TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Anee Mändmets 213431IACB

Mikrokontrollerite ja programmeerimiskeelte võrdlus videopildi viite mõõtmisel

Bakalaureusetöö

Juhendaja: Priit Roosipuu
Magistrikraad

Kaasjuhendaja: Peeter Ellervee
Doktorikraad

Tallinn 2024

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: Anee Mändmets

13.05.2024

Annotatsioon

Antud lõputöö käsitleb eelnevalt projekti raames koostatud videopildi viite mõõtmiseks valminud süsteemi jaoks välja valitud mikrokontrollerite ning programmeerimiskeelte võrdlemist. Töö tulemusena valitakse võrreldavate seast välja sobivaim keele ja mikrokontrolleri kombinatsioon.

Valitud mikrokontrolleriteks on Raspberry Pi 5, Raspberry Pi 4B, ESP32-PoE-ISO ja Arduino Mega. Viimane jäetakse lõpuks võrdlusest välja, sest see ei võimalda antud süsteemis vajalikku funktsionaalsust. Valitud programmeerimiskeelteks on Python ja C.

Töö käigus testitakse Raspberry Pi-de erinevate sisend/väljund viikute lugemise ja kirjutamise koodipäiste kiiruseid, millest kiiremaid kasutatakse testkoodides. Lõputöö eesmärkide täitmiseks testitakse erinevate keelte ja mikrokontrollerite kombinatsioonide puhul valgusdioodi vilkumise ning fototakistiga valguse tunnetamise täpsust ning kiirust süsteemi kontekstis. Samuti võrreldakse erinevate testide pikkustega tulemuste järjepidevust.

Lõpuks pakutakse testitulemuste põhjal välja Raspberry Pi 5 ning C-keele kombinatsioon.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 31 leheküljel, 7 peatükki, 19 joonist, 5 tabelit.

Abstract

Comparison of Microcontrollers and Programming Languages for Measuring Video Latency

The goal of this thesis is to find the best combination of a microcontroller and a programming language to be used in a video latency measurement system. The system in question was developed in a previous project which the author of this thesis was also a part of.

The microcontrollers being compared are the Raspberry Pi 5, Raspberry Pi 4B, ESP32-PoE-ISO and Arduino Mega, the last of which was omitted from final comparison due to its lack of crucial functionality in the scope of the project. The chosen programming languages were Python and C. Out of these chosen options, as the result of the thesis, a combination of a language and a microcontroller is given to be used in the continuation of the aforementioned project.

For comparing, a test system was built, which was like the one used in the previous project but with some limitations and reductions. For making the best choices for the code libraries used for reading from and writing to input-output ports, tests were run on different libraries. During testing, data was collected from every microcontroller and programming language combination to be able to compare the time differences between a light emitting diode blinks and a photoresistor senses it. For comparing the accuracy of these combinations, the exact timestamps at which both operations were executed, were recorded, and used for calculations later. The tests written were comprised of test cases which were different lengths. As a result, stability of different testing durations could be compared.

As a result of the testing, the author proposes to use the Raspberry Pi 5 and C combination.

The thesis is in Estonian and contains 31 pages of text, 7 chapters, 19 figures, 5 tables.

Lühendite ja mõistete sõnastik

ADC	*Analog-Digital Converter*
GPIO	*General Purpose Input-Output*, sisend-väljund viigud
I2C	*Inter-Integrated Circuit*
LED	*Light Emitting Diode*, valgusdiood
NTP	*Network Time Protocol*
PoE	*Power over Ethernet*
RTC	*Real-Time Clock*
SPI	*Serial Peripheral Interface*
UART	*Universal Asynchronous Receiver-Transmitter*

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Käesoleva töö teemaks on võrrelda erinevate mikrokontrollerite ja programmeerimiskeelte erinevuseid ja võimekuseid videopildi viite mõõtmise süsteemis. Täpsemalt käsitletakse viidet irdtornide videopildis ehk viivitust, mis tekib irdtornist üle kantava ning juhtimiskeskuses vastu võetava videopildi vahel.

Süsteem, mida hakatakse käsitlema, on eelnevalt aines „IAS1420 Arvutite ja süsteemide projekt“ valminud *proof-of-concept* [1]. Algses lahenduses on mikrokontrollerina kasutusel Raspberry Pi 4B ning programmeerimiskeelena Python. Kuna lennunduses on väga tähtis täpsus, siis on oluline ka, et tööriistad, millega süsteemi testitakse oleksid võimalikult töökindlad ning täpsed. Seetõttu on oluline uurida, millisest riist- ja tarkvarast peaks mõõtesüsteem koosnema, et tulemused oleksid võimalikult täpsed ning arvestatavad.

Antud töös võrreldakse omavahel nelja mikrokontrollerit ning kahte programmeerimiskeelt. Mikrokontrolleritena on valitud Raspberry Pi 4B, Raspberry Pi 5, Arduino Mega ja ESP32-PoE-ISO. Programmeerimiskeeltena on kasutusel eelnevalt valitud Python ning lisaks on veel juurde võetud C.

1.1 Taustainfo

Valdkondades, kus iga väiksemgi viga süsteemis võib tekitada ohtu inimeludele on tähtis selle täpsus ning töökindlus. Üheks selliseks valdkonnaks on lennundus. Vähe sellest, et lennukid peavad olema turvalised, peaksid seda olema ka infosüsteemid, millega turvalisust tagatakse.

Järjest enam kasutatakse lennuliikluse jälgimiseks irdtorne ehk kaugjuhitavaid lennujuhtimistorne [2]. Irdtornide kasutamise mõte on, et lennujuht ei peaks olema lennujaamas kohapeal, vaid et ta saaks teha oma tööd kaugelt, irdtornikeskusest. Selline lahendus aga tekitab olukorra, kus keskuse ja irdtorni vahele tekib täiendav viide, sest videopilt peab üle sidesüsteemi irdtorni jõudma ning see ei toimu silmapilkselt.

Hetkeseisuga ei ole võimalik tekkivat viidet täiesti ära kaotada. Seega on vaja teada, mis suurusjärgus tekkiv viide on.

1.2 Ülesanne

Praegune viis, kuidas eelmisel semestril teema püstitanud firma videopildi viidet mõõdab, on enamasti töötav, kuid siiski suhteliselt tülikas ja mitte eriti kasutajasõbralik. Nimelt pannakse tornis olevale kaamerale kaadrisse ekraan, millel näidatakse reaalajas kella. Juhtimiskeskuses tehakse pilt sissetulevast videopildist antud kaamerast ning lisaks veel samasugust reaalaja kella näitavast ekraanist. Saadud pildist saab välja lugeda kaks kellaaega, mis lahutatakse üksteisest ning nii saadakse videopildi viide.

Eelmisel semestril püstitatud probleemiks oli sellises süsteemis tekkiva videopildi viite mõõtmine. Kuna torn, kus on kaamerad ja juhtimiskeskus, kuhu videopilt jõuab on eraldi asukohtades, siis peab mõõtmiseks loodud süsteem koosnema kahest eraldiseisvast seadmest. Lisaks on kaamera andurist kuni juhtimiskeskuses oleva monitorini lubatud videopildi viide kuni 1 sekund, seega loodav süsteem ei tohi lisada juurde viidet, mis ületaks antud piiri.

Kuna tegemist on suhteliselt uue teemaga, siis pole teada olevalt varem välja mõeldud / kokku pandud sarnast süsteemi. Sellest tulenevalt pole ka varem omavahel võrreldud erinevaid mikrokontrollereid ja programmeerimiskeeli, mida oleks võimalik kasutada antud süsteemis.

Käesoleva bakalaureuse lõputöö eesmärgiks oli testida ning võrrelda omavahel valitud mikrokontrollereid ning samuti programmeerimiskeeli C ja Python. Võrdluse tulemusena leitakse sobivaim mikrokontroller ning keele kombinatsioon, mida on võimalik kasutada videopildi viite mõõtmise süsteemis. Uurimusküsimused, millele see lõputöö soovib vastata, on järgmised:

- Mikrokontrollerite valik ja võrdlemine.
- Programmeerimiskeelte valik ja võrdlemine.
- Mikrokontrollerite ja programmeerimiskeeltes loodud testkoodide mõõtmistäpsuste võrdlus.

Et lõputöö tegemine liigselt ajakulukaks ei muutuks on valitud võrreldavateks mikrokontrolleriteks ja keelteks sellised väljavalitud, millega autoril on juba varasem kokkupuude olemas.

2 Võrreldavad mikrokontrollerid

Mikrokontrollerite valiku puhul on võetud arvesse autori eelnevat kogemust. Seda asjaolul, et tööd oleks võimalik alustada võimalikult vara. Kuna *proof-of-concept*is oli kasutusel Raspberry Pi 4B, siis üheks testitavaks jäi see. Testiti veel ka eelneva uuemat mudelit Raspberry Pi 5-te. Lisaks valiti veel Arduino Mega ning ESP32-PoE-ISO.

Samuti peeti valiku tegemisel eelkõige silmas, et sellel peab kindlasti olemas olema etherneti võimekus, sest viite mõõtmiseks on vaja süsteemis olevate seadmete kellaaegade sünkroniseerimist interneti kaudu. Veelgi parem kui nad toetavad *Power over Ethernet* (PoE) kasutamist. PoE lubab seadet toita vooluga üle etherneti kaabli ning seeläbi kaotaks ära ühe lisa kaabli.

Kuna programmeerimiskeelteks on valitud C ja Python, siis pidi arvestama, et mikrokontrolleritel peaks olema võimalik mõlemaid kasutada. Samuti pidi arvestama, et kontrollerid peaksid olema piisavalt hea võimekusega, et antud süsteemi oleks võimalik nendel jooksutada.

Arendatavas süsteemis oleks hea kui oleks võimalik NTP (*Network Time Protocol*) serverilt saada aega vähemalt mikrosekundi täpsusega. Seetõttu on üheks nõudeks võimekus koodis seda NTP serverilt piisava täpsusega ka küsida.

Järgnevalt on ära toodud valitud mikrokontrollerite spetsifikatsioonid ning kui mõnega neist tekkisid probleemid, siis ka kuidas need lahendati.

2.1 Raspberry Pi

Raspberry Pi on oma olemuselt nagu tavaline arvuti. Sellele on võimalik peale installeerida Linux'i operatsioonisüsteem ning sel on üleüldiselt väga lai kasutusala. Näiteks kõige tavalisemad kasutusjuhud on erinevate serverite jooksutamine, meediakeskusena kasutamine või viimasel ajal ka tehisintellekti jaoks [3], [4].

Olenevalt sellest, milleks seadet kasutatakse ja kui palju on jõudlust või laiendusvõimalusi vaja, on Raspberryl valikus erinevaid kontrollereid. Näiteks kui pole

vaja väga palju võimekust, laiendusvõimalusi ega ligipääsetavust, siis on võimalik kasutada Raspberry Pi Picot [5] või Zerot [6]. Kui aga on soovi kasutada Pi-d personaalarvutina või jooksutada serverid jms, siis on valikus peamine tootesari, millest lõputöö kirjutamise ajal kõige uuem on Pi 5 [7], [8].

Erinevalt meie igapäevases kasutuses olevatest arvutitest on Raspberrydel olemas *General Purpose Input-Output* (GPIO) viigud, millega on võimalik teha kiirprototüüpimist ning kokku panna lihtsamaid elektriskeeme.

