#### ***3.1. Raspberry PI 3***

Raspberry PI 3 on väga arvukate võimalustega mikroarvuti. See on väga väike arvuti, mis võib ühendada sensoreid, andureid, kaamerat jne. Sellesse arvutisse on võimalik installida Linuxi operatsioonisüsteemi ja kasutada seda serverina. See on optimaalne energiatõhususe platvorm arvutinägemise rakenduseks ja seda on võimalik installida ükskõik mis kohas.

#### ***3.2. Raspberry PI 3 probleemid ja piirangud***

Kuna Raspberry PI 3 on odav mikroarvuti, siis on loogiline, et seal ei ole eriti palju operatiivmälu või mingit väga kiiret protsessorit, see on just tasakaalustatud platvorm, mis sobib väikestele ja haridusalastele eesmärkidele, kuna seal ei ole eriti palju arvutiressursse.

Rakendus peab töötleva videovoo reaalajas ja Raspberry PI lahendusega see on päris raske, see on võimalik ainult sel juhul, kui rakendus on hästi optimeeritud. Samuti on suur jahutusprobleem, sest baasversioonis arvuti tuleb ilma radiaatorita ja ventilaatorita. Kui arvuti

töötab pikka aega suure koormuse all, hakkab kiip üle kuumenema ja süsteemi jõudlus langeb järsult.

Raspberry PI 3 protsessor on ehitatud energiasäästlikul arhitektuuril ARM, kuid seetõttu on olemas piirang, et tavalised x86 ja x64 programmid sellele protsessorile ei sobi ja on vaja otsida ARM raamistiku versiooni, mis selle protsessoriga töötab [8].

### **3.3. *Camera V2.1***

Raspberry PI 3 mikroarvutiga on võimalus ühendada kaamerat, selleks on spetsiaalne kaamera, mida kasutatakse Raspberry PI mikroarvutiga. See kaamera on pisike ja võib pildistada videot Full HD eraldusvõimega 30 kaadrit sekundis, kaamera toetab ka teisi eraldusvõimeid, näiteks 640x480 90 kaadrit sekundis [9]. Kaamera töötab koodiga `imutils.video.pivideoostream` raamistiku abil. Kaamera ühendamiseks arvutiga tuleb see ühendada kaabliga, kaamera ei vaja täiendavat võimsust ega täiendavaid kaableid.

### **3.4. *Python 3 ja OpenCV 3***

Python on andmeteaduses väga populaarne programmeerimiskeel, seal on palju instruksioone, materjale ja see keel on tasuta ega nõua midagi programmi käivitamiseks, kõik, mis on vaja, on konfigureeritud operatsioonisüsteem, milles Pythoni keskkond peab olema installeeritud.

Töös kasutatakse OpenCV3 ja Python3 versioone. OpenCV3 on kiirem ja stabiilsem kui OpenCV2 ja samamoodi on Python-iga 2 ja Python-iga 3 [10].

### **3.5. *Raspbian operatsioonisüsteem***

Raspberry PI arvuti töötab Raspbian Linux distributiiviga, see distributiiv on mõeldud kasutamiseks Raspberry PI arvutiga [11]. Raspbian operatsioonisüsteemis on integreeritud kasutajaliides, mõned baasrakendused ja see on juba kasutamiseks valmis.

### ***3.6. Rakenduse pilditöötlemise põhialgoritm***

Rakendus võtab videovoo ja loeb iga kaadrit eraldi. Tagaplaani eemaldamiseks kasutatakse MOG2 meetodit ja selle künnisfiltrit, nende abil eemaldatakse kaadrist müra. Järgmises sammus rakendus loeb kontuurid kaadrist ja töötleb neid tsüklis. Selles tsüklis töötleb rakendus kontuuride suurused ja künnisfiltri abil kasutatakse ainult suurte objektide kontuure. Kui objekt ning objekti kontuur sisenes lugemispiirkonda ja hetkel asub seal, siis rakendus arvutab kontuuride keskpunkte ja säilitab keskpunkte koordinaadi mälusse ja joonistab joone, mis näitab parema mõistmise huvides objekti liikumist trajektooriga. Kui objekt lahkub lugemispiirkonnast, siis rakendus arvestab, mis suunas objekt liikus. Kui objekt liikus vasakule, siis see lisab ühiku loendurisse, milles on inimeste arv, kes läks vasakule ja vastupidi, kui objekt liikus paremale, siis see lisab ühiku loendurisse, mis salvestab inimeste arvu, kes läksid paremale, veel rakendus säilitab andmebaasis hetke aega ja objekti liikumise suunda. Järgmiseks sammuks arvutatakse loendurite vahe ning inimeste arv hoones (vt Lisa 1).

