Tampereen yliopisto

Lassi Ollinen

LEGO JUNAN OHJAAMINEN INFRAPU-NALÄHETTIMELLÄ

Kandidaatintyö
Tekniikan ja luonnontieteiden tiedekunta
Elokuu 2021

TIIVISTELMÄ

Lassi Ollinen: LEGO Junan ohjaaminen infrapunalähettimellä
Tampereen yliopisto
Teknisten tieteiden kandidaatin tutkinto-ohjelma
Kandidaatintyö
Elokuu 2021

Kandidaatintyössä tutkittiin reaaliaikaisen sovelluksen suunnittelua ja toteutettiin tapausesimerkkinä LEGO Power Functions RC -protokollalla ohjattavan LEGO Passenger Trainin® ohjausjärjestelmä. Ohjain toteutettiin käyttäen Raspberry Pi® -piiritietokonetta ja siihen liitettyä infrapunalähetintä siten, että sen voi jatkokehityksessä liittää Beckhoffin® automaatiologiikkaan. Kandidaatintyön keskeisimmät tutkintokysymykset ovat: kuinka toteuttaa mikrosekunnin reaaliaikaisuutta vaativa infrapunalähetin käyttäen Raspberry Pi -piiritietokonetta, kuinka huomioida automaatiojärjestelmältä vaadittavat luotettavuusvaatimukset ja miten Beckhoffin automaatiologiikka tulee huomioida ohjaimen ohjelmistorajapinnassa.

Työn teoriaosuudessa käsitellään Raspberry Piin liittimet ja rakenne sekä Raspbian® -käyttöjärjestelmän asentaminen. Ohjelmointia tarkastellaan Pythonin ja C++:n näkökulmasta keskittyen erityisesti C++:n ja Pythonin saatavilla oleviin ohjelmointikirjastoihin sekä näiden etuihin ja haittoihin etenkin reaaliaikaisuuden näkökulmasta. Lisäksi käsitellään LEGO Power Functions RC -protokollan signaalikoodauksen toiminta ja sen asettamat reaaliaikavaatimukset. Lisäksi käsitellään kyseisen protokollan eri signaalimuodot ja miten muodostaa infrapunakäsky protokollan ”single output mode” -signaalimuotoa käyttämällä.

Työn käytännön osuudessa ohjaimen tekeminen aloitettiin Python-ohjelmointikielellä, mutta lopullinen ohjain toteutettiin C++-ohjelmointikielellä. Tähän päädyttiin, koska oskilloskoopilla tarkasteltaessa todettiin, että Python ei kykene tuottamaan mikrosekunnin reaaliaikaisuutta vaativia signaaleja. Lopullinen ohjelma toteutettiin käyttäen avuksi C++:n ”Nanosleep”- ja ”WiringPi” -ohjelmointikirjastoja. Toteutuksessa keskityttiin etenkin siihen, miten mikrosekunnin reaaliaikaisuutta vaativa infrapunasignaali lähetetään Raspberry Piillä luotettavasti siten, että se täyttää automaatiojärjestelmälle keskeiset luotettavuusvaatimukset.

Työn teoriaosuuden pohjalta saatiin lopputuloksena toimiva ohjausjärjestelmä, joka täyttää järjestelmälle asetetut reaaliaikavaatimukset sekä luotettavuusvaatimukset lähettimen sijainnin ennakkoehtojen täyttyessä. Lisäksi jatkokehityksessä ideoidaan, kuinka luotettavuutta saisi parannettua vaihtamalla infrapunaledi tehokkaampaan ja signaalin reaaliaikaisuutta tarkentamalla. Lisäksi huomioidaan se, miten Raspberry Piillä toteutetun ohjaimen saisi liitettyä Beckhoffin automaatiologiikkaan OPC®-rajapintaa käyttämällä tai yksinkertaisemmin C++:n ”WiringPi”-ohjelmointikirjastoa hyödyntäen.

Avainsanat: LEGO®, LEGO Power Functions RC, ohjausjärjestelmä, Raspberry Pi®, infrapunalähetin, Beckhoff®, Python, C++, WiringPi, Nanosleep

SISÄLLYSLUETTELO

1. JOHDANTO 1
2. RASBERRY PI JA OHJELMOINTIKIELET 3
2.1 Raspberry Pi 4 B 1GB: rakenne ja käyttöjärjestelmät 3
2.2 Raspberry Pi 4 B 1GB: liittimet. 5
2.3 Pythonin teoriaa 7
2.4 C++-teoriaa 8
3. LEGO PASSENGER TRAININ OHJAUS 10
3.1 LEGO Power Functions RC-protokollan eri signaalimuodot 12
3.2 LEGO Power Functions RC-signaalikoodaus 12
3.3 Single output mode 14
4. KÄYTÄNNÖN TOTEUTUS 17
4.1 Järjestelmän vaatimukset 17
4.2 Suunnitteluprosessi 18
4.3 Ohjaimen toteutus ja haasteet 20
4.3.1 Pythonin reaaliaikaisuuden haasteet ja siirtyminen C++ -ohjelmointikieleen. 20
4.3.2 Geanyn ja GPIO-pinnien nimeämisen aiheuttamat haasteet 22
4.3.3 Lopullisen ohjelmakoodin toteuttaminen C++:lla 23
5. LOPPUTULOS JA JATKOKEHITYS 29
6. YHTEENVETO 32
LÄHTEET 34

KUVALUETTELO

Kuva 1. Raspberry Pi 4 Model B®:n rakenne. [7] *4*
Kuva 2. Raspberry Pi Imagerin käyttöliittymä [9] *5*
Kuva 3. Raspberry Pi -piiritietokoneen “GPIO-header” [4] *6*
Kuva 4. Raspberry Piin pinnien eri numerointitekniikat. GPIO perässä on GPIO.BCM-numerointitekniikalla numeroidut pinnit, suluissa on ”WiringPiin” pinnien numerointijärjestelmä ja keskimmäisenä on pinnien ”header”-numerointijärjestelmä. [16] *8*
Kuva 5. Kandidaatintyössä käytetty LEGO Passenger Train®. *11*
Kuva 6. IR-signaalin rakenne. [1] *14*
Kuva 7. ”Single output mode”:n käskyt ”data”-bitteinä ***D****. [1]* *15*
Kuva 8. Infrapunalähettimen piirikaavio *19*
Kuva 9. Pythonilla toteutettu ”increment PWM” -komento. *21*
Kuva 10. Geanyn ”set build commands"-ikkunan asetukset, joilla ”WiringPi” toimii. *22*
Kuva 11. Ohjelmakoodi, joka tauottaa noin 13 µs ajaksi käyttäen ”while”-looppia. *24*
Kuva 12. IR-syklin tilan vaihteluiden pituus oskilloskoopilla mitattuna. *25*
Kuva 13. Ykkösbitin luontiin tarkoitettu funktio ”one( )”. *26*
Kuva 14. Metodin ”initcommand” toteutus. *27*
Kuva 15. ”PWM forward step 7” käynnistävä ”incseven”-metodi. *28*
Kuva 16. Lopullisen ohjaimen ”brake, then float”-komento. *30*

LYHENTEET JA MERKINNÄT

PWM	Pulssinleveysmodulaatio
IR	*Infrapuna*
IR-LED	*Infrapunaledi*
RC	*Radio-ohjattava*
RAM	*engl. Random Access Memory, keskusmuisti*
GPIO	*General Purpose Input/Output*
GB	*engl. gigabyte, gigatavu*
RasPi	*Raspberry Pi*

1. JOHDANTO

Tässä kandidaatintyössä tutkitaan reaaliaikaisuutta vaativan sovelluksen toteuttamista ja toteuttaa reaaliaikaisuutta vaativa sovellus käytännön tapausesimerkin pohjalta. Kandidaatintyössä keskitytään etenkin infrapunasignaalin eli IR-signaalin lähettämiseen ja siihen miten sen vaativan kovan reaaliaikaisuuden saa toteutettua sekä teoriassa että käytännössä käyttäen Raspberry Pi-piiritietokonetta.

Käytännön osuudessa käytetään tapausesimerkkinä LEGO Passenger Trainia, jonka taustalla on automaatiotekniikan laitoksen LEGO Cyber City® -ympäristö, joka tulee lopulta koostumaan useista kandidaatintöistä. Valmis LEGO Cyber City -ympäristö tulee pitämään sisällään useita eri osakokonaisuuksia esimerkiksi höyryvoimalan pienoismallin, LEGO Passenger Trainin® ja LEGO Cyber Cityn valaisujärjestelmän. Näitä kaikkia olisi tarkoitus onnistua ohjaamaan Beckhoffin® automaatiojärjestelmällä, joka sijaitsee LEGO Cyber City -ympäristössä. Tässä kandidaatintyössä reaaliaikaisuuden käytännön tapausesimerkkinä toteutetaan automaatioympäristön osa, jossa LEGO Passenger Trainia ohjataan Rasberry Piihin® integroidulla infrapunalähettimellä. Kandidaatintyöhön ei sisälly Rasberry Pin integroiminen osaksi Beckhoffin automaatiojärjestelmää mutta se huomioidaan ohjelmiston rajapintaa suunniteltaessa. Koska kyseessä on automaatiojärjestelmä niin ohjainta suunniteltaessa ja toteuttaessa onkin erittäin tärkeää, että ohjaimen luotettavuus ja reaaliaikaisuus ovat riittävät. Kandidaatintyö on jaettu neljään osaan, joissa käsitellään työn suorituksen eri osa-alueita.

Toisessa luvussa käsitellään Rasberry Pi -piiritietokonetta sekä sille reaaliaikaisen ohjelmiston toteuttamista teoreettisesti neljästä eri näkökulmasta. Ensimmäisessä alaluvussa kerrotaan Rasberry Pi -piiritietokoneen rakenteesta ja mitä käyttöjärjestelmiä siihen on saatavilla. Toisessa alaluvussa käsitellään Raspberry Piin liittimiä ja mitä eri palveluita ne tarjoavat ohjelmistoille. Lisäksi keskitytään erityisesti Raspberry Piin ”General Purpose Input/Output (GPIO) -headerin” pinneihin ja niiden toimintaan sekä siihen miten niillä voidaan mahdollistaa kovaa reaaliaikaisuutta vaativa järjestelmä teoriassa. Kolmannessa ja neljännessä alaluvussa käsitellään ohjelmointikieliä keskittyen pääasiassa Raspberry Piin tarjoamiin ohjelmointikirjastoihin, sekä siihen miten niillä voi teoriassa toteuttaa kovaa reaaliaikaisuutta vaativan IR-lähettimen. Luvussa kerrotaan myös ohjelmointikirjastoista, joita voisi käyttää samantyylisen ohjelmiston toteuttamisessa tai tämän kandidaatintyön tapausesimerkin jatkokehityksessä.