Mõlema kasutusel olnud Pi puhul laaditi peale 64-bitine *Raspian* OS.

2.1.1 Mikrokontroller Raspberry Pi 4B

Raspberry Pi 4B (Joonis 1) oli kasutusel juba antud lõputööle eelnevas projektis ning seetõttu jäi see ka üheks valituks. Peamiseks põhjuseks oli võrdlusmomendi tekkimine. Testid, mida lõputöö käigus andmete saamiseks kasutati olid edasiarendused koodist, mida kasutati projekti käigus, ning millest saadud tulemuste võrdlemisel peaks tekkima hea ülevaade sellest, milline on valitutest parim.

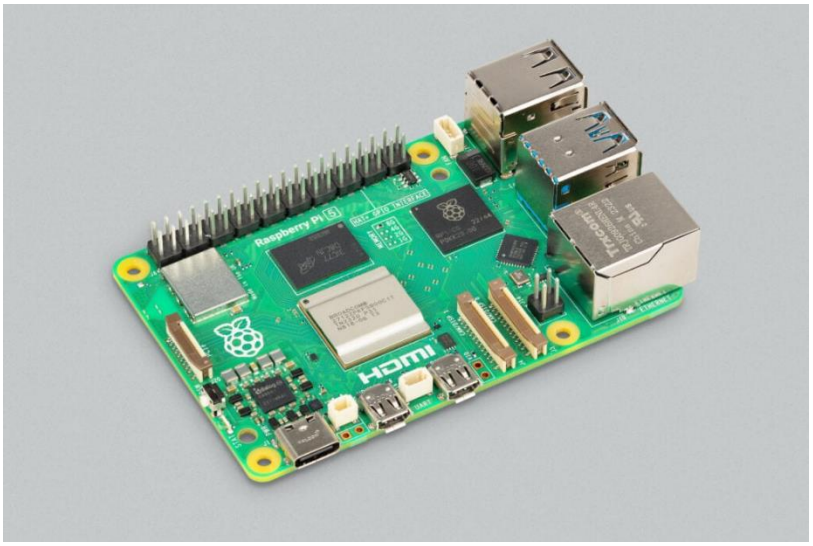
Joonis 1. Raspberry Pi 4B [9].

Kasutatud Raspberry Pi 4B on 4-gigabaidise muutmäluga (*Random Access Memory*) ning sellel on Broadcom BCM2711 kiip, millel on nelja tuumaline ja 64-bitine Cortex-A72 protsessor. Ethernet, juhtmevaba internet ja ka vajadusel Bluetooth on sellel olemas. Samuti on võimalik kasutada PoE-d, kuid seda ainult selleks ette nähtud laiendusplaadiga.

Prototüüpimiseks on olemas 40 GPIO viiku, millest osasid saab antud töö juures kasutada väliste sensorite ja muude seadmete ühendamiseks, näiteks LED-i (Lig*ht Emitting Diode*) vilgutamiseks ja ADC (*Analog-Digital Converter*) ning fototakistiga valgustasemete mõõtmiseks.

2.1.2 Mikrokontroller Raspberry Pi 5

Raspberry Pi 5 (Joonis 2) oli lõputöö kirjutamise hetkel kõige uuem Pi versioon. Selle valimise põhjuseks oli võrrelda omavahel kahte erinevat Raspberry Pi-d ning leida, kas uuem mudel toob kaasa ka märgatavalt paremad mõõtmistulemused või sobib jääda vanema juurde.



Joonis 2. Raspberry Pi 5 [10].

Nagu uuematele mudelitele kombeks, on Pi 5 puhul kasutusel uuemad ja paremad tehnoloogiad. Antud töös oli kasutusel seadme 8-gigabaidi muutmäluga mudel. Selles on kasutusel uus Broadcom BCM2712 kiip, milles on 64-bitine, nelja tuumaline Arm Cortex-A76 protsessor. Nagu oli ka Pi 4-l, on Pi 5-l olemas PoE võimekus, kuid seda kasutades lisaplaati.

Pi 5-l on samuti prototüüpimiseks olemas 40 GPIO viiku. Siinkohal aga tuleb meeles pidada, et uue mudeliga tehti ümber nende tarkvaraline juhtimine, seega osad GPIO juhtimiseks mõeldud koodipäised mis töötasid Raspberry Pi 4B puhul ei pruugi seda enam teha Pi 5 puhul. Näiteks *proof-of-concept*is kasutatud RPi.GPIO koodipäis hetkeseisuga Pi 5-e peal ei tööta [11], [12]. See asjaolu võib aga muutuda aja edasi minnes, sest selleks tuleks antud teek uue Pi jaoks ümber kirjutada, millega inimesed kindlasti ka tegelevad. Seda eriti asjaolul, et teegid on avatud koodiga ning isegi kui originaalautor ei tegele enam selle arendusega, saavad seda teha teised asjatundjad, kellel on selleks huvi.

2.2 Mikrokontroller ESP32-PoE-ISO

Kolmandaks võrreldavaks valiti ESP32-PoE-ISO (Joonis 3), sest see on ehituselt rohkem mikrokontrolleri lähedane. Sellel puudub operatsioonisüsteem ning selleks, et seda kasutada tuleb kirjutatud programm laadida mikrokontrolleri mällu ning seeläbi saab neid jooksutada. Kuna ESP32 on üks populaarsemaid arendusplaate, siis on selle kohta saadaval palju materjale.

Neil on olemas suurel hulgal erinevaid lisamooduleid, mida saab kasutada *Serial Peripheral Interface* (SPI), *Inter-Integrated Circuit* (I2C) või *Universal Asynchronous Receiver-Transmitter* (UART) liidestega. Paljudel on olemas ka sisseehitatud ADC, mis teeb lihtsaks erinevate analoogandurite kasutamise. ESP32 on hea valik kui vaja mikrokontrollerit, mis on võimekas, kuid on samas väike ja vähese energiatarbega.

Tavalisteks ESP32 kasutusjuhtudeks ongi väiksemad süsteemid, mis on ehitatud mingiks kindlaks eesmärgiks. Kuna ESP32-del on enamasti olemas internetile ligipääsu võimekus, siis kasutatakse neid väga palju asjade interneti süsteemides. Mõned näited nende kasutuse kohta oleks näiteks robootika, kaugelt ligipääsu süsteemid, keskkonna monitoorimine ja ka igasugused kantavad seadmed. [13]

Kindlasti on tegemist spetsifikatsioonide poolest vähem võimekama seadmega kui seda on Raspberry Pi-d. Konkreetsel ESP32-PoE-ISO-l on muutmälu ainult 520 kilobaiti. Lisaks on sellel 4 megabaiti välkmälu [14]. Kui projektis on aga vaja rohkem mäluruumi, siis on ESP32-l olemas SD-kaardi kasutamise võimekus.

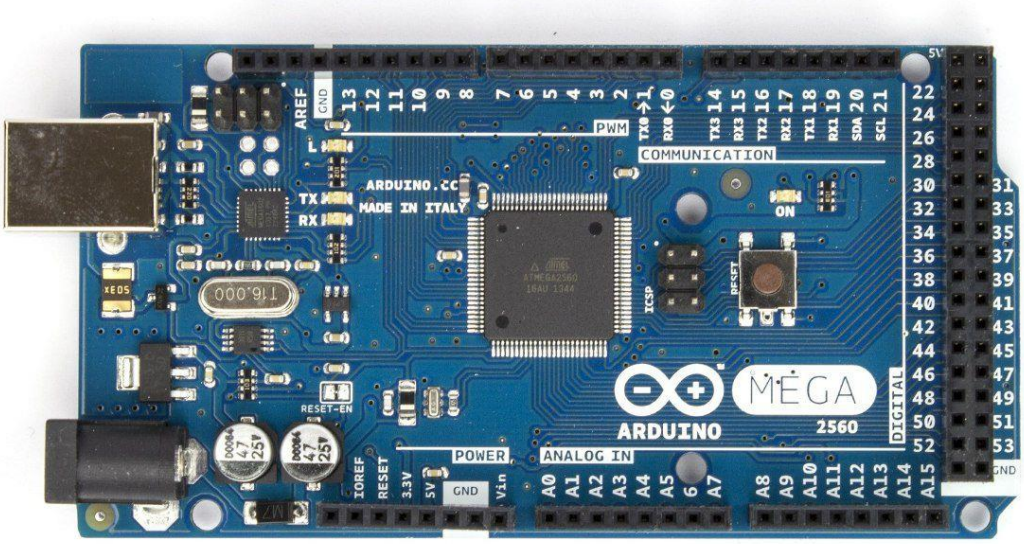
Joonis 3. ESP32-PoE-ISO [15]

Prototüüpimiseks on olemas 20 GPIO viiku, neist 6 on vabalt kasutatavad [16]. Samuti on antud arendusplaadil olemas ka 12-bitine ADC, mida on võimalik testimisel analoogsensoritega kasutada ilma, et oleks vaja eraldiseisvat kiipi.

Üks kõige määravamaid omadusi, mis valitud mikrokontrolleril on ning mille pärast see ka valiti, on selle PoE võimekus. See on üks väheseid ESP32 mikrokontrollereid, millel on olemas *on-board* PoE võimekus.

2.3 Mikrokontroller Arduino Mega

Arduino Mega (Joonis 4) on valitud mikrokontrolleritest kõige tagasihoidlikumate spetsifikatsioonidega. Arduino arendusplatvorm ongi tavaliselt mõeldud pigem algajale riistavara programmeerijale ning pole mõeldud väga suurteks projektideks. Samas on olemas suurel hulgal sellega ühilduvaid väliseid andureid ja lisaseadmeid. Samuti on mitmel Arduino arendusplaadil olemas Ethernetiga ühendamise tugi, kasutades selleks ettenähtud lisaplaati [17]. Kahjuks aga pole antud mikrokontrolleril ja Etherneti lisamoodulil PoE võimekust, mis on selle üheks miinuseks.



Joonis 4. Arduino Mega [18]

Mega on varustatud ATmega 2560 protsessoriga ning sellel on ainult 8 kilobaiti muutmälu. Lisaks on olemas ka 256 kilobaiti välkmälu. Kui kasutada Etherneti laiendusplaati, siis on võimalik info salvestamiseks kasutada ka SD-kaarti.

Arduino Megale koodi kirjutamist alustati C-keelest. Kõigepealt kirjutati kood internetiühenduse loomiseks ning kui see saadi tööle, kirjutati Pi-del kasutatud koodidele sarnase algoritmiga testimiskood.

Nagu oli tehtud eelnevalt Pi-del, püüti NTP serverilt saadud kellaaja täpsust muuta alles peale koodi valmis kirjutamist. Hakati uurima erinevate viiside kohta selle saavutamiseks, kuid üpris kiiresti tekkis probleem. Nimelt pole Arduino Megale sobivat koodipäist, millega oleks võimalik küsida aega isegi vähemalt millisekundi täpsusega. On olemas teek *precise_sntp*, mis seda võimaldaks, kuid see töötab ainult teatud arhitektuuriga Arduino mikrokontrolleril, mida Mega pole.