Lühidalt öeldes võetakse iga videokaader eraldi, taust on välja lõigatud kasutades MOG2 mudeli meetodit, saadakse edasi ainult liikuvate objektide kontuurid, mille positsioonid ning asukohad on muutunud. Seejärel töötleb rakendus objektide kõiki kontuure ja asukohti, loeb inimesi, võtab mälust olevad andmed ja joonistab kontuurid, lõppsammuks kõik informatsioon, nagu inimeste arv hoones, liinid ja kontuurid kuvatakse ekraanile ja andmed salvestatakse andmebaasile.

### ***3.7. Lahenduse probleemid***

Saavutatud tulemusena on rakendus, mis töötab Raspberry PI 3 mikroarvutis. Selles rakenduses juba kasutatakse künnisfiltri meetodit, aga see rakendus töötab ikka aeglaselt ja reaajas see ei saa töödelda videot. Seega arvuti töötab pikka aega suure koormuse all ja hakkab üle kuumenema ning veelgi aeglasemalt töötama.

### ***3.8. Rakenduse optimisatsioon***

Rakenduse optimisatsiooniks on kasutatud muud meetodit kaamera andmete lugemiseks, sest standardne meetod on sünkroon ja aeglane, kuid teine meetod on asünkroon ja töötab kiiremini.

Veel üheks probleemiks on üleliigselt suur kaadri suurus, mis ei ole vajalik kontuuride tuvastamiseks ja ülemäärane kaadrisagedus.

### ***3.8.1. Videovoo suuruse muutmine***

Programmi kiirendamiseks on vaja vähendada sisendandmete suurust ja see on üks kõige tõhusamaid meetodeid, kuna programmi kiirus sõltub andmete hulgast. Andmete koguse ja rakenduse kiiruse vahel peab olema tasakaal, sest kui pilt on liiga suur, siis rakendus töötab aeglaselt, ning kui pilt liiga väike, siis rakendus ei saa midagi kaadris eristada.

### ***3.8.2. Kaadrisageduse muutmine***

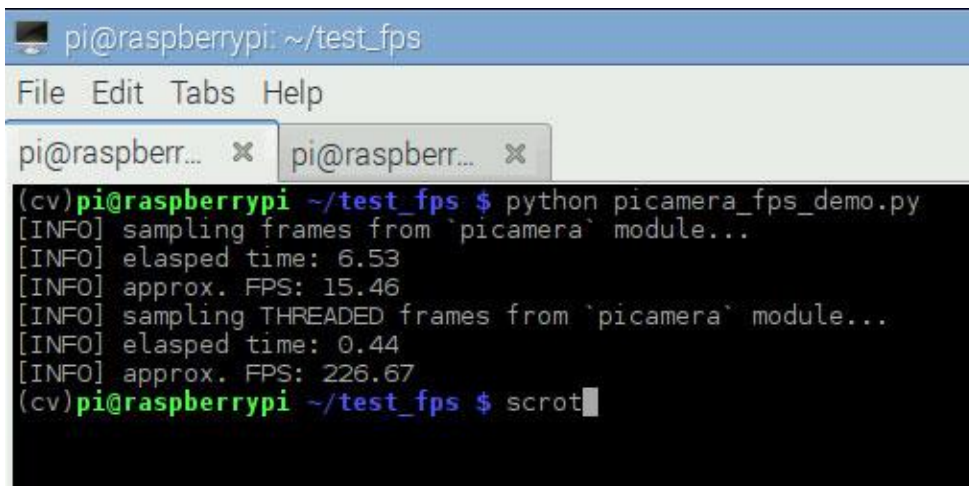
Programmi kiirendamiseks on veel vaja muuta kaadrisagedust, sest inimeste arvestuseks ei ole vaja suurt kaadrisagedust, näiteks kui kaadrisagedus on rohkem kui sada kaadrit sekundis, siis see on just üleliigne informatsioon. Samuti kaadrisagedusega peab olema tasakaal, et kaadrisagedus oleks liikumise tuvastamiseks piisav.

### ***3.8.3. cv2.VideoCapture(0)***

See on standardne baasmeetod, mis loeb andmeid kaamerast. Arv null näitab, et see meetod võtab video esimesest kaamerast [12]. See meetod on sünkroon ja see ei ole efektiivne.

### ***3.8.4. PiVideoStream()***

PiVideoStream on väga efektiivne asünkroon meetod ja see võimaldab lugeda informatsiooni videokaamerast kümme korda (vt Joonis 13) kiiremini ja suuremini [13].



```
pi@raspberrypi: ~/test_fps
File Edit Tabs Help

pi@raspberr... x pi@raspberr... x

(cv)pi@raspberrypi ~/test_fps $ python picamera_fps_demo.py
[INFO] sampling frames from `picamera` module...
[INFO] elapsed time: 6.53
[INFO] approx. FPS: 15.46
[INFO] sampling THREADED frames from `picamera` module...
[INFO] elapsed time: 0.44
[INFO] approx. FPS: 226.67
(cv)pi@raspberrypi ~/test_fps $ scrot
```