Kolmannessa luvussa siirrytään teoriasta tapausesimerkkiin ja keskitytään LEGO Passenger Trainiin ohjaamisen näkökulmasta sekä siihen, mitä vaatimuksia se luo reaaliaikaisen järjestelmän toteutukselle. Luvussa keskitytään erityisesti LEGO City Passenger Trainiin ja sen infrapunavastaanottimen toimintaan käyttäen avuksi LEGO Power Functions RC® -protokollan dokumentaatiota [1]. Erityisesti keskitytään protokollan Single Output modeen sekä, miten sen signaalikoodausta käyttäen saadaan lähetettyä bittejä infrapunasignaalina. Lisäksi luvussa käsitellään lyhyesti ohjelmistovaatimuksia, joita Bechoffin automaatiologiikka luo ohjelmiston rajapinnalle.

Neljännessä luvussa käsitellään alaluvuittain käytännön tapausesimerkin toteutusta alkaen järjestelmän vaatimuksista, joissa käydään lävitse mitä vaatimuksia LEGOn infrapunavastaanotin asettaa ohjelmistolle ja Raspberry Pi -piiritietokoneelle. Toisessa alaluvussa kerrotaan suunnitteluprosessista ja siitä, miksi erillisiin osaratkaisuihin päädyttiin. Kolmannessa alaluvussa käydään läpi työn toteutus sekä haasteet. Lopulta neljännessä alaluvussa kerrotaan, millainen lopullisesta ohjelmakoodista tuli ja mitkä tekijät vaikuttivat siihen.

Viidennessä luvussa käsitellään työn lopputulos ja se, kuinka hyvin tapausesimerkin lopputulos vastaa työn alussa esitettyjä tavoitteita reaaliaikaisuuden ja luotettavuuden osalta. Lisäksi tässä luvussa kerrotaan, miten projektia voisi jatko kehittää niin ohjelmiston kuin fyysisen virtapiirin osalta. Jatkokehityksessä keskitytään erityisesti Raspberry Piin -ohjelmointikirjastoihin, joilla saataisiin parannettua ohjelman toimintaa.

2. RASBERRY PI JA OHJELMOINTIKIELET

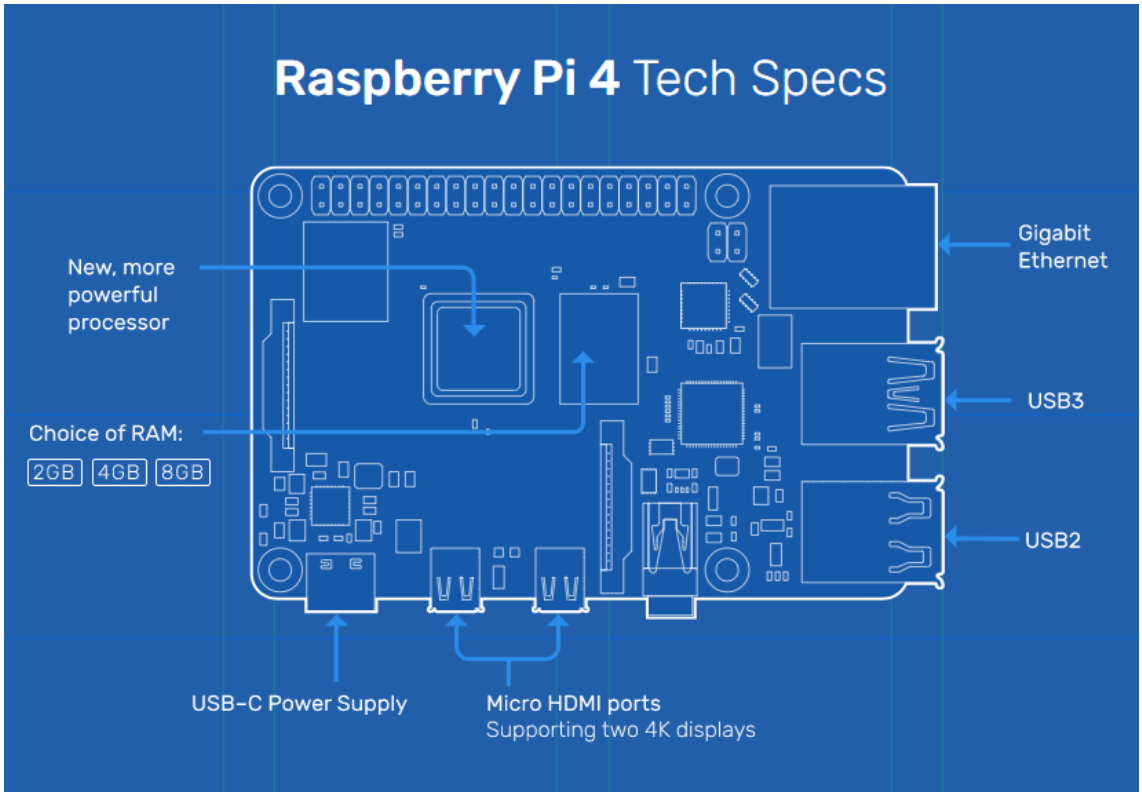
Raspberry Pi tai lyhennettynä ”RasPi” on Raspberry Pi Foundationin® valmistama piiritietokone, jota käytetään useissa eri tietotekniikan applikaatioissa [3]. Tämän kandidaatintyön toteutuksessa käytetään Raspberry Pi 4 B®:n yhden gigatavun RAM-muistilla varustettua versiota, koska sellainen oli saatavilla Tampereen Yliopistolta ja koska LEGO Passenger Trainin infrapunalähettimen ohjelmisto ei tule tarvitsemaan paljon laskentatehoa.

Tässä luvussa käsitellään kandidaatintyössä käytettävää Raspberry Pi -piiritietokonetta ja sen ohjelmoitia neljästä eri näkökulmasta. Ensimmäisessä luvussa 3.1 käsitellään Raspberry Piitä rakenteen ja käyttöjärjestelmävaihtoehtojen osalta eli miten Raspberry Pi toimii ja mitä käyttöjärjestelmiä sille on saatavilla. Toisessa luvussa 3.2 asiaa käsitellään Raspberry Piin liittimien osalta, keskittyen erityisesti Raspberry Pi-piirilevyn ”general-purpose input/output”-pinneihin eli lyhennettynä ”GPIO”-pinneihin, jotka mahdollistavat laajan datan sisään- ja ulostulon piiritietokoneesta eri applikaatioiden tarpeeseen. [4] Kolmannessa luvussa käsitellään Pythonin teoriaa sekä mitä kirjastoja ja funktioita Python tarjoaa reaaliaikaisen sovelluksen toteuttamiseen Raspberry Pillä. Kolmannessa alaluvussa kerrotaan, kuinka Python kirjastoja voidaan käyttää PWM-signaalin generointiin sekä yleisesti ledin sytyttämiseen ja sammuttamiseen. Neljännessä alaluvussa käsitellään C++ teoriaa ja mitä kirjastoja ja funktioita se tarjoaa reaaliaikaisen sovelluksen toteuttamista varten. Erityisesti käsitellään C++ kirjastoja, joita voidaan käyttää PWM-generointiin ja yleisesti ledin sytyttämiseen ja sammuttamiseen. Molemmissa kappaleissa on kuitenkin syytä huomioida, että PWM-signaalia generoivat koodit toimivat vain ja ainoastaan Raspberry Piin pinneissä, jotka tarjoavat PWM-signaalin generointiin tuen. Nämä pinnit luetteloidaan luvussa 2.2. Raspberry Pi 4 1GB® -liittimet sivulla 6.

2.1 Raspberry Pi 4 B 1GB: rakenne ja käyttöjärjestelmät

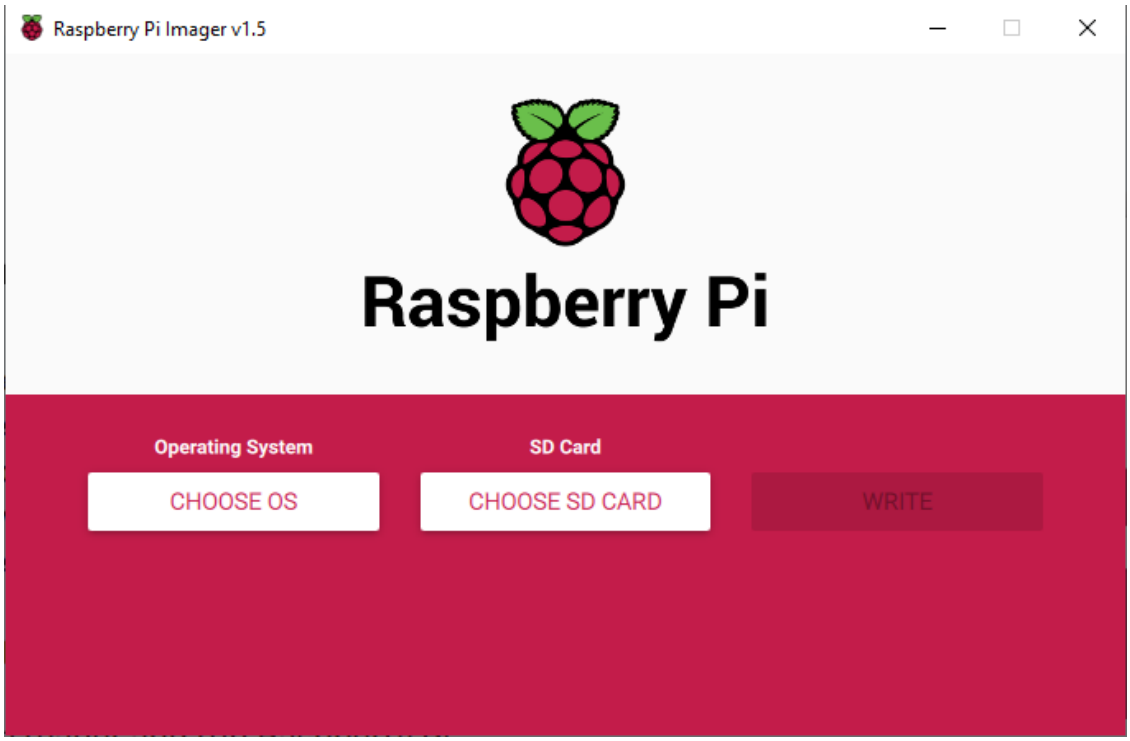
Rasberry Pi on rakenteeltaan yhdelle piirilevylle rakennettu tietokone (Kuva 1), joka tarvitsee toimiakseen kaiken sen mitä normaali pöytätietokonekin tarvitsee. Tällaisia asioita ovat esimerkiksi virtalähde, näyttö, hiiri ja näppäimistö. Tämän kandidaatintyön toteutuksessa käytettiin Raspberry Piin virtalähteenä Raspberry Pi 4:n virallista virtalähdettä [8]. Raspberry Pi 4 B:tä on tarjolla yhden GB-, kahden GB-, neljän GB- ja kahdeksan GB

RAM-muistilla varustetut mallit ja niiden hinnat ovat 35 yhdysvaltojen dollarista ylöspäin [3].