Lõpuks, peale mitmeid tunde otsimist ning internetimaterjalide lugemist sai selgeks, et antud mikrokontrolleri arhitektuuri jaoks lihtsalt puudub võimalus saada serverilt aega suurema täpsusega kui seda on sekund. Küll aga tuli välja, et täpsema kellaaja saamiseks tuleks kasutada kas ESP32 mooduliga Arduinot või muretseda RTC (*Real-Time Clock*)

ehk reaalajakell [19]. Otsustati lõpuks, et kuna ESP32 on niikuinii üks mikrokontroller, mida edaspidi ka testitakse, siis pole vajalik Arduino Megaga edasi minna.

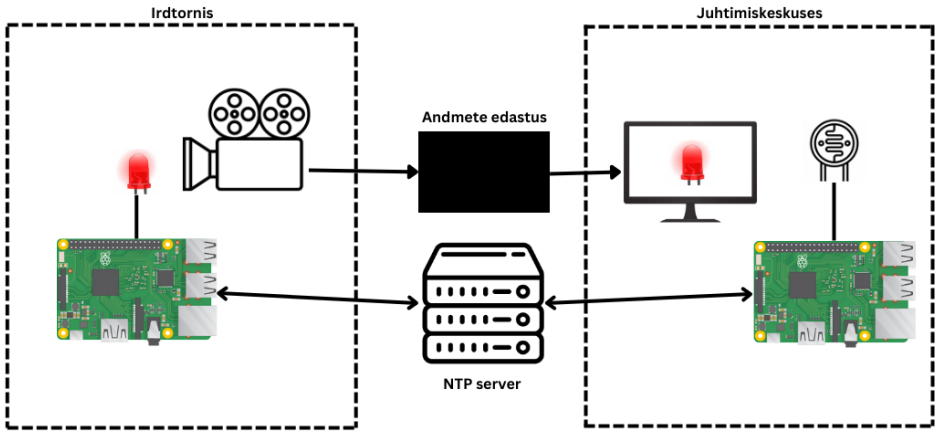
3 Võrreldavad programmeerimiskeeled

Eelnevalt valminud *proof-of-conceptis* oli programmeerimiskeelena kasutusel Python ning see jäi ka üheks võrreldavaks keeleks edasi. Kuna tegemist on interpreteeritava keelega, siis ei ole see väga mikrokontrolleri sõbralik keel. Siiski on võimalik näiteks ESP32-te Pythoniga kergelt programmeerida. Arduino puhul on asi üpris palju keerulisem, sest see on mõeldud C/C++ programmeerimiseks. Siiski on internetis saadaval mitteametlikud Arduinole sobivaks tehtud meetodid, millega teoorias peaks olema võimalik osasid Arduino arendusplaate Pythoniga programmeerida [20]. Kuna Arduinol on väga väikese võimekusega arendusplaadid, siis loomulikult ei ole võimalik sellisel viisil kasutada kogu Pythoni keele võimekust. Selline võimalus on olemas ka antud töös valitud Arduino Megal kasutades Pythoni teeki *pyfirmata* [21].

Teiseks programmeerimiskeeleks oli C. Kuna C on mõeldud just riistvaralähedaseks programmeerimiseks ning sellega on võimalik programmeerida kõiki nelja mikrokontrollerit väga lihtsasti, siis oli see üks valikutest. Lisaks oli autoril ulatuslik varasem kogemus C-keelega olemas.

4 Loodud lahendus ja testimise metoodika

Eelnevalt mainitud projektiaine raames valmis süsteem, mis koosneb kahest Raspberry Pi 4B-st. Ühe Pi küljes on elektriskeem, mis vilgutab LED-i iga kahe sekundi tagant ning teise küljes skeem, milles valgussensor tunnetab valgusimpulsse (Joonis 5). Kuna Pi-del on erinevad elektriskeemid, siis sellest tulenevalt on neil peal ka erinevad koodid, mis täidavad oma ülesandeid. Teavet andmete edastamise viisi kohta irdtornist juhtimiskeskusesse pole antud töö jaoks vaja. Joonisel 5 ehk süsteemi skeemil on see seetõttu ära märgitud musta kastina.



Joonis 5. Süsteemi skeem.

Kuna Raspberry Pi enda riistvaraline kell võib päevas umbes 0,26 sekundit triivida [22], siis süsteemis, kus aja määramine on põhifookuses, on tähtis, et kellad oleksid võimalikult täpsed. Seetõttu on mõlemal seadmel koodi sisse kirjutatud samasugune aja sünkroniseerimine NTP serveriga.

Lähemalt saab valminud *proof-of-conceptist* lugeda projekti aruandest [1].

4.1 Testitav süsteem

Erinevalt algsest süsteemist on testimiseks kasutatud korraga ühte seadet. Kui eelnevalt oli kaks Pi-d ning ühe küljes oli valgussensor ja teise küljes LED, siis testitavas lahenduses on kasutatud ainult ühte Raspberry Pi-d. Selle külge on ühendatud nii LED kui ka sensor ning mõlemaid juhitakse paralleelselt.

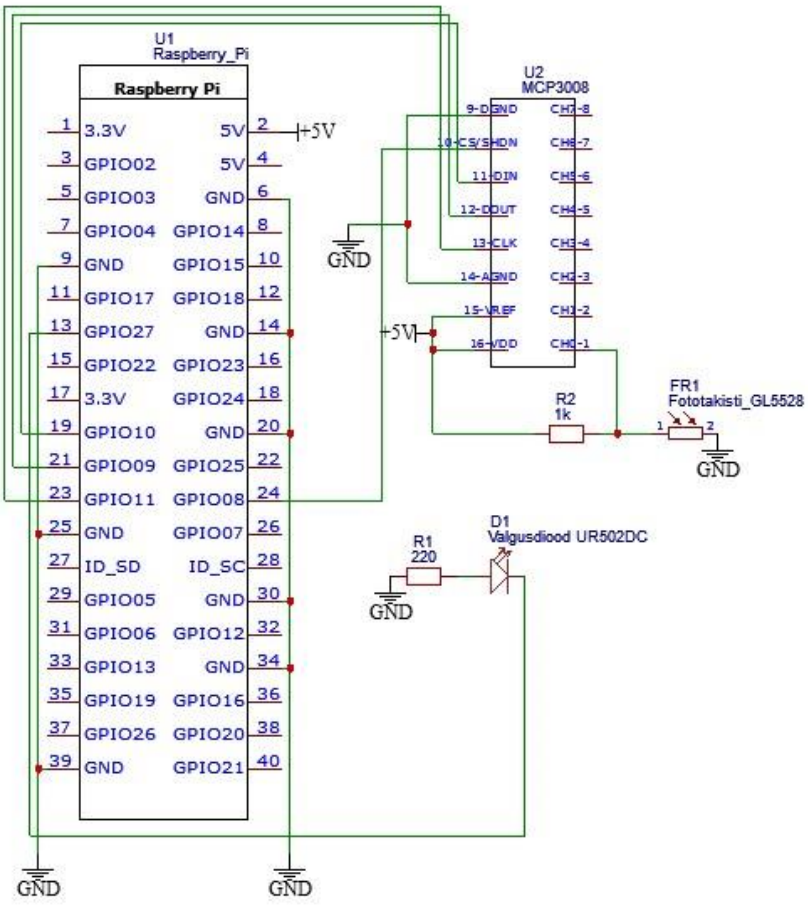
Erinevuseks on skeemis ka veel see, et sensorilt saadavate andmete lugemiseks on kasutatud elektriskeemis analoog-digitaalmuundurit (ADC), millega saab paremini jälgida, kas sensor tuvastas valgust või mitte.

Kuna töö eesmärgiks on võrrelda erinevaid mikrokontrollereid ning programmeerimiskeeli, siis et testida kohe nii LEDi kui ka valgussensori juhtimist koos samaaegselt, on pandud kõik üheks kokku. See tähendab, et kogu vajalik elektriskeem, nii LED kui ka sensor, on ühes ning neid juhitakse ühe koodiga samaaegselt.

Antud lähenemine teeb ka lihtsamaks hiljem välja töötatava lõpplahenduse, millega peab tagama, et ühel seadmel peab olema võimekus vilgutada nii valgusdioodi kui ka tunnetada sensoriga saadavat impulssi. Viimane tuleneb ettevõtte soovist, et seadmed peaksid olema vahetatavad ning ei tohiks lugeda kumb võetakse kaasa irdtorni ning kumb jääb keskusesse.

4.2 Riistvara konfiguratsioon

Testimisel kasutati lihtsat elektriskeemi, mis koosnes valgusdioodist, fototakistist GL5528 [23] ning Raspberry Pi-de puhul ka eraldiseisvast ADC-st MCP3008 [24]. ESP32-puhul oli kasutusel sellele sisseehitatud analoog-digitaalmuundur. Kasutatud elektriskeem on näha joonisel 6.



Joonis 6. Testimisel kasutatud elektriskeem.

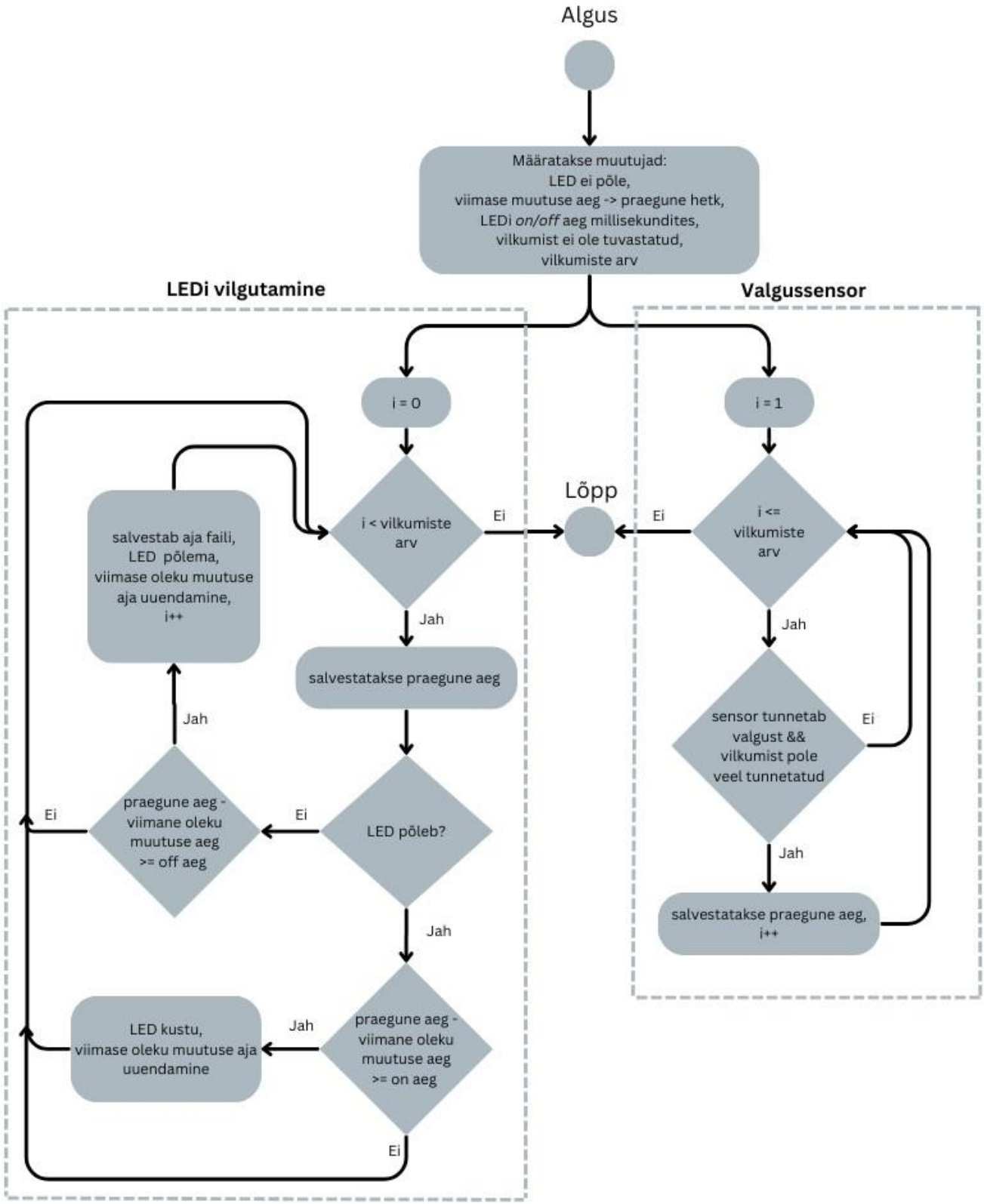
4.3 Kasutatav tarkvara ja konfiguratsioon