**Joonis 14. Võrdlemise tulemused [13]**

See meetod on juba optimeeritud, seal on pildi suuruse muutmine, mis muudab pildi suurust 320x240 pikslik, mis on liikumise tuvastamiseks piisav, veel on olemas kaadrisageduse muutmine ja nende meetodi kasutamine on nähtav lähtekoodis (vt Joonis 14) [14].

```
class PiVideoStream:
    def __init__(self, resolution=(320, 240), framerate=32):
        # initialize the camera and stream
        self.camera = PiCamera()
        self.camera.resolution = resolution
        self.camera.framerate = framerate
```

**Joonis 15. PiVideoStream meetodi lähtekood [14]**

### **3.8.5. Saavutatud rakenduse kiirus**

Optimeeritud rakendus töötab mitu korda kiiremini kui mitteoptimeeritud rakendus. Koos kõikide meetoditega töötleb see rakendus videovoo reaajas ja ühe kaamera kasutab see ainult 30% arvutiressursse. Kui kiip ei hakka üle kuumenema ja süsteemi jõudlus ei lange, siis see töötab stabiilselt. Teoreetiliselt miniarvuti saab töödelda kahte videot üheaegselt, aga selleks on vaja parandada jahutust.

## **4. STATISTIKA TÖÖTLEMINE**

### ***4.1. Andmebaas***

Statistika salvestamiseks kasutatakse andmebaase. Kui rakendusega midagi juhtub, siis andmed on kadunud, aga andmebaasiga seda probleemi ei ole, sest andmebaas säilitab andmeid failis ja andmebaasid on stabiilsed rakendused.

#### ***4.1.1. Pilvandmebaasid***

Pilvandmebaasid on uus tehnoloogia ja need ei ole just klassikalised andmebaasid, vaid mingi hosting protsess, ja nende eelis on see, et kasutaja ei pea muretsema selle andmebaasi haldamise või kasvamise pärast [15].

Raspberry PI arvutil on vähe arvutiressursse ja pilvandmebaaside kasutamine on kõige efektiivsem lahendus, sest sel juhul ei pea muretsema andmebaasi suuruse pärast jne.

#### ***4.1.2. MongoDB Andmebaas***

MongoDB pakub tasuta pilvandmebaasi, mida saab kasutada tasuta ja mis töötab mongoDB serveris. MongoDB on avatud lähtekoodiga andmebaas. MongoDB pakub igavesti tasuta andmebaasi, aga seal on mõned piirangud, näiteks maksimaalne andmebaasi maht on 512 megabaiti, aga see maht on piisav, kui on vaja ainult statistikat säilitada [16]. Rakendus säilitab andmebaasile ainult aja ja liikumise suunda, tekstivormis see on pisike informatsiooni maht.

#### ***4.1.3. Andmebaasiga ühendamine***

Kuna statistika säilitamiseks on valitud pilvandmebaas, siis selle andmebaasiga ühendamine on natuke keeruline. Kuna see ei ole just lokaalne andmebaas, siis see vajab serveriga ühendamiseks internetiühendust. Serveriga ühendamiseks on vaja konfigureerida koodis andmebaasi, selleks on vaja teada serveri IP-aadressi ja porti.

#### 4.1.4. Andmete kirjutamine andmebaasi

MongoDB säilitab andmeid JSON formaadis. Andmete lisamiseks on olemas standardsed meetodid, aga andmed peavad olema JSON formaadis, ning andmebaas tagastab samuti andmeid JSON formaadis. Järgmine joonis näitab, et rakendus võtab aja ja liikumise suuna ja salvestab need andmed andmebaasis.

```
time := datetime.now().strftime('%Y-%m-%d %H:%M')
mydict := {"direction": "out", "time": time}
x := collection.insert_one(mydict)
```

#### Joonis 16. Andmete kirjutamine andmebaasile

Järgmine joonis näitab, kuidas andmed on salvestatud MongoDB andmebaasile.



The screenshot displays three documents stored in a MongoDB database. Each document is shown in a separate entry, separated by horizontal lines. The documents are as follows:

- Document 1: `_id: ObjectId("5c97f25bad889d11c06dac9a")`, `direction: "in"`, `time: "2019-03-24 23:10"`
- Document 2: `_id: ObjectId("5c97f25ead889d11c06dac9b")`, `direction: "in"`, `time: "2019-03-24 23:10"`
- Document 3: `_id: ObjectId("5c97f25fad889d11c06dac9c")`, `direction: "out"`, `time: "2019-03-24 23:10"`