Kuva 1. *Raspberry Pi 4 Model B®:n rakenne. [7]*

Raspberry Piihin on saatavilla useita eri käyttöjärjestelmiä, jotka saa helpointen asennettua Raspberry Piihin käyttäen Raspberry Pi:n virallista Raspberry Pi Imager®-asennusohjelmaa (Kuva 2) [9]. Raspberry Pi Imager vaatii toimiakseen tietokoneen, jossa on ”SD®”-kortin lukija. Raspberry Pi Imagerinin käyttö on yksinkertaista, ensimmäisenä valitaan Raspberry Piihin haluttu käyttöjärjestelmä valikosta ”Choose OS”, ja tämän jälkeen se kirjoitetaan haluttuun ”microSD®”-korttiin. Käyttöjärjestelmän latauduttua ”microSD”-korttiin se asetetaan Raspberry Piin pohjassa olevaan ”microSD”-kortin porttiin (Kuva 1), jonka jälkeen Raspberry Pi on käyttövalmis.

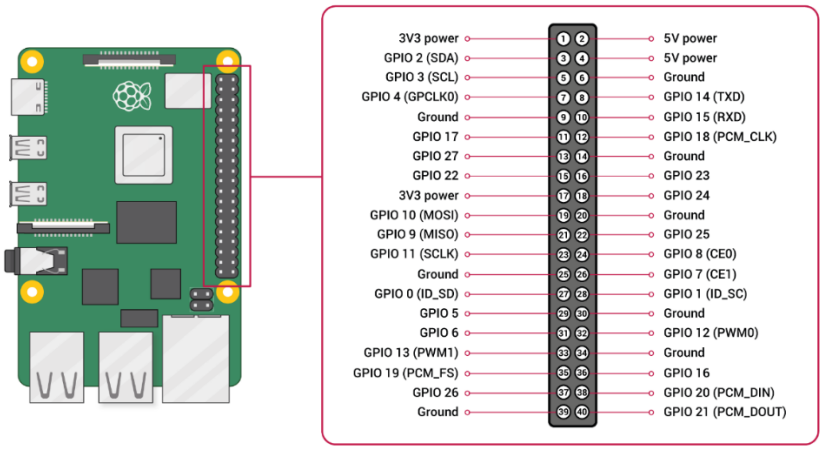


Kuva 2. *Raspberry Pi Imagerin käyttöliittymä [9]*

Raspberry Pi Imagerin kautta Raspberry Piihin saa asennettua useita eri käyttöjärjestelmiä, joista suosituin on ”Linux®”-pohjainen Raspbian®. Raspbian on yhteisörahoitettu ohjelmistokehitykseen tarkoitettu käyttöjärjestelmä, joka pitää sisällään yli 35 000 valmista ohjelmistopakettia, jotka helpottavat sen käyttöä. [10] Rasberry Pi on osien osalta kuin mikä tahansa tietokone. Raspbianin lisäksi siihen on saatavilla useita eri kolmannen osapuolen ylläpitämiä käyttöjärjestelmiä kuten Ubuntu®, LibreElec®, RetroPie® ja TLXOS®. [19]

2.2 Raspberry Pi 4 B 1GB: liittimet.

Aiemmin tässä kandidaatintyössä mainittiin Raspberry Pi-piirilevyn keskeisen osan luovat ”GPIO”-pinnit, jotka mahdollistavat Raspberry Pi-piiritietokoneen käytön erilaisissa tietoteknisissä sovelluksissa. Kaikki tällä hetkellä valmistettavat Raspberry Pi -piiritietokoneet onkin varustettu 40-pinnin ”GPIO-headerilla” (Kuva 3). [4]



Kuva 3. *Raspberry Pi -piiritietokoneen “GPIO-header” [4]*

Kuten kuvasta 3 nähdään niin Raspberry Piin ”GPIO-header” koostuu 40 pinnistä, joista seitsemän on maadoitus pinnejä, kaksi kappaletta 5V jännite pinnejä, kaksi kappaletta 3V3-jännitepinnejä ja 27 kappaletta on ”GPIO”-pinnejä, joiden sisääntuloa ja ulostuloa voidaan säätää ohjelmistossa useilla eri tavoilla. ”GPIO”-pinnit vastaavat jännitteeltään 3V3-pinnejä eli niiden jännite on 3.3 V, ja ne tarjoavat useita eri palveluita mahdollisten järjestelmien tarpeeseen. [4] ”GPIO”-pinnit tarjoavat erilaisia palveluita järjestelmän tarpeeseen kuten ”MOSI”, ”MISO”, ”SCLK, ID_SD”, ”PWM, PCM_FS”, ”TXD”, ”RXD”, ”PCM_CLK”, ”CE0”, ”CE1”, ”ID_SC”, ”PCM_DIN” ja ”PCM_DOUT”. [12]

Tämän kandidaatintyön kannalta olennainen palvelu reaaliaikaisuutta vaativaan IR-signaaliin luomiseen sekä LEGO Passenger Trainin ohjaamiseen on Raspberry Pi -piiritietokoneen tarjoama PWM eli ”Pulse-Width modulation”, jolla voidaan säätää tehon prosentuaalista syöttöä digitaalisesti. PWM siis simuloi analogista signaalia digitaalisessa muodossa ja PWM:n kaksi keskeisintä arvoa ovat taajuus (frequency) ja pulssisuhde (duty). Taajuus määrittää millä taajuudella PWM-signaalin päälle/pois tila vaihtelee. Sen sijaan pulssisuhde määrittää kuinka suuren osan ajasta signaalin tila on ”päällä”. Yleensä duty määritelläänkin prosentteina välillä 0–100 [13] Raspberry Piissä ”GPIO-header”:in pinneistä PWM-palvelua tarjoavat ”GPIO 12”, ”GPIO 13”, ”GPIO 18” ja ”GPIO 19”. [12]

”GPIO-header”:in ja ”microSD”-kortinlukijan lisäksi Raspberry Pi 4 B sisältää muutkin tietokoneelle tärkeät liittimet kuten kaksi kappaletta ”micro HDMI”-portteja näyttöjä varten, ”MIPI CSI”-kamera portin, 3.5 mm ääniliittimen, kaksi kappaletta ”USB 2.0”-liittimiä, kaksi kappaletta ”USB 3.0”-liittimiä, 1 GB ”ethernet”-liittimen, ”PoE HAT”-headerin, ”2.4/5 GHz

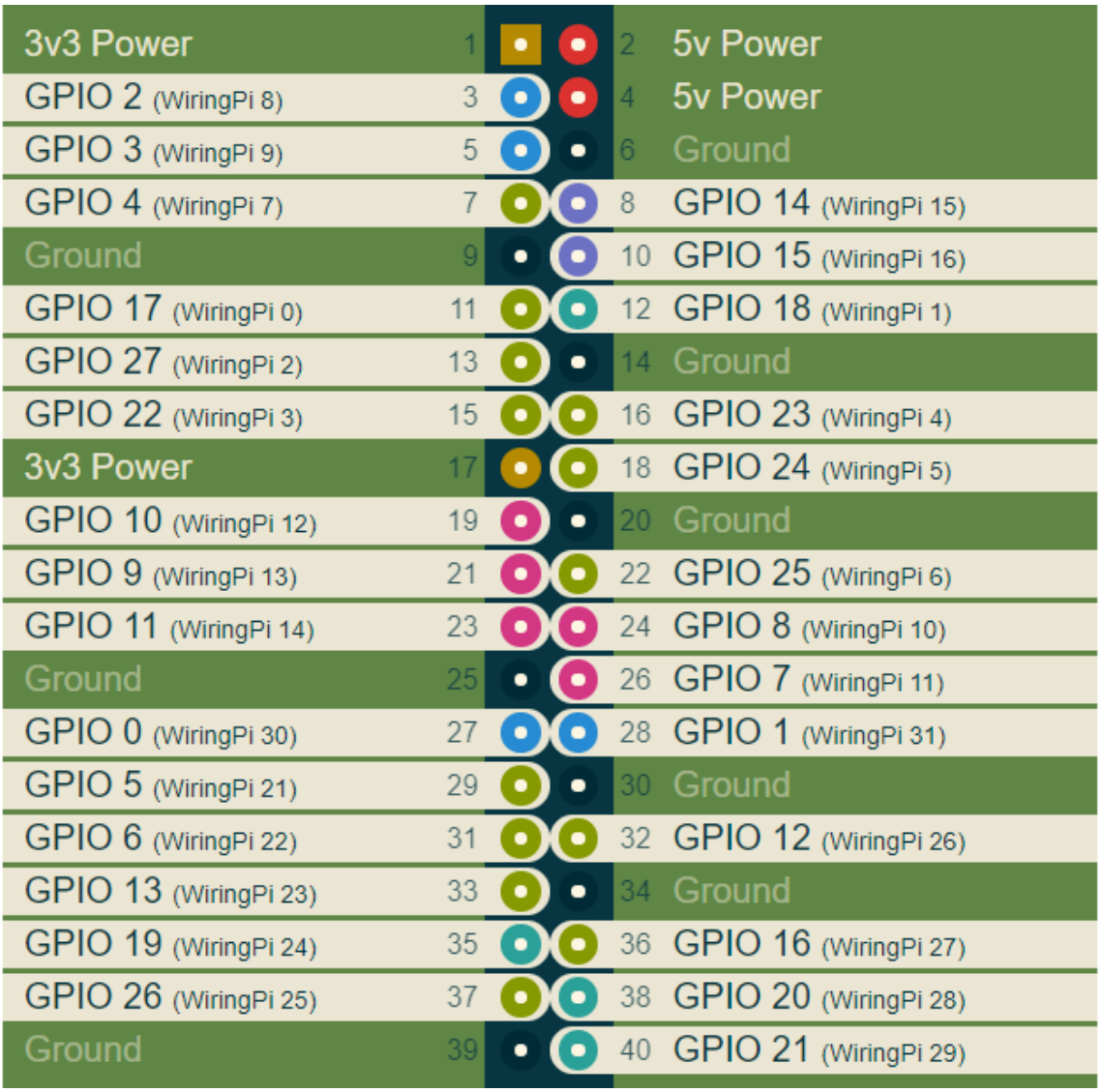
Wifi” ja ”Bluetooth 5.0”-kortin, ”MIPI DSI”-portin näytölle sekä ”USB-C”-portin virtalähdettä varten. [7]

2.3 Pythonin teoriaa

Python on syntaksiltaan hyvin yksinkertainen ohjelmointikieli, joka sopii erityisen hyvin niin uusille kuin kokeneillekin ohjelmoijille. Pythonin erityinen hyöty on se, että sen yksinkertainen syntaksi mahdollistaa hyvin nopean ohjelmoinnin [6]. Tosin yksinkertaisuuden kääntöpuolena on se, että monet monimutkaiset sovellukset, jotka voi toteuttaa C++:lla, ei ole käytännössä mahdollista toteuttaa Pythonilla kevyesti ja reaaliaikaisesti.