Kirjutatud koodid on ülesehituselt võimalikult sarnased. Raspberry Pi-de puhul on nii C kui ka Pythoni koodi kirjutamise aluseks kasutatud ühte algoritmi (Joonis 7). ESP32 puhul on kasutatud samuti sarnast algoritmi (Joonis 8), kuid ilma erinevate lõimedeta (*thread*).

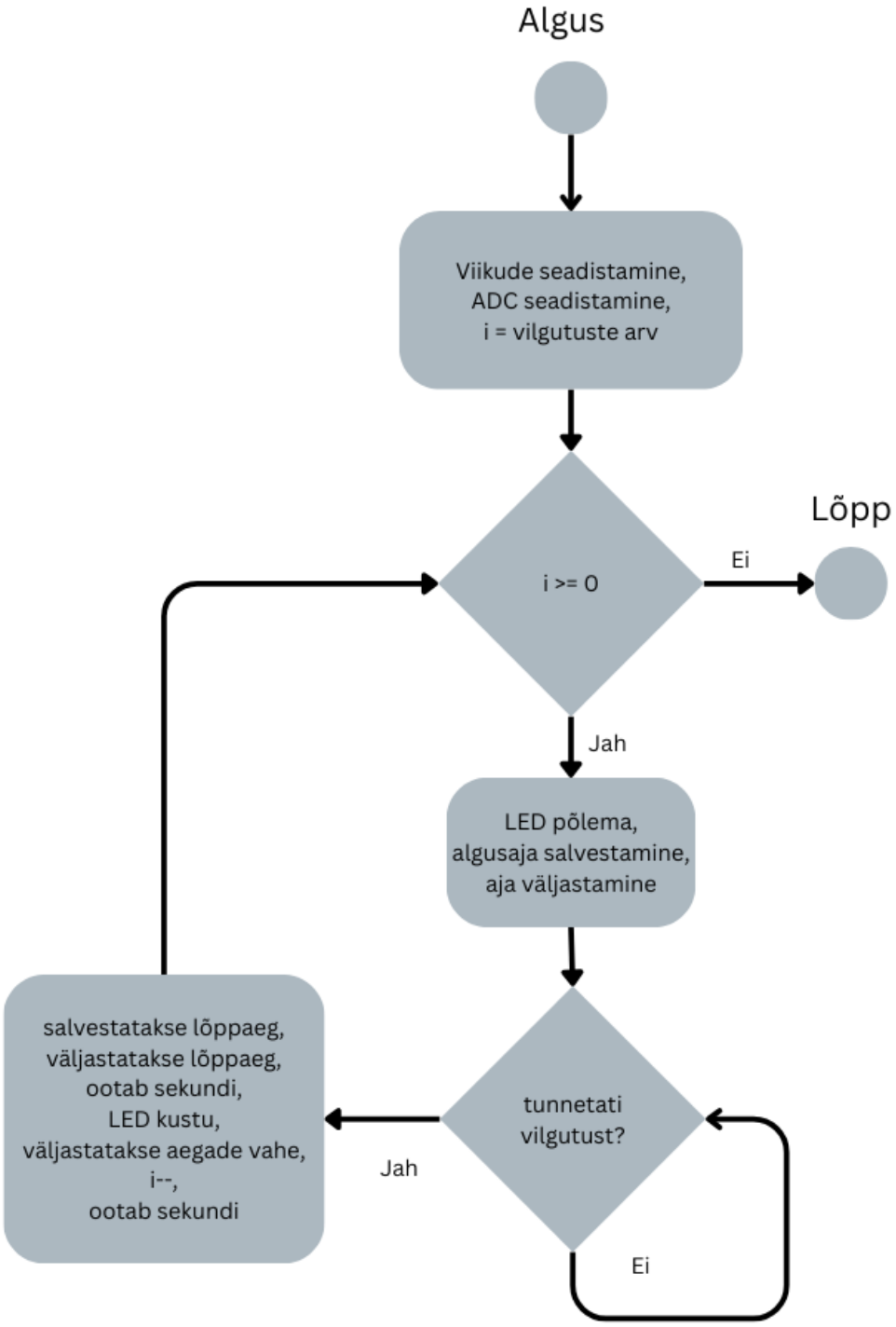
Kuna tegemist on eri keeltega, siis on koodide kirjutamiseks kasutatud erinevaid teeke. Viikude lugemiseks on mõlema Pi ning keele puhul mitmeid erinevaid teeke võimalik kasutada. Ka neid püüti valida nii, et mõlema Pi-ga oleks võimalik sama keele puhul ka samasid teeke kasutada. Valiku tegemiseks tehti koodipäiste kirjutamise ja lugemise kiiruste testid ning nende tulemuste põhjal valiti sobivad teegid.

Katsetest ning nende tulemustest on võimalik lähemalt lugeda peatükis 5.1. Lõpuks valiti nii C-keele kui ka Pythoni jaoks teek *libgpiod* ehk *gpiod*, sest see andis mõlema keele puhul kõige paremad tulemused. Link kirjutatud koodidele on võimalik leida Lisast 2.

ESP32-PoE-ISO programmeerimiseks kasutati C-keele jaoks Arduino programmeerimiskeskkonda [25] ning Pythoniga programmeerimiseks Thonny keskkonda [26].



Joonis 7. Raspberry Pi-de koodide algoritm.



Joonis 8. ESP32 testkoodide algoritm.

4.4 Ülevaade töö käigust

Tööd alustati uurimisega, milliseid mikrokontrollereid ja keeli võiks antud töö käigus kasutada. Algselt uuriti ka STM32 kohta, sest üks selline mikrokontroller oleks juhendajal veel olemas olnud. STM32 puhul oleks olnud internetiga ühenduse loomiseks

kasutada ka Arduinol kasutatavat selleks ette nähtud lisaplaati. Kahjuks aga ei saadud neid kahte omavahel tööle ning STM32 langes ära.

Kõige esimesena alustati tööd Raspberry Pi 5-ga, sest see saadi kätte Pi 4B-ga samal ajal. Üpris palju läks aega teekide leidmisele, eriti ADC lugemiseks. Tööd raskendas asjaolu, et Pi 5 oli töö tegemise hetkel alles hiljuti välja tulnud mudel ning lisaks oli sellel muudetud viikude ülesehitust, mis tähendas, et eelnevalt kirjutatud teegid ei pruukinud enam selle peal töötada. Õnneks aga olid juba olemas mõned, mida sai kasutada.

Peale Pi 5-ga tegelemist testiti ära ka Pi 4B. Selle puhul sai kasutada Pi 5-le juba kirjutatud koode, kuid enne tuli neid modifitseerida.

Järgmisena võeti käsile Arduino Mega. Sellele loodi ühendus internetiga, küsiti aega NTP serverilt ning kirjutati kood sarnase ülesehitusega nagu ka Pi-de puhul. Kahjuks aga tekkis probleem serverilt vähemalt millisekundilise täpsusega aja küsimisega. Lõpuks ei leitud viisi kuidas seda probleemi lahendada ning otsustati Arduino Mega-ga testimine välja jätta.

Arduinoga töö lõpetatud, tehti ära testid ESP32-ga. Jällegi kirjutati valmis testkood. Püüti teha ka SD-kaardile andmete kirjutamise võimalus, kuid kuna kasutatud kaart oli vigane ning sellega tegelemiseks enam väga palju aega polnud, siis otsustati see tegemata jätta.

Peale esimeste testide tegemist tehti ära ka veel GPIO teekide kiiruste testid. Selle tulemusena tuli välja, et tol hetkel kasutusel olnud teegid polnud kõige kiiremad, peale mida tuli Pi-de koodid ümber teha ning testid uuesti teha. Siinkohal mõõdeti ka üle kasutatud fototakisti omaviide.

Viimasena võeti kõik testide tulemused kokku ning analüüsiti neid, mille kohta on võimalik lugeda järgmises peatükis.

5 Eksperimentide tulemused

Antud töö jooksul testiti erinevate programmeerimiskeelte ja mikrokontrollerite kombinatsioonide täpsust ja kiirust. Samuti, et teha valik, milliseid GPIO juhtimise teeke Raspberry Pi-de puhul koodide kirjutamise jaoks kasutada, testiti nii C kui ka Pythoni puhul Raspberry Pi 5 peal teekide viikudele kirjutamise ja neilt lugemise kiiruseid (vaata peatükk 5.1).

Saadud tulemustest parema ülevaate saamiseks fototakisti omaviite maha arvestamiseks mõõdeti ka see üle.

5.1 Raspberry Pi GPIO teekide kiirused

Teekide valik tehti nende nii Pi 4B kui ka Pi 5 peal olemasolu põhjal. Nagu mainitud peatükis 2.1.2, polnud osasid teeke veel Pi 5 peale sobivalt ümber kirjutatud. Seetõttu pidi nende valimisel kindlaks tegema, et neid oleks võimalik kasutada mõlema mudeli peal.

Lõplikult jäid valituks Pythoni teegid *gpiod* ja *gpiozero* ning C teegid *gpiod* ja *WiringPi*. Järgnevalt on välja toodud antud teekide testide tulemused.