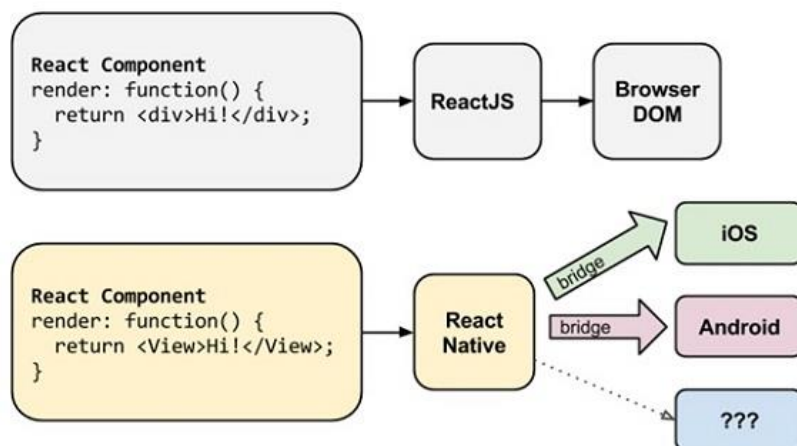
#### Joonis 17. Andmete formaat MongoDB andmebaasis

### 4.2. Mobiilirakendus statistika kuvamiseks

Kuna andmebaas ja andmed on olemas, siis on võimalik need andmed kasutajale kuvada. Statistika kuvamiseks on loodud mobiilirakendus. Mobiilirakendus on loodud kasutades React-Native raamistikku, see tähendab, et see on hübriidrakendus, mis on kirjutatud JavaScript



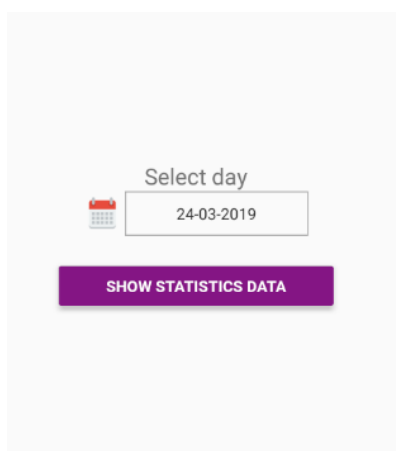
keeles. React-Native raamistiku eelis on see, et sama kood töötab Android, iOS ja muu mobiili operatsioonisüsteemides (vt Joonis 17).



**Joonis 18. React-Native raamistiku töö erinevate platvormidega [17]**

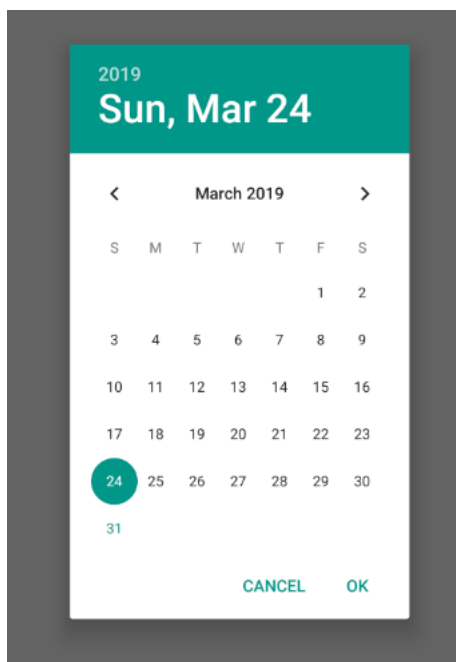
#### **4.2.1. Mobiilirakendus**

Mobiilirakendus on väike, kuid täidab oma ülesannet. Mobiilirakendus on ühendatud sama andmebaasiga, mida kasutab rakendus Raspberry PI arvuti (vt Lisa 2). Mobiilirakenduses on vaja valida päev, millest kasutaja on huvitatud. Järgmine joonis näitab rakenduse avalehte.



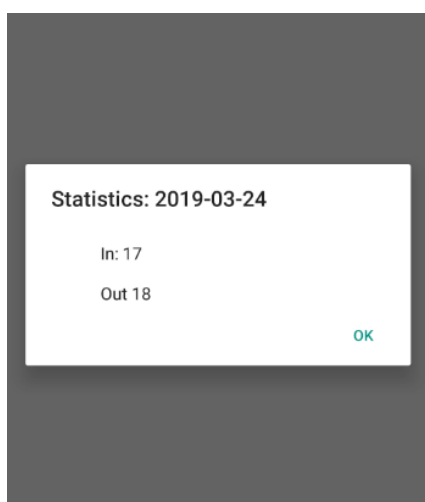
**Joonis 19. Mobiilirakenduse avaleht**

Avalehes on suur nupp ja kuupäev, kuupäevale klõpsates avatakse vorm, kus on vaja valida kuupäev. Järgmine joonis näitab kuupäeva valiku vormi.



## Joonis 20. Kuupäeva valiku aken

Kui kuupäev on valitud, siis nupule klõpsates avatakse vorm, mis näitab statistikat valitud päevas. Järgmine joonis näitab tulemuste aken, milles on kuupäev ja selle päeva statistika.