Raspberry Piitä ohjelmoitaessa Pythonilla on erityisen tärkeää saada käytettyä ”GPIO”-pinnejä. Tämä onkin tehty Pythonissa hyvin helpoksi, sillä Pythonissa on olemassa oma ”GPIO Zero”-kirjasto. Esimerkiksi ”GPIO Zeron” ledivalojen ohjaamiseen käytetyn ”LED”-komennon saa sisällytettyä ohjelmaan käskyllä ”$from\ gpiozero\ import\ LED$”. [14] Sisällyttämisen jälkeen saadaankin hyvin yksinkertaisesti määriteltyä ohjattava ”GPIO”-pinni komennolla: ”$led\ =\ LED(pin)$”, jossa pin merkitsee ”GPIO”-pinniä, josta halutaan lähettää virtaa ledille. Tämän jälkeen kyseisen pinnin tilan saa helposti määritettyä kahdella komennolla, jotka ovat: $led.on()$, jolloin pin lähettää virtaa ja komennolla ”$led.off()$”, jolloin pin ei lähetä virtaa. [14]

Lisäksi toinen Raspberry Piin ”GPIO”-pinnien ohjaamiseen tarkoitettu kirjasto on ”RPi.GPIO”. ”RPi.GPIO:n” etuna on se, että sillä saadaan helposti generoitua PWM-signaalia. ”RPi, GPIO”:n PWM-signaalin etuna on äärimmäinen tarkkuus, joka on noin $1\ \mu s$. Tämä onkin parempi kuin C++:n tarjoama PWM-signaalin reaaliaikaisuuden virhemarginaali. ”RPi.GPIO”:n saa lisättyä nimelle ”GPIO” seuraavalla komennolla: ”$import\ RPi.GPIO\ as\ GPIO$”. Tämän jälkeen saadaan asetettua kirjasto käyttöön komennolla: ”$GPIO.setmode(GPIO.BOARD)$”, joka asettaa pinnien numeroiksi pinnien omat ”header”-numerot (Kuva 4) tai komennolla: ”$GPIO.setmode(GPIO.BCM)$”, joka ottaa käyttöön ”BCM GPIO”-numeroinnin (Kuva 4). Esimerkiksi pinnin, jonka ”header”-numero on 12, niin ”BCM GPIO”-nimeämistekniikalla pinnin numero on 18. [13]



Kuva 4. *Raspberry Piin pinnien eri numerointitekniikat. GPIO perässä on GPIO.BCM-numerointitekniikalla numeroidut pinnit, suluissa on ”WiringPiin” pinnien numerointijärjestelmä ja keskimmäisenä on pinnien ”header”-numerointijärjestelmä. [16]*

Tämän jälkeen komennolla ”$GPIO.setup(pin, GPIO.OUT)$” saadaan määriteltyä minkä pinnin virransyöttöä halutaan ohjelmalla säädellä. Komennolla ”$piPWM = GPIO.PWM(pin, frequency)$” saadaan määriteltyä pinnin PWM-taajuus ja lopulta komennolla ”$piPWM.start(duty)$”, saadaan käynnistettyä PWM määritellyllä pulssisuhteella eli ”$duty$”:lla välillä 0–100. [13]

2.4 C++-teoriaa

Toisin kuin Python, C++ on syntaksiltaan monimutkaisempi kieli ja mahdollistaa luokkapohjaisen ohjelmistorakenteen. C++:n etu suhteessa Pythoniin on sen monimutkaisempi

syntaksi ja laajemmat kirjastot, jotka tarjoavat usein laajempia ja täsmällisempiä palveluita kuin Python. Nämä osaltaan mahdollistavat hyvin kevyen ja pitkälle optimoidun ohjelmistorakenteen joka itsessään mahdollistaa reaaliaikaisen järjestelmän suunnittelun ja toteutuksen.

Raspberry Piin ohjelmointiin C++ tarjoaa ”WiringPi”-kirjaston, jolla saa määriteltyä samaan tyyliin kuin Pythonin ”GPIO Zero”-kirjastolla ”GPIO-header”:in pinnien ulostuloja. ”WiringPi”-kirjaston saa sisällettyä C++ ohjelmaan komennolla ”$\#include < wiringPi.h >$”, tämän jälkeen saadaan määriteltyä ”WiringPi”:n komennolla ”$wiringPiSetup()$”. ”GPIO”-pinnejä saa ohjattua komennolla ”$pinMode(pin, OUTPUT)$”, jossa ”pin” on pinnin numero ”WiringPi”:n omalla pinnien numerointijärjestelmällä, joka on esitelty kuvassa 4. ”WiringPi”:n käyttämän pinnien numerointitekniikan näkee kuvassa 4 suluissa tekstin ”WiringPi” jälkeen. Lopulta kyseisen pinnin ulostulon saa määriteltyä kahdella komennolla ”$digitalWrite(pin, HIGH)$”, jolloin kyseisestä pinnistä lähetetään virtaa ja komennolla ”$digitalWrite(pin, LOW)$”, jolloin kyseisestä pinnistä ei lähetetä virtaa. [17]

Lisäksi ”WiringPi” tarjoaa PWM-generointiin tarkoitetut komennot, jotka toimivat muuten samoin mutta output määritellään PWM-generointia varten komennolla ”$pinMode(pin, PWM\_OUTPUT)$”. Tämän jälkeen PWM-generointi pinnissä saadaan käynnistettyä komennolla ”$pwmWrite(pin, duty)$”, jossa ”$duty$” määrittelee PWM:n pulssisuhteen välillä 0–100. [13]

3. LEGO PASSENGER TRAININ OHJAUS

Infrapunalähetin lähettää sähkömagneettista säteilyä, jota kutsutaan infrapunasäteilyksi. Tämän aallonpituus on näkyvää valoa suurempi mutta pienempi kuin mikroaaltojen. [2] Infrapunasäteily on erittäin hyödyllinen tiedon lähettämiseen langattomasti, koska infrapunalähettimet ovat rakenteeltaan yksinkertaisia ja halpoja valmistaa. Yleisin käytännön sovellus infrapunalähettimille onkin erilaiset langattomat ohjaimet. Tässä luvussa käydään lävitse tämän kandidaatintyön tapausesimerkkiä eli sitä, miten LEGO Passenger Trainin (Kuva 5) ohjaus infrapunalähettimellä teoriassa onnistuu sekä mitä reaaliaikavaatimuksia se asettaa sovelluksen toteutukselle. LEGO Passenger Train käyttää signaalin lähettämiseen LEGO Power Functions RC (Radio Control) ® -protokollaa [1], joka on LEGOn käyttämä langattomien ohjainten signaaliprotokolla, joka toimii infrapunalähettimellä ja vastaanottimella, joita käyttämällä LEGOn® RC-ohjain lähettää ohjaussignaaleja LEGO Passenger Trainille. Tässä luvussa keskitytään erityisesti siihen, miten kyseistä LEGOn protokollaa käyttämällä saadaan toteutettua kandidaatintyön ohjausjärjestelmään tarvittavat käskyt, joilla junan liikkumista voidaan hallita mahdollisimman luotettavasti. Lisäksi keskitytään siihen mitkä ovat LEGO Power Functions -protokollan asettamat reaaliaikavaatimukset.

Kuva 5. *Kandidaatintyössä käytetty LEGO Passenger Train®.*

Tämä luku koostuu kolmesta eri alaluvusta, joista ensimmäisessä käsitellään yleisesti ohjauksessa käytettävää LEGO Power Functions RC -protokollaa ja millaisia erilaisia mahdollisuuksia se tarjoaa LEGO Passenger Trainin ohjaamiseksi. Toisessa luvussa käsitellään kuinka LEGO Power Functions RC -protokolla lähettää bittejä sekä lasketaan reaaliajalliset vaatimukset IR-sykleille. Kolmannessa alaluvussa käsitellään tarkemmin

tässä kandidaatintyössä käytettävää LEGO Power Functions RC-protokollan tukemaa ”Single Output”-signaalimuotoa ja sen signaalin kehystä. Lisäksi kolmannessa alaluvussa lasketaan kandidaatintyössä käytettävät komennot bitteinä.

3.1 LEGO Power Functions RC-protokollan eri signaalimuodot

LEGO Passenger Trainin ohjaus toteutetaan lähettämällä infrapunasignaaleja junan sisällä olevaan RC-vastaanottimeen, joka käyttää LEGOn omaa LEGO Power Functions RC-protokollaa. LEGO Power Functions RC -protokolla pitää LEGOn dokumentaation mukaan sisällään neljä eri signaalimuotoa, joita voidaan käyttää eri tilanteissa olevien langattomien LEGOn radio-ohjattavien lelujen ohjaamiseen. Nämä neljä eri signaalimuotoa ovat: “extended mode”, “combo direct mode”, “single output mode” ja “combo PWM mode”. [1]

Signaalimuodoista ”combo Direct mode” ja ”combo PWM mode” on tarkoitettu kahden eri junan ohjaamiseen samanaikaisesti käyttäen samaa ohjainta. Nämä muodot kumminkin eroavat siten, että ”combo PWM mode” on monimutkaisempi ja sillä on enemmän mahdollisia komentoja LEGO-junalle. Siinä missä ”combo direct mode”lla pystyy lisäämään nopeutta, vähentämään nopeutta tai pysähtymään junan kokonaan niin ”combo PWM mode”:lla pystyy tarkasti yhdellä käskyllä määrittelemään junan nopeuden eteen tai taakse seitsemän vaihteisella nopeusasteikolla. [1]

Vastaavasti signaalimuodoista ”extended mode” ja ”single output mode” on tarkoitettu yhden junan ohjaamiseen. Näistä ”extended mode” ei pidä sisällään yhtä montaa komentoa kuin ”single output mode”. Siinä missä ”extended mode:”lla pystyy lisäämään nopeutta, vähentämään nopeutta ja pysähtymään niin single output modella pystyy seitsemänvaiheisella asteikolla määrittelemään junan nopeuden eteen tai taakse. Lisäksi sillä pystyy antamaan muita monimutkaisempia käskyjä esimerkiksi ”Täyttä vauhtia eteenpäin”, jolloin juna mahdollisesti jarruttaa ja vaihtaa suuntaa, mikäli se on aiemmin mennyt taaksepäin. [1]