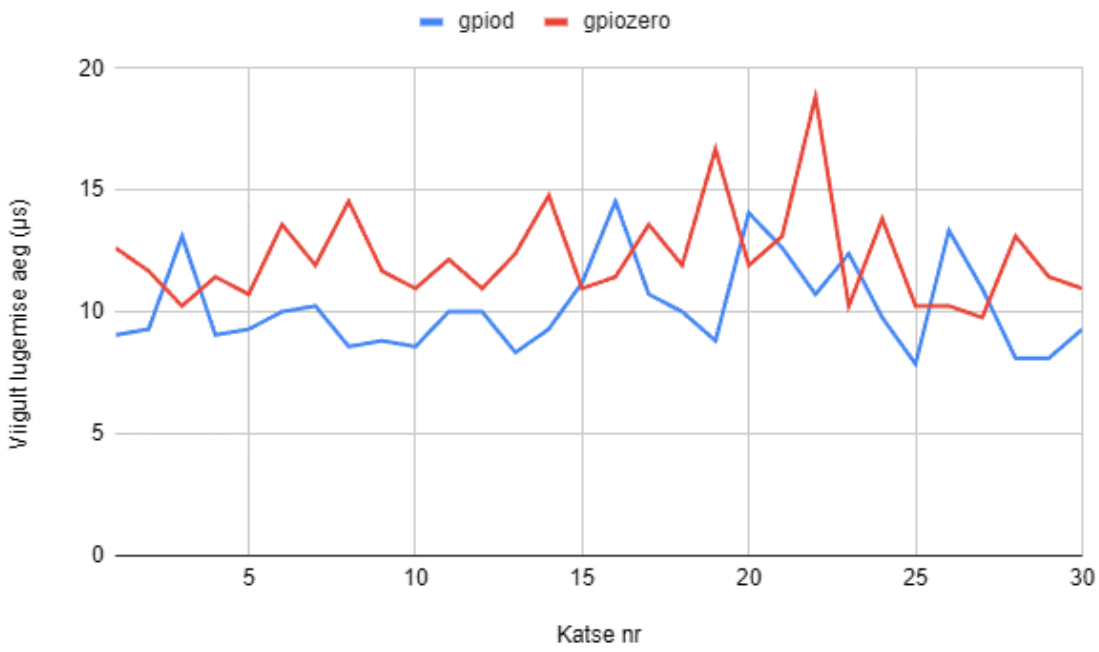
5.1.1 Pythoni koodipäised

Saadud tulemuste põhjal on viigult lugemise kiirused *gpiod* ja *gpiozero* puhul väga erinevad. Teegi *gpiozero* keskmine lugemiskiirus on umbes 25,9 mikrosekundit samas kui *gpiod* puhul on see lausa pea kolmekordselt väiksem: 8,8 mikrosekundit. Samuti on standardhälbed vastavalt 1,5 ja 2,8 µs. Joonisel 9 on graafiliselt ära toodud mõõtmiste tulemused. On näha, et *gpiod* lugemiskiirus on konstantselt madalam ning stabiilsem kui seda on *gpiozero* oma.

Joonis 9. Pythoni teekide viigu lugemiskiirused.

Kirjutamiskiiruste puhul on tulemused juba võrreldavamad. Teegi *gpiod* keskmine kirjutamiskiirus on 10,2 µs ning *gpiozero* puhul on see 12,3 µs. Standardhälbed on vastavalt 1,8 ja 2,0 µs.

Joonisel 10 on näha, et kiirused on üpris sarnased, kuid *gpiozero* puhul on need kohati kõrgemad ning ebastabiilsemad.



Joonis 10. Pythoni teekide viigule kirjutamise kiirused.

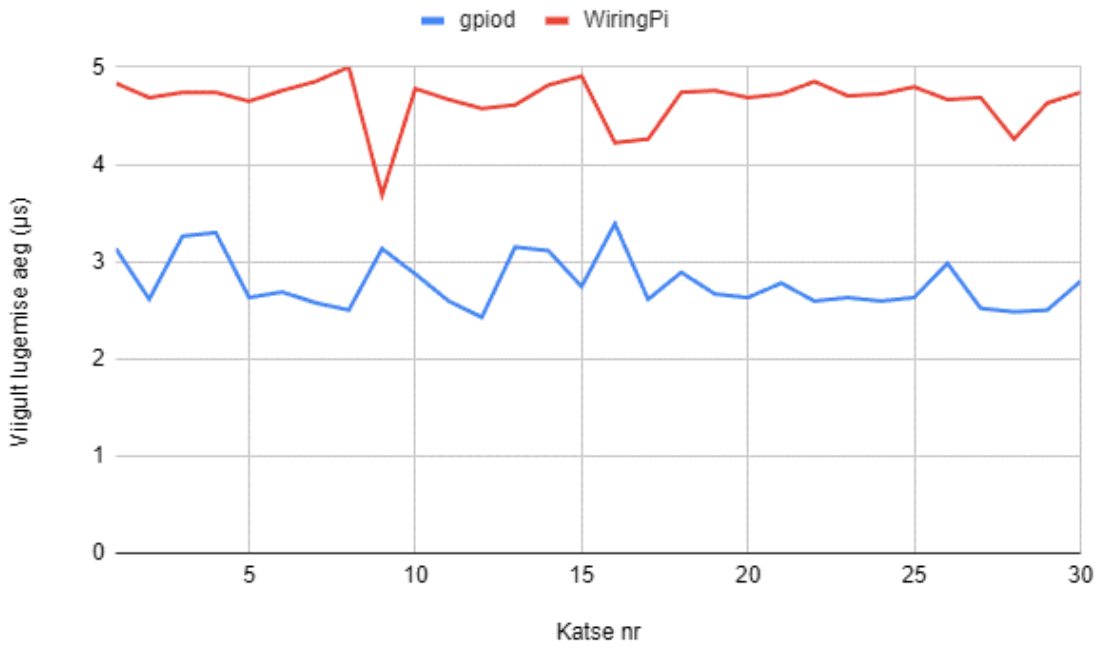
Teekide võrdlused on toodud ka tabelis 1.

Tabel 1. Pythoni teekide võrdlus.

	Keskmine kirjutamiskiirus (µs)	**Keskmine lugemiskiirus (µs)**	**Kirjutamiskiiruse standardhälve (µs)**	**Lugemiskiiruse standardhälve (µs)**
gpiod	10,2	25,9	1,8	1,5
gpiozero	12,3	8,8	2,0	2,8

5.1.2 C koodipäised

C programmeerimiskeele teekide *gpiod* ja *WiringPi* lugemiskiirused olid mõlemad madalamad kui Pythoni puhul oli *gpiod*. Kui teha keele valik juba selle põhjal, võiks öelda, et C keel on kindlasti parem valik. Teegi *gpiod* keskmine lugemiskiirus oli 2,8 µs ja *WiringPi* puhul oli see 4,7 µs. Standardhälbed on mõlemal 0,3 µs. Antud tulemuseid illustreerib joonis 11.

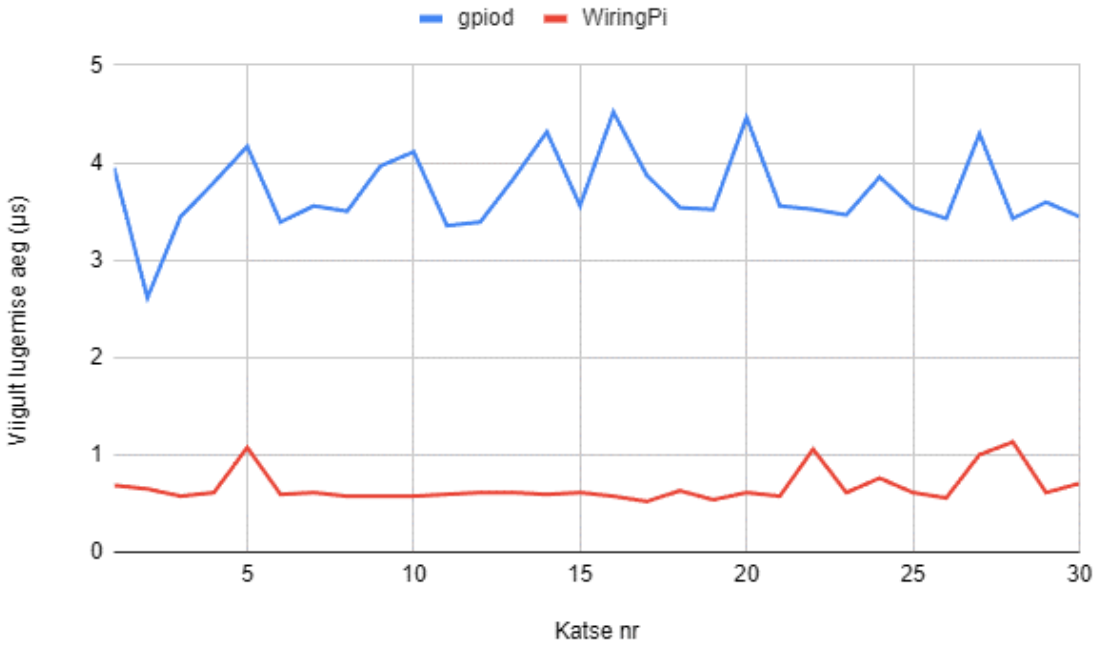


Joonis 11. C teekide viigu lugemiskiirused.

Gpiod teek on konstantselt kiirem. Selle puhul on küll rohkem hüppeid, kuid need on väiksemad kui seda on *WiringPi* puhul.

Kirjutamiskiiruse puhul oli aga vastupidiselt kiirem hoopis *WiringPi* ning seda lausa kolmekordselt. Kui *gpiod* puhul oli keskmine kiirus 3,7 µs, siis *WiringPi* puhul oli see lausa 0,7 µs.

Joonisel 12 on näha, et *WiringPi* koodipäise kirjutamiskiirus on kohati väga konstantne. Arvutatud standardhälve oli *WiringPi* puhul koguni 0,2 µs ning *gpiod* puhul oli see 0,4 µs.



Joonis 12. C teekide viigule kirjutamise kiirused.

Programmeerimiskeele C koodipäiste võrdlused on näidatud ka tabelis 2.

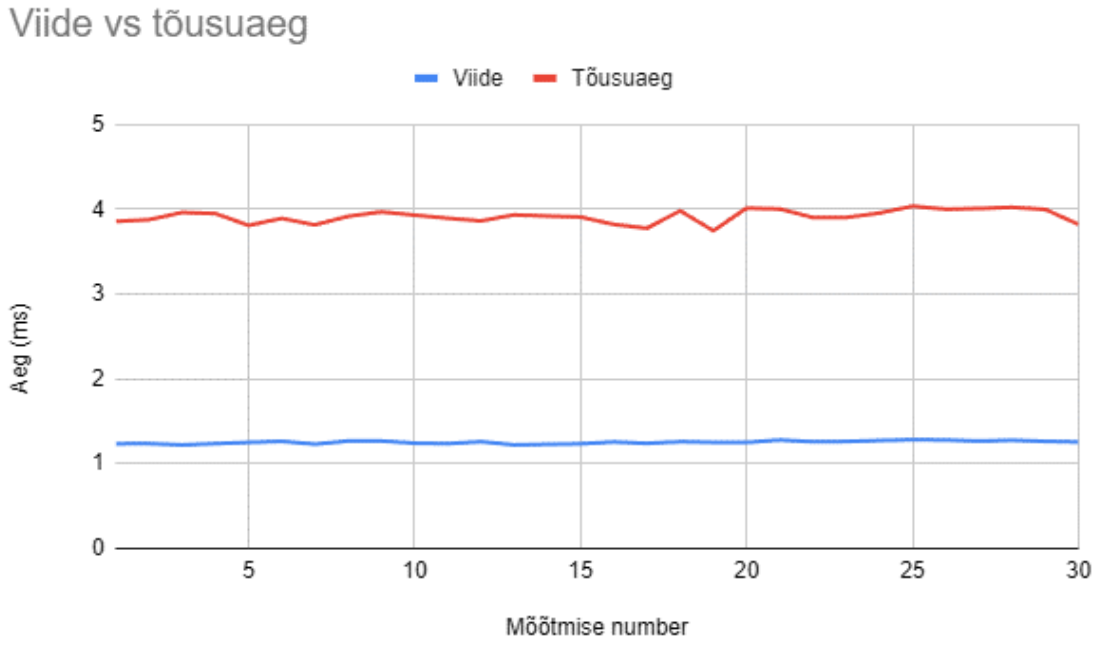
Tabel 2. C teekide võrdlus.