## Joonis 21. Tulemuste aken

## KOKKUVÕTE

Arvutinägemise ülesannete arv kasvab tänapäeval pidevalt, ja üks nendest arvutinägemise probleemidest on autoriga lahendatud ning kirjeldatud käesolevas lõputöös. Käesoleva töö eesmärgiks oli uurida arvutinägemise võimalusi ja luua rakendusi inimeste loendamiseks, mis peab töötama Raspberry pi mikroarvutis reaalajas. Lisaks see rakendus peab kirjutama andmeid andmebaasi ning oli vaja luua ka mobiilirakendus statistika kuvamiseks.

Rakendus oli loodud Python programmeerimiskeeles kasutades OpenCV tehisinägemise raamistikku ja optimeeritud, veel oli loodud mobiilirakendus JavaScript programmeerimiskeeles kasutades React Native raamistikku ja MongoDB andmebaas. Lisaks oli kirjeldatud arvutinägemise põhitõed ja videovoo analüüsimise meetodid ja selleks oli toodud näited reaalsest rakendusest. Kõik ülaltoodud eesmärgid on täielikult rakendatud.

Kuna töös kasutatavad algoritmid ei ole ideaalsed, võib olla parem meetodite kombinatsioon ja parem algoritmid, seega peaks arvutinägemise rakenduse ja mobiilirakenduse arendamine jätkuma ka tulevikus, et parandada tulemuste täpsust.

Lahenduse edasiseks arendamiseks on vaja muuta objekti jälgimiskoodi nii, et samaaegselt saab jälgida mitmeid liikuvaid objekte, mis asuvad lugemispiirkonnas ja samuti tuleb muuta algoritmi nii, et kontuurid ei ühenduks üheks kontuuriks, kui mitu inimest lähevad ligistikku ja lisaks tuleb arendada mobiilirakendust nii, et see oleks mingi statistika analüüsimise platvorm.

Tulevikus tuleb luua tehisnärvivõrgu mudel ja kasutada seda isikute tuvastamiseks või kasutada masinõpetamise algoritmi kolledži külastatavuse kohta ennustuste tegemiseks ja tuleb veel lisada nende tulemuste kuvamine mobiilirakendusse.

Kuna tehisnärvivõrgud on väga ressursimahukad, siis nende kasutamiseks tuleb kasutada kiiremat arvutit või ühendada rakendus mingi serveriga, milles kõik andmed töödeldakse.

## ABSTRACT

The number of computer vision tasks is constantly increasing today, and one of these problems with computer vision has been solved by the author and described in this thesis. The purpose of this work was to explore the possibilities of computer vision and to create an application for counting people, which should run in real time on the Raspberry Pi computer. Additionally, this application needs to write data into a database and it was also necessary to create a mobile app to display statistics.

The application was written in the Python programming language using the OpenCV artificial vision framework and optimized, also were made the mobile application which was written in JavaScript programming language using the React Native framework and the MongoDB database. In addition, the basics of computer vision and methods of analyzing video streams were described and examples of a real application were provided for this purpose. All the above objectives are fully implemented.

Since the algorithms used in the work are not ideal, there may be a better combination of methods and better algorithms, so the development of the computer vision application and the mobile application should continue in the future to improve the accuracy of the results.

For further development of the solution, it is necessary to change the object tracking code so that several moving objects located in the monitoring area can be tracked at the same time, and the algorithm must also be changed so that the contours do not merge into one contour when people are going close to each other, and in addition the mobile application needs to be developed so it will be a statistics analyzing platform.

In the future, an artificial neural network model should be created and used to identify individuals, or to use a machine learning algorithm to make predictions on college attendance and display of results should be added to the mobile application.

Because artificial neural networks are very resource-intensive, a faster computer should be used for using them or application may be connected to a server where all the data will be processed.

## KASUTATUD KIRJANDUS

1. Värvide nägemine. Tikkurila. (03.03.2012) [WWW] [https://www.tikkurila.ee/ehitus-ja\\_remontvarvid/varvitoonid/varvidest/varvide\\_nagemine](https://www.tikkurila.ee/ehitus-ja_remontvarvid/varvitoonid/varvidest/varvide_nagemine) (02.03.2019).
2. How to Make a Digital IR Counter to Count Door Entries. Mitchell, R. (05.07.2017) [WWW] <https://maker.pro/pcb/projects/count-door-entries-digital-ir-counter> (02.03.2019).
3. About. OpenCV. [WWW] <https://opencv.org/about.html> (02.03.2019).
4. Image Data Pre-Processing for Neural Networks. Nikhil, B. (11.09.2017) [WWW] <https://becominghuman.ai/image-data-pre-processing-for-neural-networks-498289068258> (03.03.2019).
5. Image Smoothing using OpenCV Gaussian Blur. TutorialKart. [WWW] <https://www.tutorialkart.com/opencv/python/opencv-python-gaussian-image-smoothing/> (03.03.2019).
6. Eroding and Dilating. OpenCV. (12.07.2018). [WWW] [https://docs.opencv.org/2.4.13.7/doc/tutorials/imgproc/erosion\\_dilatation/erosion\\_dilatation.html](https://docs.opencv.org/2.4.13.7/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html) (09.03.2019).
7. cv::BackgroundSubtractorMOG2 Class Reference. OpenCV. (22.12.2017). [WWW] [https://docs.opencv.org/3.4/d7/d7b/classcv\\_1\\_1BackgroundSubtractorMOG2.html](https://docs.opencv.org/3.4/d7/d7b/classcv_1_1BackgroundSubtractorMOG2.html) (09.03.2019).
8. RASPBERRY PI 3: SPECS, BENCHMARKS & TESTING. The MagPi. (2016). [WWW] <https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/> (09.03.2019).
9. Camera Module. The MagPi. [WWW] <https://www.raspberrypi.org/documentation/hardware/camera/> (09.03.2019).
10. OpenCV 3.0. OpenCV. (04.06.2015). [WWW] <https://opencv.org/opencv-3-0.html> (10.03.2019).