3.2 LEGO Power Functions RC-signaalikoodaus

LEGO Power Functions RC -protokolla käyttää signaalin koodaukseen ja lähettämiseen kolmea bittiä, jotka ovat ykkönen, nolla ja ”start/stop”-bitti, jolla ilmoitetaan signaalin alku ja loppu. Signaalikoodauksessa jokainen bitti koostuu kahdesta eri syklistä, joita ovat IR-sykli ja ”pause-sykli”. IR-syklissä infrapunavalo on päällä ja ”pause”-syklissä pois päältä. Jokaisen bitin alussa on sama kuusi kappaletta IR-sykliä mutta ”pause”-syklien määrä

on eri lähetettävän bitin mukaan. Nollabitti koostuu kuudesta IR-syklistä ja kymmenestä ”pause”-syklistä, ykkösbitti koostuu kuudesta IR-syklistä ja 21 ”pause”-syklistä ja vastaavasti ”start/stop”-bitti koostuu kuudesta IR-syklistä ja 39 ”pause”-syklistä. Signaalikoodauksessa syklien taajuus on 38 kHz, jolloin signaalikoodauksessa käytettävän syklin pituus eli jaksonaika saadaan laskettua seuraavalla kaavalla: [1]

$$T = \frac{1}{f} => \frac{1}{38000\ Hz} \approx 26.316\ \mu s.$$

Vastaavasti lisäämällä edellisen kaavan eteen kertoimena X bitin signaalin syklien määrän saadaan seuraava kaava, jolla saadaan laskettua bittisignaalin t_{signal} kesto sekunneissa:

$$t_{signal} = X \cdot \frac{1}{f}.$$

Tällä kaavalla saadaan muodostettua taulukko 1, jossa on eroteltuna jokaisen bitin IR- ja pause-syklien kestot ja kesto kokonaisuudessa:

Taulukko 1. *LEGO Power Functions RC-protokollan bittien rakenne ja niiden kesto.*

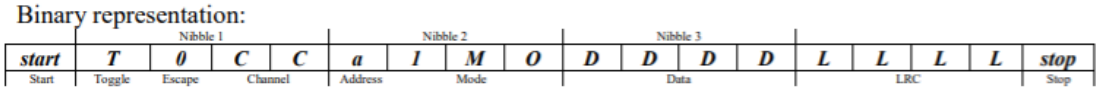
	0-bitti	1-bitti	Start/Stop-bitti
IR-syklien kesto (t)	$6 \cdot \frac{1}{38000Hz} \approx 157.89\ \mu s$	$6 \cdot \frac{1}{38000Hz} \approx 157.89\ \mu s$	$6 \cdot \frac{1}{38000Hz} \approx 157.89\ \mu s$
”Pause”-syklien kesto (t)	$10 \cdot \frac{1}{38000Hz} \approx 263.16\ \mu s$	$21 \cdot \frac{1}{38000Hz} \approx 552.63\ \mu s$	$39 \cdot \frac{1}{38000Hz} \approx 1026.32\ \mu s$
Kesto kokonaisuudessa (t)	$16 \cdot \frac{1}{38000Hz} \approx 421.05\ \mu s$	$27 \cdot \frac{1}{38000Hz} \approx 710.53\ \mu s$	$45 \cdot \frac{1}{38000Hz} \approx 1184.21\ \mu s$

Syklien ja bittien kestojen perusteella voidaan päätellä että IR-signaalin lähettäminen tulee vaatimaan tarkkaa reaaliaikaisuutta joka asettaa haasteita Raspberry Piin sovelluksen toteutukselle. Tämän lisäksi LEGO Power Functions RC-protokolla lähettää signaalia lomitetusti neljää eri kanavaa käyttäen lisätäkseen todennäköisyyttä siitä, että LEGO Passenger Train vastaanottaa signaalin [1]. Tässä kandidaatintyössä kanavakoodausta ei kuitenkaan käytetä, koska se lisäisi sen verran monimutkaisuutta työn toteutukseen.

3.3 Single output mode

Tässä kandidaatintyössä käytetään näistä signaalimuodoista ”single output modea”, eikä tästä syystä muiden signaalien rakennetta sen tarkemmin käydä lävitse. ”Single output mode”, niin kuin muutkin LEGO Power Functions RC -protokollan tarjoamista signaalimuodoista, hyödyntää sekä pulssileveysmodulaatiota eli PWM:mää, että kolmesta eri bitistä koostuvaa signaalinkoodaus tekniikkaa. Tässä kappaleessa käydään erityisesti lävitse LEGOn dokumentaation pohjalta signaalin binäärirakenne ja sen osiot ”single output modessa”. [1]

LEGO Power Functions RC -protokollan dokumentaatiossa on single output moden signaalin bittirakenteesta seuraavan lainen kuva (Kuva 6): Signaalissa ensimmäisenä on start-bitti, joka merkkaa signaalin alun, tämän jälkeen signaali koostuu neljästä eri ”nibblestä” eli neljästä bitin osiosta. Ensimmäisessä bittiosiossa ensin tulee ”toggle”-bitti T, joka LEGOn dokumentaation mukaan ”Vaihtuu uuden komennon merkiksi”. Tätä enempää ”toggle”-bitistä ei kuitenkaan mainita dokumentaatiossa, joten sen merkitys jäi työtä tehdessä hiukan avonaiseksi. ”Escape”-bitti E on single output modessa aina 0, kuten kuvassa (Kuva 6) nähdään ja se onkin dokumentaation mukaan suunniteltu ”combo PWM mode”:n tarpeisiin. Seuraavaksi tulee kaksi kanava (Channel) bittiä C, jotka määrittävät mitä neljästä kanavasta signaali käyttää. [1]



Kuva 6. *IR-signaalin rakenne. [1]*

Toisessa neljän bitin osiossa eli ”nibble 2”:ssa ensimmäinen bitti on osoite (address)-bitti a, jonka oletusbitti on 0 mutta poikkeustilanteissa siinä käytetään bittiä 1. Seuraava bitti on single output modessa oletuksena 1. Seuraava bitti on muoto(mode)-bitti M, joka ”single output mode”:ssa PWM komentoja käytettäessä on 0 ja vastaavasti käytettäessä clear/set/toggle/Inc tai Dec komentoja on 1. Tämän jälkeen tulee output bitti O jonka merkitys jäi itselleni vielä teoria osuudessa epäselväksi. Kolmannessa neljän bitin osuudessa eli ”nibble 3”:ssa sijaitsee neljä ”data”-bittiä D eli varsinainen käsky LEGO Passenger Trainille, jonka määritellään LEGOn dokumentaatiossa seuraavasti. (Kuva 7) [1]

Mode = PWM			
Data	***DDDD***	0000	Float
		0001	PWM forward step 1
		0010	PWM forward step 2
		0011	PWM forward step 3
		0100	PWM forward step 4
		0101	PWM forward step 5
		0110	PWM forward step 6
		0111	PWM forward step 7
		1000	Brake **then float**
		1001	PWM backward step 7
		1010	PWM backward step 6
		1011	PWM backward step 5
		1100	PWM backward step 4
		1101	PWM backward step 3
		1110	PWM backward step 2
		1111	PWM backward step 1

Mode = Clear/Set/Toggle/Inc/Dec			
Data	***DDDD***	0000	**Toggle full forward (Stop → Fw, Fw → Stop, Bw → Fw)**
		0001	**Toggle direction**
		0010	**Increment numerical PWM**
		0011	**Decrement numerical PWM**
		0100	Increment PWM
		0101	Decrement PWM
		0110	Full forward (timeout)
		0111	Full backward (timeout)
		1000	Toggle full forward/backward (default forward)
		1001	**Clear C1 (negative logic – C1 high)**
		1010	**Set C1 (negative logic – C1 low)**
		1011	**Toggle C1**
		1100	**Clear C2 (negative logic – C2 high)**
		1101	**Set C2 (negative logic – C2 low)**
		1110	**Toggle C2**
		1111	**Toggle full backward (Stop → Bw, Bw → Stop, Fwd → Bw)**

Kuva 7. *”Single output mode”:n käskyt ”data”-bitteinä* $\boldsymbol{D}$*. [1]*

Viimeisessä neljän bitin osiossa tulee niin sanotut neljä ”LRC”-bittiä $\boldsymbol{L}$, jotka lasketaan edeltäneiden kolmen edeltävän bittiosion mukaan logiikkalaskuna seuraavasti. [1]

$$LLLL = 0xF\ xor\ nibble\ 1\ xor\ nibble\ 2\ xor\ nibble\ 3$$

Jossa 0xF on heksadesimaaliluku, joka on binäärinä ”1111”, jolloin kaava jalostuu muotoon:

$$LLLL = 1111\ xor\ nibble\ 1\ xor\ nibble\ 2\ xor\ nibble\ 3$$

Viimeisen neljän bitin osion jälkeen tulee vielä viimeinen stop-bitti, joka merkitsee signaalin päättyneeksi. Edellä käydyn pohjalta voidaankin laskea ”Single output mode”:lla seuraavat komennot kanavan C ollessa yksi. (Taulukko 2)

Taulukko 2. *Työssä käytettäviä komentoja listaava taulukko.*

Komennot	**”Toggle”-bitti T = 0**	**”Toggle”-bitti T = 1**
PWM forward Step 7.	s0000010101111101s	s1000010101110101s
PWM forward Step 6.	s0000010101101101s	s1000010101100101s
PWM forward Step 5.	s0000010101011111s	s1000010101010111s
PWM forward Step 4.	s0000010101001110s	s1000010101000110s
PWM forward Step 3.	s0000010100111001s	s1000010100110001s
PWM forward Step 2.	s0000010100101000s	s1000010100100000s
PWM forward Step 1.	s0000010100011011s	s1000010100010011s
Brake then float	s0000010110000010s	s1000010110001010s
PWM backwards step 1	s0000010111110101s	s1000010111111101s
PWM backwards step 2	s0000010111100100s	s1000010111101100s
PWM backwards step 3	s0000010111010111s	s1000010111011111s
PWM backwards step 4	s0000010111000110s	s1000010111001110s
PWM backwards step 5	s0000010110110001s	s1000010110111001s
PWM backwards step 6	s0000010110100001s	s1000010110101000s
PWM backwards step 7	s0000010110010011s	s1000010110011011s

Taulukossa on laskettuna ”toggle”-bitin T molemmille arvoille omat komennot. Komentojen bittijonossa s on start/stop bitti, 1 on 1-bitti ja 0 on 0-bitti. Lisäksi komennot käyttävät ohjaimen kanavaa 1 jolloin kanavabitit $C = 1$. Näitä samoja komentoja käytetäänkin myös myöhemmin työn toteutus osiossa luvussa 4.3.3 (Lopullisen ohjelmakoodin toteuttaminen C++:lla) lopullisen ohjelmiston toteuttamiseen.