	Keskmine kirjutamiskiirus (µs)	**Keskmine lugemiskiirus (µs)**	**Kirjutamiskiiruse standardhälve (µs)**	**Lugemiskiiruse standardhälve (µs)**
gpiod	3,7	2,8	0,4	0,3
WiringPi	0,7	4,7	0,2	0,3

5.2 Fototakisti omaviide

Selleks, et oleks parem ülevaade, milline on tegelik koodidest ja mikrokontrolleritest tekkinud viide, on lõputöö jooksul üle mõõdetud ka kasutatud fototakisti omaviide. Omaviite mõõtmiseks kasutati üldtestideks kasutatud elektriskeemi (Joonis 6) ning mõõtmised tehti ostsilloskoobiga Keysight DSOX112G. Kokku tehti 30 mõõtmist, et saaks hea ülevaate, milline on keskmine fototakisti omaviide.

Tehtud mõõtmiste tulemusena saadi keskmiseks viiteks 1,3 ms ning keskmiseks tõusuajaks 3,9 ms, mida illustreerib ka joonis 13.



Joonis 13. Fototakisti omaviite mõõtmise tulemused.

5.3 Programmeerimiskeelte ja mikrokontrollerite võrdlused

Testide eesmärgiks oli leida erinevate koodide ja mikrokontrolleritega nende kiirused ehk kui kaua läheb aega, et LED saaks vilkuda ning sensor sellele reageerib. Samuti arvutatakse välja LEDi vilkumiste aegade vahed ning standardhälbe abil saadakse teada kui täpselt seda tehakse ning milline mikrokontroller ja programmeerimiskeel on kõige stabiilsem. Sama tehakse ka sensori tunnetamise aegadega, ehk arvutatakse vahed ning seeläbi leitakse valguse tunnetamise stabiilsus ja täpsus.

Tehtud testid olid üles ehitatud nii, et oli viis erinevat vilkumiste arvu, millega need tehti. Valitud arvud olid 10, 50, 100, 250 ja 500. Raspberry Pi-de puhul tehti igat testi viis korda, ESP puhul tehti neid vähem. Osadel graafikutel on näha alumisel teljel nimesid nagu „Test10“, „Test 54“ või „Test 503“. Nende järgi saab ära öelda, mitu vilgutust antud testi jooksul tehti ning mitmes see oli. Näiteks „Test10“ tähendab, et testi jooksul tehti 10 vilgutust ning see oli esimene test, „Test503“ puhul oli 500 vilgutust ja tegemist oli neljanda testiga. Ehk üheliste kohal olev number, mida hakatakse lugema arvust 0, näitab ära mitmenda testiga on tegemist. Selline jaotus tehti, et näha kas erinevate vilgutuste

arvuga tekib ka näiteks mingi muster või mitte. Samuti saab selle puhul öelda, kui mitu vilgutust peaks testi jooksu tegema, et saaks vajalikud andmed kätte.

5.3.1 Kiiruste võrdlus

Kiiruste arvutamiseks salvestati testi ajal aeg, millal valgusdiood põlema lülitati, faili. Samuti tehti ajaga, millal koodis fototakistilt saadud väärtus läks üle teatud piiri. Nimetatud piiriks võeti testide puhul 70 % kogu fototakisti väärtuste ulatusest. Analüüsi käigus lahutati vastavad ajad üksteisest ehk valguse tunnetamise ajast lahutati põlema minemise aeg. Järgnevalt on tabelis 3 toodud testide käigus saadud keskmised ajad.

Tabel 3. Kiiruste keskmised väärtused.

	Raspberry Pi 5	**Raspberry Pi 4B**	**ESP32-PoE-ISO**
Python	20,5 ms	20,7 ms	1,9 ms
C	4,6 ms	3,9 ms	2,5 ms

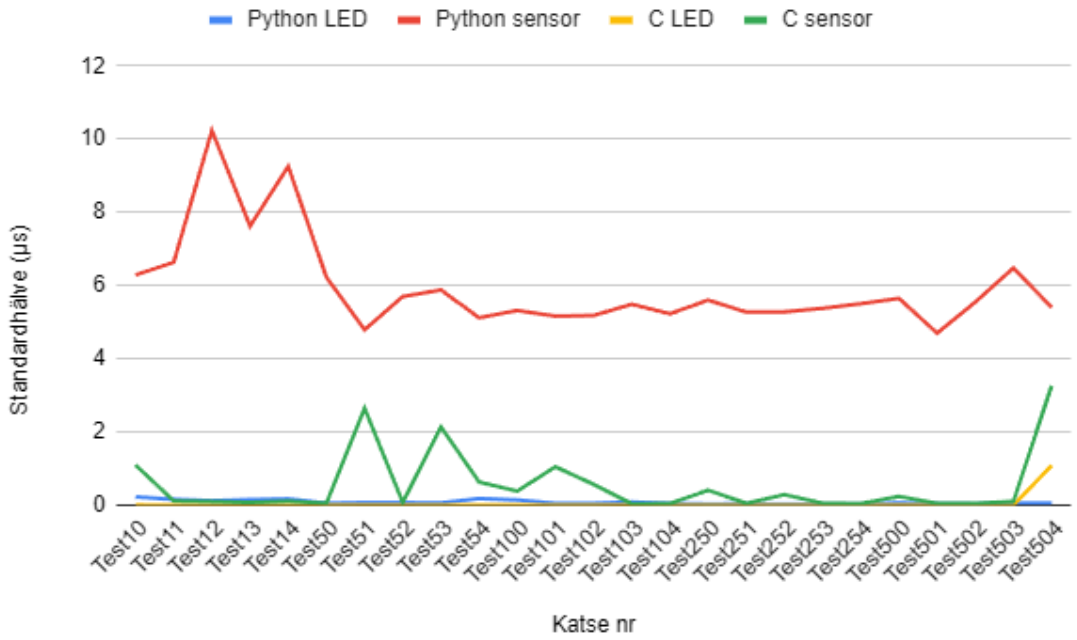
Tulemusena võib öelda, et kiiruse poolest on kõige parem ESP32-PoE-ISO. Kuigi siinkohal võib mängu tulla asjaolu, et testimise käigus Pi-de puhul kirjutati andmed faili ning ESP32 puhul mitte, sest SD-kaardile kirjutamist ei saadud tööle. Üllatav on ka asjaolu, et Pi-de puhul on Python umbes 4 korda aeglasem kui seda on C. Seda enam, et koodis on faili kirjutamised täpselt samades kohtades.

5.3.2 Täpsuste võrdlus

Kui hakata mikrokontrollerite puhul rääkima täpsusest, siit tuleb kõigepealt arvesse võtta ka mikrokontrolleri protsessori kiirust. Raspberry Pi 4B-l on see 1,5 GHz ning Pi 5 puhul 2,4 GHz. ESP32-PoE-ISO puhul on see aga 240 MHz, mis võrreldes näiteks Pi 5-ga on lausa 10 korda aeglasem. Et neid võimekusi ka praktilises mõttes üksteisega võrrelda, tehti selleks mikrokontrollerite peal ka eraldi testid.

Täpsuse mõõtmiseks lahutati nii LEDi kui ka fototakisti puhul igast järgnevast ajast, mil LEDi puhul see vilkus ning sensori puhul mil see valgust tundis ära eelnev aeg. Saades teada vahe iga oleku muutuse vahel, oli võimalik standardhälve abil arvutada, kui palju see keskmisest erines.

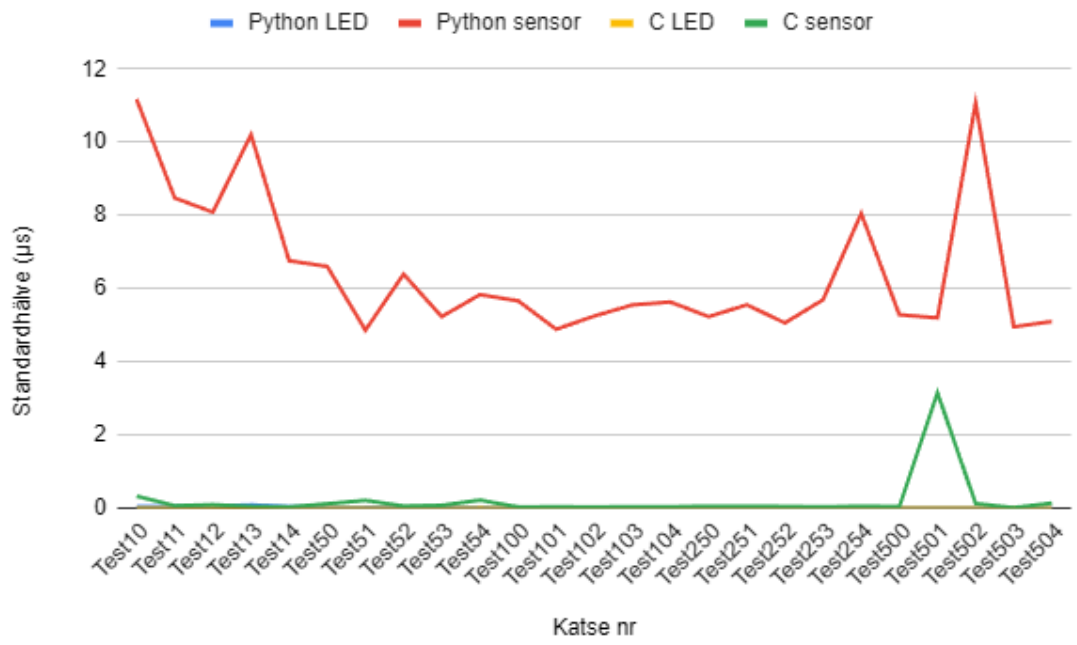
Järgnevalt on ära toodud graafikud keelte võrdlemiseks igal mikrokontrolleril.



Joonis 14. Raspberry Pi 4B standardhälbed.