11. Welcome to Raspbian. Raspbian. [WWW] <https://www.raspbian.org/> (10.03.2019).
12. Getting Started with Videos. OpenCV. [WWW] [https://docs.opencv.org/3.0-beta/doc/py\\_tutorials/py\\_gui/py\\_video\\_display/py\\_video\\_display.html](https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_gui/py_video_display/py_video_display.html) (10.03.2019).
13. Increasing Raspberry Pi FPS with Python and OpenCV. Rosebrock, A. (28.12.2015). [WWW] <https://www.pyimagesearch.com/2015/12/28/increasing-raspberry-pi-fps-with-python-and-opencv/> (15.03.2019).
14. pvideostream.py. Rosebrock, A. (09.01.2016). [WWW] <http://pvideostream.py>  
<https://github.com/jrosebr1/imutils/blob/master/imutils/video/pvideostream.py> (17.03.2019).
15. Understanding 4 Database Types, for Ecommerce. billy24, (15.09.2016). [WWW] <https://techadminlab.com/understanding-4-database-types-ecommerce/?q=%2Fet%2Funderstanding-4-database-types-ecommerce%2F> (18.03.2019).
16. Simple, All-inclusive Pricing. mLab. [WWW] <https://mlab.com/plans/pricing/#plan-type=sandbox> (18.03.2019).
17. Writing Cross-Platform Apps with React Native. Eisenman, B. (25.02.2016). [WWW] <https://www.infoq.com/articles/react-native-introduction> (27.03.2019).

# LISAD

## LISA 1. Python rakendus

```
from collections import deque
from picamera.array import PiRGBArray
from picamera import PiCamera
from threading import Thread
from imutils.video.pivideostream import PiVideoStream
from pymongo import MongoClient
from datetime import datetime

import time
import numpy as np
import cv2
import time
import imutils

#connect DB for writing the data into database
client =
MongoClient("mongodb+srv://#####.gcp.mongodb.net
/test?retryWrites=true")
mydb = client["stats"]
collection = mydb["virumaa"]

#centroid points used for drawing the line
pts = deque()
#points for determining the object movement direction
ptsXPoints = deque()

#initialization of variable to hold movement direction
dirX = ""

#boolean variable to check whether object has been already counted
isAlreadyCounted = False

#counter variables initialization
#people got in
cnt_in = 0
#people got out
cnt_out = 0
#people inside
inside_count = 0
```

```

#multithread camera reading method instead of capture(0) which is way slower
vs = PiVideoStream().start()

#time for camera start-up
time.sleep(2.0)

videoWidth = 320#video width used to locate the counting area borders
h = 240#height

#video surface area used for objects thresholding
frameArea = h*videoWidth
#object size threshold depending on the video size
areaTH = frameArea/50

#left red line
line_left = int(videoWidth*0.8)
#right blue line
line_right = int(videoWidth*0.85)

#colors used to fill lines
line_right_color = (255,0,0) #red
line_left_color = (0,0,255) #blue

#two points used for the blue line
pt1 = [line_right, 0];
pt2 = [line_right, videoWidth];
pts_L1 = np.array([pt1,pt2], np.int32)
pts_L1 = pts_L1.reshape((-1,1,2))

#two points used for the red line
pt3 = [line_left,0];
pt4 = [line_left, videoWidth];
pts_L2 = np.array([pt3,pt4], np.int32)
pts_L2 = pts_L2.reshape((-1,1,2))

#MOG2 subtractor
fgbg = cv2.createBackgroundSubtractorMOG2(detectShadows = True)

#opening transformation
kernelOp = np.ones((3,3),np.uint8)
#closing transformation
kernelCl = np.ones((11,11),np.uint8)

#font
font = cv2.FONT_HERSHEY_SIMPLEX