4. KÄYTÄNNÖN TOTEUTUS

Tässä luvussa käsitellään työn tapausesimerkin eli LEGO Passenger Trainin infrapunaohjaimen toteutusta käytännössä. Luku tarkentaakin teorian pohjalta miten ohjaimen toteuttaminen onnistui vaihe vaiheelta ja millaiseen lopputulokseen päädyttiin. Erityisesti luvussa tullaan käsittelemään miksi kuhunkin ratkaisuun projektin toteuttamisessa päädyttiin ja mitä seurauksia sillä oli.

Luku on jaettu kolmeen eri alalukuun, joista ensimmäisessä käsitellään mitä järjestelmävaatimuksia LEGO Passenger Trainin ohjaaminen asettaa. Erityisesti keskitytään siihen mitä vaatimuksia LEGO Power Functions RC-protokolla asettaa lähetettävälle signaalille ja miten Beckhoffin ohjausjärjestelmä huomioidaan ohjelmiston rajapinnassa.

Toisessa alaluvussa käydään lävitse järjestelmävaatimusten pohjalta työn suunnitteluprosessia. Luvussa myös kerrotaan, miten järjestelmän reaaliaikavaatimukset huomioitiin suunnitteluprosessin aikana esimerkiksi ohjelmointikieltä valittaessa.

Kolmannessa alaluvussa käsitellään suunnittelun pohjalta, miten ohjaimen toteutus onnistui ja pysyikö se suunnitelmassa vai jouduttiinko suunnitelmasta poikkeamaan. Luvussa erityisesti kerrotaan työn toteutuksen haasteista ja miten ne vaikuttivat työn toteutukseen.

4.1 Järjestelmän vaatimukset

LEGO Passenger Trainiin infrapunaohjaimen järjestelmävaatimuksissa keskeisimpään rooliin nousee reaaliaikaisuus. Kuten luvussa 3.2 (LEGO Power Functions RC -signaalikoodaus) käsiteltiin, niin pienimmät osat LEGO Power Functions RC -signaalissa ovat IR-syklin aikana olevat PWM-syklit, joiden jaksonaika T on noin $26.316\ \mu s$. Koska IR-sykli pitää sisällään niin päällä- kuin pois päältä tilan niin ajallisesti lyhin ledin tilan vaihtelu on pituudeltaan syklin jaksoajasta T puolet eli $13.158\ \mu s$. LEGO Power Functions RC -dokumentaatio ilmoittaa 0-bitin (low bit) vaihteluväliksi $316 - 526\ \mu s$, 1-bitin (High bit) vaihteluväliksi $526 - 947\ \mu s$ ja Start/stop-bitin vaihteluväliksi $947 - 1579\ \mu s$. Näin ollen järjestelmävaatimuksissa voidaan vaatia muutaman mikrosekunnin tarkkuutta, jotta LEGO Power Functions RC -vastaanotin kykenee vastaanottamaan signaaleja luotettavasti. [1]

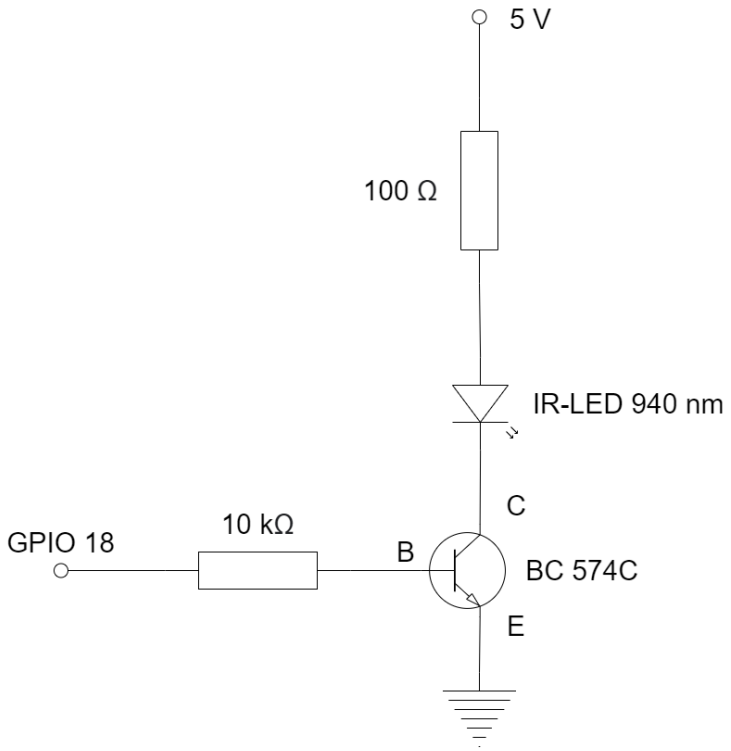
Lisäksi kuten LEGO Power Functions RC -dokumentaatiosta [1, s. 12] näkee niin protokolla käyttää signaalin lähettämisessä neljäkanavaista signaalinlähetystekniikkaa, jolla lisätään todennäköisyyttä, että LEGO Passenger Train vastaanottaa signaalin. Tässä

onnistutaan käytännössä siten että ohjain lähettää neljää eri kanavaa pitkin lomitetusti saman signaalin, jotta LEGO Passenger Trainin infrapunavastaanotin onnistuisi edes yhden niistä vastaanottamaan onnistuneesti. Tämän takia järjestelmävaatimuksissa onkin hyvä huomioida se, kuinka LEGO Passenger Train mahdollisimman todennäköisesti vastaanottaisi signaalin. Tämä on automaatiojärjestelmän toiminnan kannalta oleellista, sillä mahdollisesti myöhemmin tähän kandidaatintyöhön liitettävä junan ohjain ei tiedä onko signaali mennyt perille. Tämä LEGO Passenger Trainin kyky vastaanottaa ohjaimen signaaleja onkin reaaliaikaisuusehdon täyttymisen ohella oleellinen, jotta ohjainjärjestelmä olisi luotettava.

Lisäksi järjestelmävaatimuksena oli se, että Raspberry Piihin pohjautuvan LEGO Passenger Trainin ohjausjärjestelmän saisi hyvin integroitua Beckhoffin automaatiojärjestelmään, joka ohjaisi LEGO Passenger Trainin toimintaa automaatiologiikan pohjalta. Tämän takia ohjelmistorakennetta suunnitellessa tuli huomioida millaisella ohjelmistorajapinnalla Raspberry Piin ohjelmiston saisi teoriassa yhdistettyä Beckhoffin automaatiojärjestelmään.

4.2 Suunnitteluprosessi

Kandidaatintyön IR-lähettimen fyysinen osuus toteutettiin leipälevyyn, jonka päälle koottiin komponenteista infrapunavalo sekä sen säätöjärjestelmä, jota ohjattiin Raspberry Piin ”GPIO”-pinniä käyttämällä. Kandidaatintyön suunnittelussa osien valinnassa pääasiallisena lähteenä käytettiin 2017 vuonna tehtyä vastaavanlaista sovellusta ja sen tekemisen ohjetta komponenttien osalta [15]. Kyseisestä lähteestä löytyykin piirikaavio, jonka pohjalta saatavilla olevista komponenteista valmistin lopullisen työssä käytettävän piirikaavion (Kuva 8).



Kuva 8. *Infrapunalähettimen piirikaavio*

Kandidaatintyössä pyrittiin saamaan IR-LED mahdollisimman kirkkaaksi koska tämä parantaa signaalin kantoetäisyyttä. Tämän takia kuten piirikaaviosta (Kuva 8) näkee, LED:iä ei kytketty suoraan Raspberry Piin ”GPIO”-pinniin vaan ”GPIO”-pinniä käytetään transistorin B-kannassa virtakytkimen roolissa. ”GPIO”-pinnin päästäessä virtaa transistori BC 574C päästää lävitse virtaa Raspberry Piin viiden voltin pinnistä. Tällä pyrittiin siihen, että infrapunaledi saisi enemmän virtaa ja palaisi kirkkaampana, lisäten näin ohjaimen toiminnan kantomatkaa.

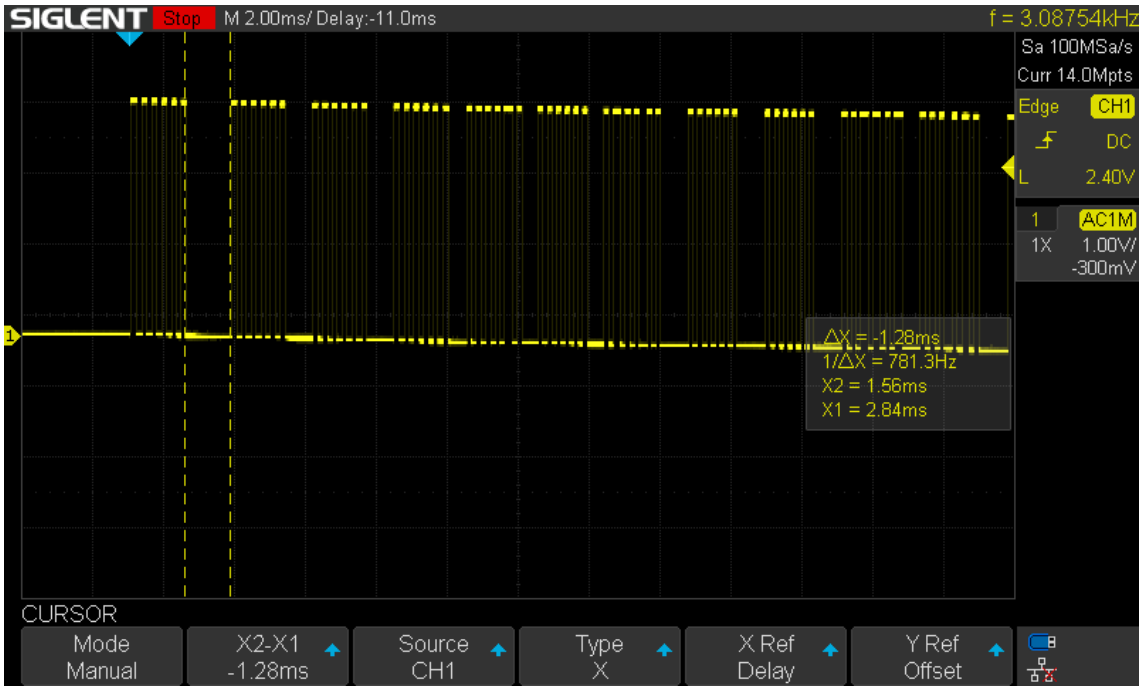
Infrapunalähettimen suunnittelun seuraava vaihe oli pohtia millä ohjelmointikielellä saadaan parhainten toteutettua järjestelmävaatimuksissa mainittu reaaliaikavaatimus. Kuten kappaleessa 2.3 Python teoriaa mainitaan, Pythonin tarjoama ”RPi.GPIO” tarjoaa hyvin tarkkaa PWM-signaalia, jonka virhe on noin 1 μs, mikä on parempi kuin C++ tarjoamassa PWM-generoinnissa [13]. PWM-generoinnin takia kandidaatintyössä päädyttiinkin alustavasti yrittämään ohjelmointia käyttäen Python ohjelmointikieltä. Tästä huolimatta kuten seuraavassa luvussa 4.3 (Ohjaimen toteutus ja haasteet) ilmenee, niin Pythonilla tarpeeksi reaaliaikaisen järjestelmän toteutus Raspberry Pillä osoittautui erittäin haastavaksi. Tämän takia lopulta työn toteutuksessa jouduttiin siirtymään C++ ohjelmointikieleen.