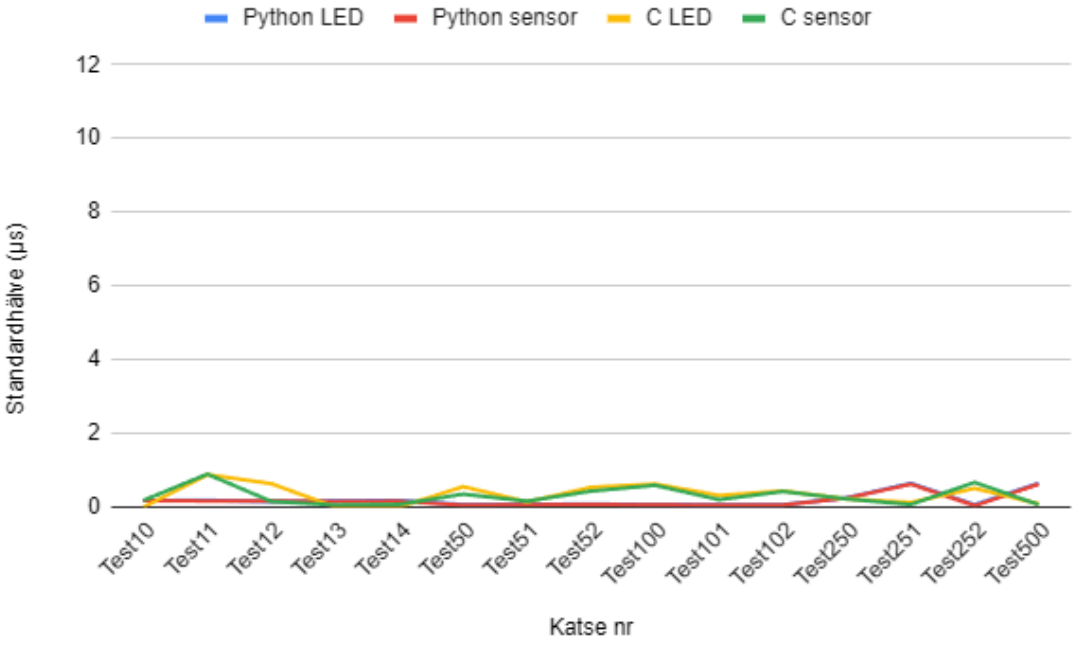
Joonisel 14 on näha, et mõlema koodi puhul on LEDi vilgutamise standardhälve suhteliselt väike. Kui aga vaadata fototakisti tunnetamist, siis selle puhul on standardhälve juba suurem ehk tegemist on ebastabiilsema keelega.

Raspberry Pi 5 puhul on tegemist sarnase tulemusega (Joonis 15). Mõlema keele puhul on valgusdioodi vilkumine veelgi stabiilsem, kuid Pythoni puhul on sensori tunnetamine ebastabiilsem samas kui C puhul on see stabiilsem.



Joonis 15. Raspberry Pi 5 standardhälbed.

ESP32 puhul on tulemused üpris ebastabiilsed (Joonis 16). Selle puhul tehti küll vähem teste ning vahede suurusjärk on väiksem, kuid siiski on näha, et eriti C puhul hüppab standardhälve suhteliselt palju võrreldes näiteks Raspberry Pi 5-ga.



Joonis 16. ESP32-PoE-ISO standardhälbed.

Parema ülevaate andmiseks on keskmised standardhälbed ära toodud ka tabelites. Tabelis 4 on näidatud vilkumise standardhälbed ning tabelis 5 sensori tunnetamise standardhälbed.

Tabel 4. LEDi vilkumise standardhälbed.

	Raspberry Pi 5	**Raspberry Pi 4B**	**ESP32-PoE-ISO**
Python	20,6 µs	82,6 µs	193,2 µs
C	1,2 µs	43,8 µs	304,1 µs

Tabel 5. Fototakistiga tunnetamise standardhälbed.

	Raspberry Pi 5	**Raspberry Pi 4B**	**ESP32-PoE-ISO**
Python	6473,4 µs	5959,3 µs	181,4 µs
C	205,6 µs	547,0 µs	304,1 µs

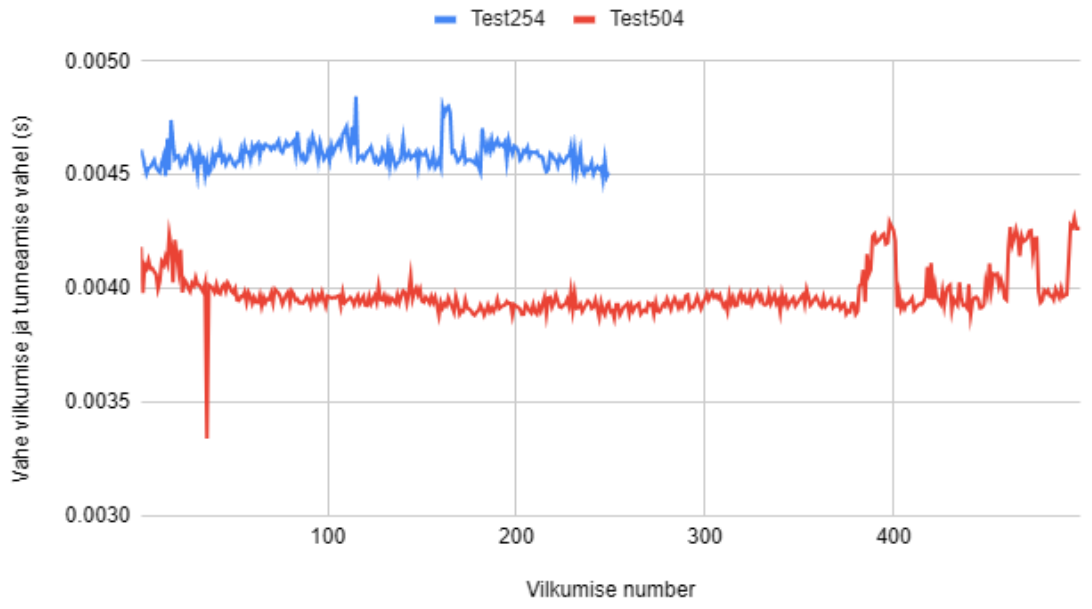
Vaadeldes kõiki keskmiseid standardhälbeid korraga, on näha, et C kood, mis kirjutati Raspberry Pi 5-le, oli kõige täpsem. Valguse tunnetamise puhul on küll ESP32 Pythoniga täpsem, kuid seda ainult 20 mikrosekundi võrra. Kuna Pi 5 on nii vilgutamise kui ka valguse tunnetamise osas üks kiiremaid, siis oleks see antud tulemuste põhjal parim valik.

5.3.3 Testide pikkuste põhjal tulemuste järjepidevuse võrdlus

Eelnevate tulemuste põhjal on üleüldse kõige paremad tulemused andnud Pi 5 ja C-keele kombinatsioon, seega et võrdlus liiga kirjuks ei läheks, võetakse võrreldavateks andmeteks just nendega tehtud testide tulemused. Erinevate testide pikkuste puhul võrdlemiseks on valitud kõikidest testidest viies test (Joonis 17, Joonis 18).

Joonis 17. Vilkumise ja tunnetamise vahe erinevate testide pikkuste puhul. Testid 14, 54 ja 104.

Lühematel testidel on selgelt näha, et nende alguses on vahed natuke suuremad ning peale mõnda LEDi vilgutust muutuvad need väiksemaks (Joonis 17). Samuti on seal näha, et vahel võivad tekkida suuremad hüpped nagu on juhtunud testi number 54 puhul.

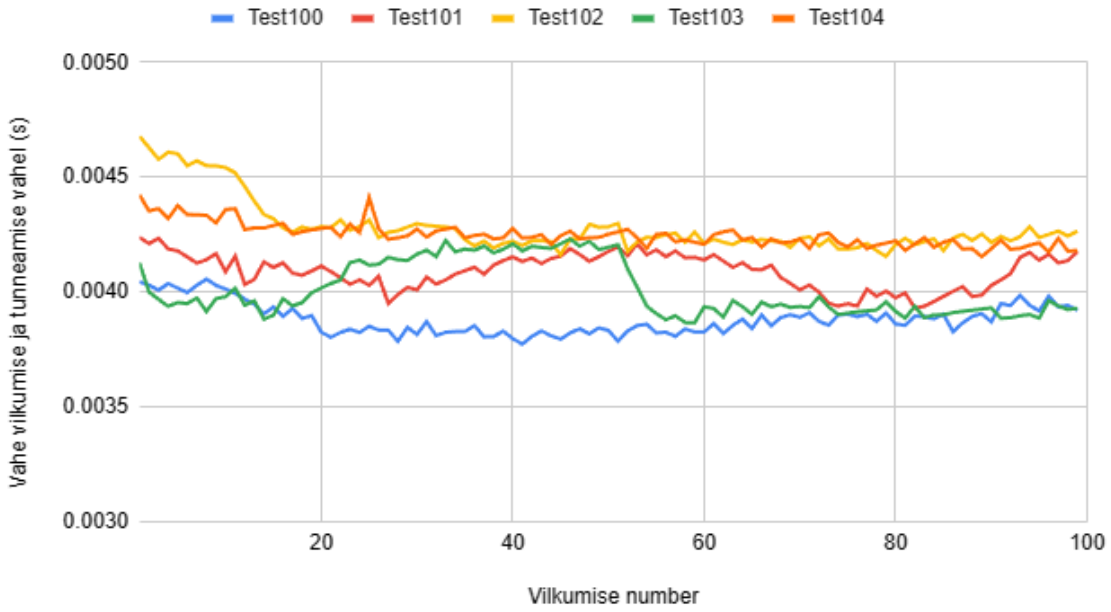


Joonis 18. Vilkumise ja tunnetamise vahe erinevate testide pikkuste puhul. Testid 254 ja 504.

Joonisel 18 on ära toodud testide 254 ja 504 tulemused. Ka nende puhul on kohati näha tavalisest suuremaid hüppeid.

Kuna on selgelt näha, et testide alguses on tavaliselt vahe natuke suurem ning siis läheb madalamaks, siis pigem ei ole hea mõte teha 10 vilgutusega teste, sest sellest võib jääda väheks, et saada adekvaatset tulemust. 500 vilgutusega test on aga testi lõpu poole üpris ebastabiilne.

Eeltoodud testidest on kõige stabiilsem number 104. Et näha, kas see on nii ka teiste 100 vilgutusega testide puhul, on need kõrvutatud joonisel 19.



Joonis 19. Vilkumise ja tunnetamise vahe erinevate testide pikkuste puhul. Testid 100 - 104.

Üleval toodud joonisel on näha, et antud vilgutuste arvuga test on üldiselt stabiilne. Jooniselt võib välja lugeda ka seda, et umbes peale 20-ndat LEDi vilgutust on vahe enamjaolt langenud keskmise väärtuse peale.

6 Tulemuste analüüs

Käesoleva bakalaureusetöö eesmärgiks oli valida ning võrrelda erinevaid mikrokontrollereid ning programmeerimiskeeli videopildi viite mõõtmise süsteemis kasutamiseks. Töö aluseks võeti eelnevas projektis valminud *proof-of-concept*, mida püüti edasi arendada.

Algselt valiti 4 mikrokontrollerit, kuid neist üks, Arduino Mega, jäeti lõpuks välja, sest sellel olid puudused, millega ei saaks seda antud süsteemis kasutada. Erinevalt teistest pole Arduino Megal võimalik NTP serverilt aega vähemalt millisekundi täpsusega küsida.

Ülejäänud mikrokontrollereid Raspberry Pi 5, Raspberry Pi 4B ja ESP32-PoE-ISO testiti ning võrreldi omavahel kasutades programmeerimiskeeli C ja Python. Võrreldi omavahel kiirust, täpsust ning stabiilsust. Täpsust ja stabiilsust mõõdeti süsteemis valgusdioodi vilkumiste aegade omavahel võrdlemisega ning sama tehti ka fototakisti valgustaseme muutuste tunnetamisega. Kiiruse arvutamiseks vaadeldi aega, mis läks hetkest kui LED läks põlema kuni hetkeni, mil fototakisti seda tunnetas.