```



```

#color for coordinates
color_yellow = (0,255,255)

#opposite of erosion, dilate using vertical rectangle element
#used to fix contours when people got divided into 2 vertical contours
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (2,4))

#main video cycle
while 1:
    #taking video frame from the camera
    frame = vs.read()

    #removing background using MOG2 substractor
    fgmask = fgbg.apply(frame)

    #reducing the noise using GaussianBlur
    fgmask = cv2.GaussianBlur(fgmask, (3, 3), 0)

    #binarization
    try:
        ret,imBin2 = cv2.threshold(fgmask,200,255,cv2.THRESH_BINARY)
        #reducing noise with opening and closing transformation
        mask2 = cv2.morphologyEx(imBin2, cv2.MORPH_OPEN, kernelOp)
        mask2 = cv2.morphologyEx(mask2, cv2.MORPH_CLOSE, kernelCl)

        #making contours bigger using rectangle element
        mask2 = cv2.dilate(mask2,kernel,iterations = 2)

    #exception handling, print all data to console in case of fail
    except:
        print ('GET IN:',cnt_in)
        print ('GET OUT:',cnt_out)
        break

    #returning external objects contours
    (_,contours, _) =
cv2.findContours(mask2,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

    for cnt in contours:
        #taking each contours area for threshold filter
        area = cv2.contourArea(cnt)

        #image moment, used to find object's centroid coordinates
        #m00, zero-order moment is amount of dots making the spot(people's
        contour)

```

```

#m10, 10 order moment is sum of X dots coordinates
#m01, 01 order moment is sum of Y dots coordinates
#taking all the moments from the contour
M = cv2.moments(cnt)
#taking information from the image moments
m00 = M['m00']#amount of dots
x = M['m10']#sum of x dots coordinates
y = M['m01']#sum of y dots coordinates
#getting average X and Y coordinates of all points
cx = int(x/m00)#X centroid coordinate
cy = int(y/m00)#Y centroid coordinate

#if contours area is more than threshold area AND if object is in the
hall (not outside)
#start tracking and showing objects borders (for better visualization)
if ((area > areaTH) and int(M["m10"] / M["m00"])>(videoWidth*0.5)):

    #taking objects borders for drawing
    x,y,w,h = cv2.boundingRect(cnt)

    #drawing of centroid
    center = (cx,cy)
    cv2.circle(frame,(center), 5, (0,0,255), -1)
    cv2.putText(frame, "%d-%d" % (x,y), (x+10,y-10), font, 1,
color_yellow, 2)
    #drawing objects borders/contours as a rectangle
    img = cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)

    #tracking X coordinate to check if object has crossed the line
    centroidHorizontalPosition = cx

    #if there's an people coming to the counting area then it resets
variables
    #it also prints from which side of the screen the object appeared
    if(centroidHorizontalPosition>(videoWidth*0.75) and
centroidHorizontalPosition<(videoWidth*0.8)):
        print("from left to right")
        isAlreadyCounted = False;
        pts.clear()
        ptsXPoints.clear()

    #same logic for the opposite direction
    if(centroidHorizontalPosition>(videoWidth*0.85) and
centroidHorizontalPosition<(videoWidth*0.9)):
        print("from right to left")

```

```

        isAlreadyCounted = False;
        pts.clear()
        ptsXPoints.clear()

        #When object is in the counting area it starts filling his
movement history for drawing lines
        #centroid's points history in the calculating area is also used
for determining the movement direction
        if(centroidHorizontalPosition>(videoWidth*0.8) and
centroidHorizontalPosition<(videoWidth*0.85)):
            pts.appendleft(center)
            ptsXPoints.appendleft(centroidHorizontalPosition)

        # loop over the set of centroid tracked points
        for i in np.arange(1, len(ptsXPoints)):

            # check if enough points were accumulated in the buffer
            if ( len(pts)>1):

                #taking first and last centroid positions
                #[0] is a first element in the queue
                #[-1] is a last moment in the queue
                firstX = ptsXPoints[0]
                lastX = ptsXPoints[-1]

                #checking X coordinates to determine the correct
movement direction
                if ( firstX < lastX):
                    dirX = "ToRight"

                if( firstX > lastX):
                    dirX = "ToLeft"

                # compute the thickness of the line and
                # draw the connecting lines
                thickness = int(np.sqrt(32 / float(i + 1)) * 2.5)
                cv2.line(frame, pts[i - 1], pts[i], (0, 0, 255),
thickness)

        #when everything is calculated it checks for the final direction and
appends values
        #if object is not yet calculated then it works and sets the flag to true
        #it writes data to the cloud database
        if(dirX == "ToRight" and isAlreadyCounted == False):
            isAlreadyCounted = True