4.3 Ohjaimen toteutus ja haasteet

Tässä luvussa käsitellään ohjaimen toteutusta ja sen haasteita. Erityisesti luvussa keskitytään reaaliaikaisuuden aiheuttamiin haasteisiin ohjelmistototeutuksen näkökulmasta ja miten ohjelmistontoteutus yleisesti aiheutti haasteita Raspberry Piitä käytettäessä. Luku on jaettu kolmeen alalukuun, joista ensimmäisessä käsitellään työn toteutuksen aloittamista Pythonilla ja sitä miksi lopulta Python ohjelmointikielestä jouduttiin vaihtamaan C++ ohjelmointikieleen. Toisessa alaluvussa käsitellään mitä haasteita Geany®-ohjelmointiympäristö aiheutti C++ ohjelmoinnin näkökulmasta ja kuinka Raspberry Piin ”GPIO”-pinnien erilaiset nimeämiskäytännöt aiheuttivat haasteita ohjelmoinnille. Kolmannessa alaluvussa kerrotaan, miten lopullinen ohjelmisto toteutettiin C++-ohjelmakielellä ja miten erilaiset reaaliaikaisuuden aiheuttamat haasteet ratkaistiin käyttäen C++:n tarjoamia kirjastoja ja muita ratkaisuja.

4.3.1 Pythonin reaaliaikaisuuden haasteet ja siirtyminen C++ -ohjelmointikieleen.

Ohjaimen toteutus aloitettiin Python ohjelmointikielellä käyttäen Raspberry Piihin tarjolla olevaa ”RPi.GPIO”-kirjastoa, jonka käyttämiseen tarvittavat komennot on kerrottu aiemmin luvussa 2.3 Python teoriaa. Teoriassa käsiteltyjä komentoja sekä Pythonin unikomentoa $Sleep(time)$, jossa $time$ on aika sekunteina luotiin alustava ohjelmakoodi. Pythonilla luotu ohjelmakoodi vaikutti toimivan ja ledi välähteli rytmeittäin. Ledin syklit ovat kuitenkin niin lyhyitä, että niiden ajallinen tarkkuus tarkastettiin oskilloskoopilla. Oskilloskooppi näytti, että Pythonilla luotu ohjelmointikieli ei ollut kappaleessa 4.1 (Järjestelmävaatimukset) mainitun reaaliaikavaatimuksen mukainen. Kuvasta 9 nähdään, mihin Pythonin ”sleep”-komento kykeni ”start/stop-bitin” ”pause”-syklissä. ”Pause”-syklin pituuden tulisi olla teoriaosuuden kappaleen 3.2 (LEGO Power Functions RC -signaalikoodaus) mukaan $1184.21\ \mu s$, tämä kuitenkin oli $1280\ \mu s$ joten se heitti tarkkuudessa noin $100\ \mu s$.



Kuva 9. *Pythonilla toteutettu ”increment PWM” -komento.*

Koska ”start/stop”-bitin ”pause”-sykli on bittien sykleistä pisin, tulisi tämä epätarkkuus aiheuttamaan lyhemmissä sykleissä entistäkin suurempia ongelmia ajallisen tarkkuuden osalta. Tämä aiheuttaisikin kappaleessa 4.1 (Järjestelmän vaatimukset) käsitellyn bittien raja-arvojen ylityksiä heikentäen ohjaimen luotettavuutta merkittävästi. Työn suorituksen aikana verratessa muihin vaihtoehtoisiin unikomentoihin selvisikin, että ohjelmakoodin epätarkkuuden aiheutti Pythonin ”$Sleep(time)$”-komento, joka ei kykene riittävän tarkkaan reaaliaikaisuuteen. Kyseinen epätarkkuus olikin työn suorituksen kannalta liian suuri ja tulisi vaikuttamaan merkittävästi LEGO Passenger Trainin kykyyn vastaanottaa lähetetty signaali oikein. Tämän takia kandidaatintyön onnistumisen kannalta olikin selvää, että unikomento joudutaan vaihtamaan. Suurimman ongelman kandidaatintyön toteutuksen kannalta aiheutti se, ettei yrityksistä huolimatta Pythoniin ollut saatavilla unikomentoa, joka täyttäisi mikrosekunnin reaaliaikavaatimuksen eikä Pythonin luvussa 2.3 (Python teoriaa) käsitelty ”RPi.GPIO” -kirjastolla pystynyt yksinkertaisesti määrittämään IR-syklien määrää mikä teki toteutuksesta hyvin vaikeaa. Tämän takia kandidaatintyön käytännön osuudessa päädyttiin siirtymään C++-ohjelmointikieleen, joka tarjoaa ajallisesti hyvinkin tarkkoja ohjelmaa tauottavia ”unikomentoja”. Vaikka C++-ohjelmointikielen syntaksi onkin Pythonia monimutkaisempaa, niin se tarjoaa alemman tason ohjelmointikielenä laajempia palveluita, kuin mitä Python kykenee tarjoamaan.

4.3.2 Geanyn ja GPIO-pinnien nimeämisen aiheuttamat haasteet

Raspberry Piin Raspbian käyttöjärjestelmä pitää oletuksena sisällään Geany ohjelmointiympäristön C++-ohjelmointia varten. Geany on hyvin kevyt ja yksinkertainen tekstieditori, jota pystyy käyttämään erilaisten ohjelmistokoodien ohjelmointiin [18]. Geanyn kevyys ja tekstieditorimainen rakenne asettaa kuitenkin haasteita ohjelmoinnin alkuvalmisteluissa, kun haluaa asettaa Geanyn asetukset tietyn ohjelmakielen ohjelmointia varten.

Kandidaatintyön aikana Geany aiheutti ajoittain haasteita ja ihmetystä, kun ohjelmoitu koodi ei toiminut. Erityistä haastetta aiheutti se, miten Geanyn sai ohjaamaan Raspberry Piin ”GPIO”-pinnejä ja kuinka Geanyn saa kääntämään projektin niin, että ohjelma sisältää kaikki tarvittavat kirjastot. Erityistä ongelmaa aiheutti ”WiringPi”-kirjasto, joka ei ole Geanyssa oletuksena ”build”-komennoissa. Lisäksi koska Geany on harvemmin käytetty ohjelmistoympäristö, niin sen käyttöohjeiden löytäminen oli verrattain vaikeaa. Geany:n ”build”-komentoja ohjataan ”build”-valikon alta löytyvästä ”Set Build Commands”-sarakkeesta, josta aukeaa Geanyn ”Set Build Commands”-ikkuna (Kuva 10).

Kuva 10. *Geanyn ”set build commands"-ikkunan asetukset, joilla ”WiringPi” toimii.*

Kuten kuvasta 10 näkyy niin Geanyn ”Set Build Commands”-ikkuna pitää sisällään listan komennoista, joilla Geany kääntää ja rakentaa ohjelman. Kuvassa 10 olevat asetukset ovat muuten Geanyn oletusasetukset mutta osioihin ”Compile” (käännä) ja ”Build” (rakenna) on lisätty molempiin kohta ”-lwiringPi”, joka lisää kääntäjään ja rakentajaan luvussa 2.2 C++ teoriaa käsitellyn ”WiringPi”-kirjaston [4]. Kuitenkin lähteestä [5] poiketen kandidaatintyötä tehdessä Geany ei toiminut jos ”Execute” (suorita) komennon eteen

pisti ohjeissa mainitun ”sudo”-komennon, joka suorittaisi ohjelman ”roottina” eli yleisesti pääkäyttäjänä. [5, s. 42]. Siitä huolimatta ”WiringPi”-kirjaston sai toimimaan Geanylla ilman ”Sudo”-lisäystä niin kuin aiemmin luvussa 2.4 (C++ teoriaa) käydään lävitse.

Tämän lisäksi C++:lla ohjelmoitaessa haasteita aiheutti kuvassa 7 nähtävä Raspberry Piin ”GPIO”-pinnien nimeämistekniikat. Kandidaatintyön IR-ohjainta aloittaessa Python ohjelmointikielellä GPIO-pinnejä ohjattiin käyttäen luvussa 2.3 (Python teoriaa), mainittua ”GPIO Zero”-kirjastoa. Tämä käytti lähtökohtaisesti Raspberry Piin ”GPIO”-pinnien numeroinnissa kuvassa 7 näkyvää ”GPIO.BCM”-nimeämistekniikkaa. Sen sijaan luvussa 2.4 (C++ teoriaa) käsitelty ”WiringPi”-kirjasto käyttää oletusarvona kuvassa 4 näkyvää ”WiringPi”:n nimeämistekniikkaa. Tämän takia tätä kandidaatintyötä työtä tehdessä aiheuttikin haastetta se, millä numerolla sai halutun pinnin lähettämään signaalia halutulla tavalla. Esimerkiksi ”GPIO.BCM”-nimeämistekniikalla ”header” numerossa 12 oleva pinni on numeroltaan 18 kun taas ”WiringPiin” omalla nimeämistekniikalla pinnin numero on 1.

Lisäksi Geanyssa aiheutti haasteita vaikeus tehdä monesta tiedostosta koostuvaa C++ rakennetta, joita C++ luokkarakenteessa yleisesti käytetään. Tästä johtuen kandidaatintyössä päädyttiin luomaan infrapunaohjaimen sisällään pitävä luokka $Controller$ yhteen C++-ohjelmointikielen ”.cpp” tiedostoon. Kun nämä haasteet saatiin ratkaistua, voitiin siirtyä lopullisen ohjaimen tekemiseen C++ ohjelmointikielellä.