Testide tulemusena võib järeldada, et parimaks kombinatsiooniks osutus kasutada Raspberry Pi 5-te koos C-keelega. Ka teised ei ole halvad valikud, kuid sellises süsteemis, kus täpsus ja kiirus on olulised, oleks just eeltoodud kombinatsioon kõige optimaalsem.

Lisaks võrreldi testide käigus viite erinevat vilgutuste arvu, ning püüti leida neist videopildi viite mõõtmise süsteemis kasutamiseks kõige optimaalsem. Sellise võrdluse tulemusena saadi, et parima tulemuse võiks anda 100 LEDi vilgutust, millest umbes 20 esimest tuleks maha arvestada, sest need on tavaliselt keskmisest kõrgema tulemusega. Samuti ei ole mõtet valida liiga suurt arvu, sest siis võivad tulemused liiga ebastabiilseks muutuda. Siinkohal võivad mängu tulla Pi-l teised jooksvad protsessid, mis võivad mõjutada testi tulemusi.

Edasiarendused

Kahjuks pole võimalik antud töö tulemusena öelda, et miks tekivad vahepeal testide tulemustes tavalisest suuremad hüpped, kuid sellega on siiski süsteemi loomisel hea arvestada. Edasiarendusena aga oleks hea näiteks teha täiendavaid teste ning sealjuures näiteks vähendada taustal jooksvaid protsesse või neid jälgida, et näha mis võib selle põhjuseks olla. Samuti oleks veel hea mõte uurida koodis erinevateks tegevusteks kulunud aega nagu näiteks faili kirjutamise puhul.

Lisaks võiks veel uurida ja testida teisi mikrokontrollereid, mis ei oleks Raspberry Pi-d, kuid on võimekamad kui antud töö käigus kasutatud ESP32-PoE-ISO, sest nii võiks ehk leida stabiilsema alternatiivi Pi-le kui seda on antud ESP32.

Kokkuvõte

Tehtud töö tulemusena leiti, et parim valik videopildi viite mõõtmise süsteemi jaoks oleks Raspberry Pi 5 ning seda võiks programmeerida C-keeles. Samuti tuli töö käigus välja, et näiteks üheks valikuks olnud Arduino Mega ei sobi üldse antud ülesandeks.

Bakalaureusetöö jaoks püstitatud eesmärgid said täidetud. Erinevaid mikrokontrollereid ning programmeerimiskeeli sai võrreldud ning nende seast leiti ülesandeks sobivaim. Edasiarenduseks võiks antud töö puhul tulevikus mõõta ära näiteks aja, mis kulub Pythoni ja C puhul failidesse kirjutamiseks, sest hetkel pole kindel kas see mängis näiteks Pythoni puhul suurt rolli märgatavalt suurema viite tekkimisel fototakistiga valguse tunnetamisel. Samuti võiks teha täiendavaid teste kui peatada võimalikult palju Linuxi peal jooksvaid taustaprotsesse ning seeläbi vaadata, kas on võimalik saada stabiilsemaid tulemuseid.

Töö käigus õppis autor üpris palju erinevate mikrokontrollerite ning nende töötamise kohta. Seda ka eelnevalt kasutatud Raspberry Pi puhul. Samuti aitas tehtud töö mõista kui tähtis on enne töö alustamist oma valitud tööriistade kohta põhjalikult uurida ning planeerida juba varakult ära mida täpselt hakatakse testimise käigus tegema. Sedasi saab ennetada juba tekkivaid probleeme ning hoida aega kokku testide kordamiselt. Sellegipoolest oli töö väga kasulik ning aitas kinnistada autoril eelnevalt erialal õpitut.

Kasutatud kirjandus

[1] A. Mändmets, L. Kõrgmaa, K. Paabut ja A. Prääm, „Irdtorni videoühenduse viite mõõtmine,“ Tallinn, 2023.

[2] Lennuliiklusteeninduse aktsiaselts, „Kaugjuhitav lennujuhtimistorn | EANS,“ Lennuliiklusteeninduse aktsiaselts, [Võrgumaterjal]. Available: https://www.eans.ee/tegevused/arendustegevused/kaugjuhitav-lennujuhtimistorn. [Kasutatud 4, Apr. 2024].

[3] C. Christian, „20 Awesome Raspberry Pi Projects Anyone Can Do,“ Make Use Of, 21, Oct. 2023. [Võrgumaterjal]. Available: https://www.makeuseof.com/tag/different-uses-raspberry-pi/. [Kasutatud 16, Apr. 2024].

[4] A. Locker, „50 Cool Raspberry Pi Projects for April 2024 | All3DP,“ All3DP, 4, Apr. 2024. [Võrgumaterjal]. Available: https://all3dp.com/1/best-raspberry-pi-projects/. [Kasutatud 16, Apr. 2024].

[5] Raspberry Pi, „Buy a Raspberry Pi Pico - Raspberry Pi,“ Raspberry Pi, [Võrgumaterjal]. Available: https://www.raspberrypi.com/products/raspberry-pi-pico/. [Kasutatud 16, Apr. 2024].

[6] Raspberry Pi, „Buy a Raspberry Pi Zero - Raspberry Pi,“ Raspberry Pi, [Võrgumaterjal]. Available: https://www.raspberrypi.com/products/raspberry-pi-zero/. [Kasutatud 16, Apr. 2024].

[7] Raspberry Pi, „Buy a Raspberry Pi 5 - Raspberry Pi,“ Raspberry Pi, [Võrgumaterjal]. Available: https://www.raspberrypi.com/products/raspberry-pi-5/. [Kasutatud 16, Apr. 2024].

[8] Raspberry Pi, „Raspberry Pi - About us,“ Raspberry Pi, [Võrgumaterjal]. Available: https://www.raspberrypi.com/about/. [Kasutatud 16, Apr. 2024].

[9] [Võrgumaterjal]. Available: https://portworld-solu.com/wp-content/uploads/2021/05/2-1-2.jpg. [Kasutatud 25, Apr. 2024].

[10] Raspberry Pi, „Introducing Raspberry Pi 5! - Raspberry Pi,“ 28, Sep. 2023. [Võrgumaterjal]. Available: https://www.raspberrypi.com/news/introducing-raspberry-pi-5/. [Kasutatud 25, Apr. 2024].

[11] L. Pounder, „How to Control the Raspberry Pi 5 GPIO with Python 3 | Tom's Hardware,“ 28, Oct. 2023. [Võrgumaterjal]. Available: https://shorturl.at/atIR2. [Kasutatud 21, Apr. 2024].

[12] joan, „Raspberry Pi 5 - gpiod vs RPi.GPIO - Raspberry Pi Forums,“ 16, Nov. 2023. [Võrgumaterjal]. Available: https://shorturl.at/apsI5. [Kasutatud 21, Apr. 2024].

[13] ptmaker, „How ESP32 microcontrollers works & Use cases - Hive,“ 2023. [Võrgumaterjal]. Available: https://hive.blog/hive-163521/@ptmaker/how-esp32-microcontrollers-works-and-use-cases. [Kasutatud 21, Apr. 2024].

[14] Olimex, „ESP32-POE-ISO - Open Source Hardware Board,“ Olimex, [Võrgumaterjal]. Available: https://www.olimex.com/Products/IoT/ESP32/ESP32-POE-ISO/open-source-hardware. [Kasutatud 16, Apr. 2024].

[15] [Võrgumaterjal]. Available: https://nettigo.eu/system/images/4014/original.jpg?1633205607. [Kasutatud 25, Apr. 2024].

[16] Olimex, „ESP-POE-ISO-GPIO.png (1920x1280),“ [Võrgumaterjal]. Available: https://www.olimex.com/Products/IoT/ESP32/ESP32-POE-ISO/resources/ESP32-POE-ISO-GPIO.png. [Kasutatud 16, Apr. 2024].

[17] Arduino, „Ethernet Shield Rev2 | Arduino Documentation,“ Arduino, [Võrgumaterjal]. Available: https://docs.arduino.cc/hardware/ethernet-shield-rev2/. [Kasutatud 21, Apr. 2024].

[18] [Võrgumaterjal]. Available: https://images.tcdn.com.br/img/img_prod/715570/arduino_mega_2560_r3_rev3_833_1_20201213232553.jpg. [Kasutatud 25, Apr. 2024].

[19] M. Rouse, „What is a Real-Time Clock (RTC)? - Definition from Technopedia,“ Technopedia, 12, May. 2015. [Võrgumaterjal]. Available: https://www.techopedia.com/definition/2273/real-time-clock-rtc. [Kasutatud 25, Apr. 2024].

[20] I. Ayyub, „Python (Pymite-09) on Arduino Mega 2560 with ATmega 16 MCU,“ Atmel-avr, 6, Jun. 2012. [Võrgumaterjal]. Available: https://atmega32-avr.com/using-atmega16-micrcontroller-running-python-pymite-09-on-an-arduino-mega-2560/. [Kasutatud 28, Apr. 2024].

[21] R. Candido, „Arduino With Python: How To Get Started - Real Python,“ Real Python, [Võrgumaterjal]. Available: https://realpython.com/arduino-python/. [Kasutatud 22, Apr. 2024].

[22] R. Bergsma, „How accurately can the Raspberry Pi keep time? << Remi Bergsma's blog,“ 12, May. 2013. [Võrgumaterjal]. Available:

https://blog.remibergsma.com/2013/05/12/how-accurately-can-the-raspberry-pi-keep-time/. [Kasutatud 16, Apr. 2024].

[23] „GL5528.xls - SEN-09088.pdf,“ [Võrgumaterjal]. Available: https://cdn.sparkfun.com/datasheets/Sensors/LightImaging/SEN-09088.pdf. [Kasutatud 28, Apr. 2024].

[24] Microchip, „MCP3008 | Microchip Technology,“ Microchip Technology Inc., [Võrgumaterjal]. Available: https://www.microchip.com/en-us/product/mcp3008#document-table. [Kasutatud 28, Apr. 2024].

[25] Arduino, „Software | Arduino,“ Arduino, [Võrgumaterjal]. Available: https://www.arduino.cc/en/software. [Kasutatud 25, Apr. 2024].

[26] Thonny, „Thonny, Python IDE for Beginners,“ Thonny, [Võrgumaterjal]. Available: https://thonny.org/. [Kasutatud 25, Apr. 2024].

[27] L. S. Sterling, The Art of Agent-Oriented Modeling, London: The MIT Press, 2009.

[28] Arduino, „precise_sntp - Arduino Reference,“ Arduino, [Võrgumaterjal]. Available: https://reference.arduino.cc/reference/en/libraries/precise_sntp/. [Kasutatud 22, Apr. 2024].

[29] R. Teja, „Arduino Mega Pinout | Arduino Mega 2560 Layout, Specifications,“ ElectronicsHub, 1, Apr. 2024. [Võrgumaterjal]. Available: https://www.electronicshub.org/arduino-mega-pinout/. [Kasutatud 22, Apr. 2024].

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

Mina, Anee Mändmets

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Mikrokontrollerite ja programmeerimiskeelte võrdlus videopildi viite mõõtmisel“, mille juhendaja on Priit Roosipuu.
   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.

13.05.2024

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 – Koodid

Kirjutatud koodid on võimalik leida lingilt:

https://github.com/AneeMandmets/reval/tree/main