```

```

        cnt_in +=1
        inside_count +=1
        time = datetime.now().strftime('%Y-%m-%d %H:%M')
        mydict = { "direction": "in", "time": time}
        x = collection.insert_one(mydict)

    if(dirX == "ToLeft" and isAlreadyCounted == False):
        isAlreadyCounted = True
        cnt_out +=1
        inside_count -=1
        time = datetime.now().strftime('%Y-%m-%d %H:%M')
        mydict = { "direction": "out", "time": time}
        x = collection.insert_one(mydict)

#reset direction for the next object
dirX = ""

#on-screen printing results
str_in = 'IN: ' + str(cnt_in)
str_out = 'OUT: ' + str(cnt_out)
str_inside = 'INSIDE: ' + str(inside_count)
#drawing visual lines for better understanding
frame = cv2.polylines(frame,[pts_L1],False,line_right_color,thickness=2)
frame = cv2.polylines(frame,[pts_L2],False,line_left_color,thickness=2)
cv2.putText(frame, str_in ,(10,40),font,0.5,(255,255,255),2,cv2.LINE_AA)
cv2.putText(frame, str_in ,(10,40),font,0.5,(0,0,255),1,cv2.LINE_AA)
cv2.putText(frame, str_out ,(10,90),font,0.5,(255,255,255),2,cv2.LINE_AA)
cv2.putText(frame, str_out ,(10,90),font,0.5,(255,0,0),1,cv2.LINE_AA)
cv2.putText(frame, str_inside
,(10,130),font,0.5,(255,255,255),2,cv2.LINE_AA)
cv2.putText(frame, str_inside ,(10,130),font,0.5,(255,0,0),1,cv2.LINE_AA)

#showing the video and the mask
cv2.imshow('Frame',frame)
cv2.imshow('Mask2',mask2)

#looping over the video flow
k = cv2.waitKey(30) & 0xff
if k == 27:
    break

cap.release()
cv2.destroyAllWindows()

```

## LISA 2. Mobiilirakendus

```
import React, { Component } from 'react';
import { Alert, Text, View, StyleSheet, Button } from 'react-native';
import DatePicker from 'react-native-datepicker';
import { Stitch, AnonymousCredential, BSON } from 'mongodb-stitch-react-native-sdk';
const MongoDB = require('mongodb-stitch-react-native-services-mongodb-remote');

export default class MyDatePicker extends Component {
  constructor(props) {
    super(props);
    this.state = {
      client: 'null',
      countIn: 0,
      countOut: 0,
      date: '2019-03-24',
    };
    this._loadClient = this._loadClient.bind(this);
  }

  componentDidMount() {
    this._loadClient();
  }

  render() {
    var client = this.state.client;
    return (
      <View style={{ flex: 1, justifyContent: "center", alignItems: "center" }}>

        <View>
          <Text style={{fontSize: 20}}>Select day</Text>
        </View>

        <DatePicker
          style={{ width: 200 }}
          mode="date" //The enum of date, datetime and time
          placeholder="select date"
          format="YYYY-MM-DD"
          confirmBtnText="Confirm"
          cancelBtnText="Cancel"
          minDate="2019-01-01"
        />
      </View>
    );
  }
}
```

```

maxDate="2030-01-01"
date={this.state.date} //initial date from state
customStyles={{
  dateIcon: {
    position: 'absolute',
    left: 0,
    top: 4,
    marginLeft: 0,
  },
  dateInput: {
    marginLeft: 36,
  },
}}
onDateChange={date => {
  this.setState({ date: date });
}}
/>

<View>
<Text style={{fontSize: 20}}></Text>
</View>

<Button

  onPress={() => {
    if(client.auth.isLoggedIn) {
      const dbClient =
client.getServiceClient(MongoDB.RemoteMongoClient.factory, "mongodb-atlas");
      this.setState({ currentUserid: client.auth.user.id })
      const db = dbClient.db('stats');
      const statDetails = db.collection('virumaa')

      var cntIn = 0;
      var cntOut = 0;
      var date = this.state.date;

      statDetails.count({
        $and: [
          {time: new BSON.BSONRegExp(date, 'i') },
          {direction: new BSON.BSONRegExp('in', 'i') }
        ]
      })
      .then(numDocs =>
        cntIn = numDocs
      )
    }
  }
}

```

```

        .catch(err => console.error("Failed to count documents: ",
err))

        statDetails.count({
            $and: [
                {time: new BSON.BSONRegExp(date, 'i') },
                {direction: new BSON.BSONRegExp('out', 'i') }
            ]
        })
        .then(numDocs =>
            cntOut = numDocs
        )
        .catch(err => console.error("Failed to count documents: ",
err))

        setTimeout(function(){
            total = cntIn + cntOut;
            Alert.alert(`Statistics: ${date}`, `
                In: ${cntIn} \n
                Out: ${cntOut}`)
        }, 2000);
    }}}

    title="          Show statistics data          "
    color="#841584"
    size={30}
    accessibilityLabel="Show amount of the people in chosen period"
    />

</View>
);
}

_loadClient() {
    Stitch.initializeDefaultAppClient('statsdiploma-bcbzo').then(client => {
        this.setState({ client });
    });
}
}

const styles = StyleSheet.create({
    container: {
        flex: 1,

```

```
    alignItems: 'center',  
    justifyContent: 'center',  
    marginTop: 50,  
    padding: 16,  
  },  
});
```