4.3.3 Lopullisen ohjelmakoodin toteuttaminen C++:lla

Lopullista infrapunaohjaimen ohjelmakoodia tehtäessä eniten haasteita aiheutti edelleen luvussa 4.1 (Järjestelmän vaatimukset) mainittu reaaliaikaisuusvaatimus, joka vaatii $13.158\ \mu s$ pituisien signaalien lähettämistä mahdollisimman pienellä virheellä, joka ei vaikuta signaalin tulkitsemiseen. Kandidaatintyössä tämä ongelma, joka oli aiheuttanut C++ ohjelmointikieleen siirtymisen, ratkaistiin pääasiassa käyttäen C++:n tarjoamaa ”nanosleep”-kirjastoa [11, luku 8.10].

”Nanosleep”-kirjasto tarjoaa perinteistä unikomentoa ”$sleep(time)$” tarkempaa koodin tauotusta. ”Nanosleeppia” käyttäen voidaankin ohjelmoida ajallisesti reaaliaikaisempaa ohjelmaa kuin perinteistä ”sleep”-komentoa käyttäen. Kuitenkin ”Nanosleepin” tarkkuutta rajoittaa edelleen Linuxin ytimen (kernel) toiminta [11, luku 8.10]

Nanosleep saadaan käyttöön sisällyttämällä nanosleep kirjasto komennolla ”$\#include < time.h >$”. Tämän jälkeen voidaan luoda osoitin struct timespec olioon komennolla ”$struct\ timespec\ tim1, tim2$”, jossa ”$tim1$” on parametri, jolla nanosleepille voidaan asettaa haluttu taukoaika nanosekunteina ja ”$tim2$” on parametri, jolla voidaan asettaa

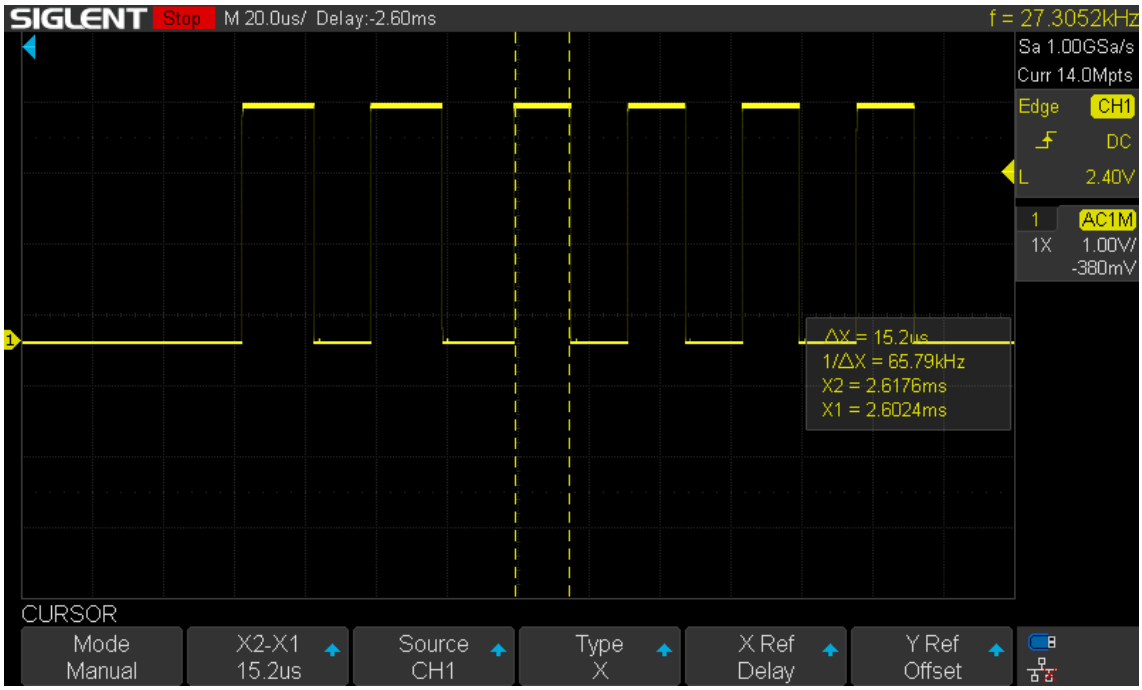
nanosleepille haluttu taukoaika sekunteina. Lopulta ”nanosleepin” taukoaika lasketaankin summaamalla $tim1$ ja $tim2$. ”Nanosleepille” annetaan parametri nanosekunteina komennolla ”$tim.tv.nsec = nanoseconds$”, jossa ”$nanoseconds$” on nanosekuntien määrä. ”Nanosleep” komennolle annetun ajan keston tulee olla alle $10^9\ ns$. Tämän jälkeen ”nanosleep” tauon sai käynnistettyä komennolla ”$nanosleep(\&tim, NULL)$”. [11, luku 8.10]

Vaikka ohjelmakoodi saatiin pääasiassa toteutettua reaaliaikavaatimusten mukaisesti käyttäen ”nanosleeppia” mutta lyhimpiä $13.158\ \mu s$ kestoisia ledin tiloja ei pystynyt toteuttamaan tarkasti edes sitä käyttämällä. Tämä johtui siitä, että vaikka ”nanosleepille” annetaan tauon aika nanosekunneissa, niin käytännössä oskilloskoopilla pystyi toteamaan, että ”nanosleep” komento kykeni tässä kandidaatintyössä käytettävällä Raspberrylla ja käyttöjärjestelmällä suorittamaan minimissä noin. $100\ \mu s$ pituisia signaaleja luotettavasti. Tämän takia täytyikin keksiä luova ratkaisu $13.158\ \mu s$ välisien tilojen vaihteluun IR-syklin aikana. Kandidaatintyössä päädyttiinkin käyttämään ”while”-looppia ja oskilloskoopilla testaamalla määrittelemään kuinka monta kertaa ohjelman täytyi käydä ”while”-looppia lävitse, että siitä saatiin noin $13\ \mu s$ pituinen tauko koodille. Käytännön testaamisen ja oskilloskoopilla mittauksen seurauksena saatiin aikaiseksi kuvan 11 mukainen ohjelmakoodi funktio nimeltä ”$ircycle$”, joka kandidaatintyössä käytettävällä Raspberry Piillä ja käyttöjärjestelmällä tauotti ohjelman noin $13\ \mu s$ ajaksi.

```
//runs for ~13.16 mikroseconds
void ircycle()
{
    int t = 0;
    while (t < 1200)
    {
        ++t;
    }
}
```

Kuva 11. *Ohjelmakoodi, joka tauottaa noin 13 µs ajaksi käyttäen ”while”-looppia.*

Tämä ”while”-loopilla toimiva tauotusmekanismi osoittautuikin hyvin yksinkertaiseksi ja oskilloskoopilla testattaessa verrattavan luotettavaksi tavaksi toteuttaa tasaisia viiveitä. Tämä onkin nähtävissä hyvin kuvasta 12 joka on otettu oskilloskoopilla mitatessa lopullisen infrapunaohjaimen signaalia IR-syklin aikana.



Kuva 12. *IR-syklin tilan vaihteluiden pituus oskilloskoopilla mitattuna.*

Kuten kuvasta 12 voidaan nähdä, niin IR-syklin päälle/pois osien pituus on noin $15.2\ \mu s$. Vaikka tämä ei ole täysin luvussa 4.1 (Järjestelmän vaatimukset) mainitun $13.158\ \mu s$ pituinen, niin LEGO Passenger Trainilla testattaessa järjestelmä osoittautui riittävän tarkaksi ja LEGO Passenger Train vastaanotti signaalit hyvin. Lisäksi oskilloskoopilla voitiin havaita, että funktiolla $ircycle$ tuotetut tauot vaihtelivat yleisesti $13 - 16\ \mu s$ välillä.

”Nanosleeppia” ja funktiota ”$ircycle$” käyttäen voitiinkin yleisesti ratkaista käytännössä reaaliaikaisuuden ohjausjärjestelmälle aiheuttamat haasteet. Niiden pohjalta voitiin luoda ohjelmarakenne, jonka keskiössä on luokka ”$Controller$”. ”$Controllerin$” yksityinen (private) rajapinta koostui kolmesta eri funktiosta, joita oli ”$zero()$”, ”$one()$” ja ”$startorstop()$” joista jokaista kutsumalla saatiin toistettua yksi työssä käytettävä bitti infrapunasignaalina. Esimerkkinä yhdestä bitin luovasta funktiosta on ykkösbitin luova funktio ”$one()$” kuvassa 13. Luokalla pyritäänkin yksinkertaistamaan mahdollista tulevaa ohjelmiston laajentamista sekä varmistamaan rajapinnan yksinkertaisuus, kun mahdollisesti tulevaisuudessa ohjain liitetään Beckhoffin automaatiologiikkaan.

```
struct timespec tim, tim2;
int startstop = 1026316-89014;
int nolla = 263158-64158;
int yksi = 552632-55368;
int ir_time = 13158; //13158

//Initializes 1-bit
void one()
{
    int i = 0;
    while (i < 6)
    {
        digitalWrite(pin, HIGH);
        ircycle();
        digitalWrite(pin, LOW);
        ircycle();
        ++i;
    }
    tim.tv_nsec = (yksi); //21*10^9L/(frequency)
    nanosleep(&tim, NULL);
}
```

Kuva 13. *Ykkösbitin luontiin tarkoitettu funktio ”one( )”.*

Kuten kuvasta 13 nähdään, niin bitin IR-syklin osuudet luodaan while-looppia käyttämällä, jonka jälkeen pause-sykli toteutetaan käyttäen tarkkaa nanosleep komentoa. Vastaavanlaisella tekniikalla toteutettiin myös 0- ja ”start/stop”-bittiä toteuttavat funktiot. Nanosleepin sisällä oleva parametri $yksi$ on numero int, joka pitää sisällään kappaleessa 3.2 (LEGO Power Functions RC -signaalikoodaus) ykkösbitin pituuden nanosekunteina eli ”552632” ja siitä testattaessa vähennetyn arvon jolla ”55368” jolla pyritään pienentämään ”pause”-syklin virhettä, joka aiheutui erinäisistä virheistä ajastuksessa. Tämä virhettä vähentävä arvo määriteltiin käyttämällä oskilloskooppia ja mittaamalla sillä syklien kestoa ja iteroimalla kunnes syklien kestot olivat mahdollisimman lähellä toivottua arvoa.

Tämän lisäksi luokka ”$Controller$” pitää sisällään julkisissa (public) metodeissa metodin ”$initcommand(char\ command[])$” (Kuva 14). Metodille annetaan parametrina $command$ taulukko (array), joka pitää sisällään kolmea eri kirjainta (char). Nämä kirjaimet kuvaavat eri bittejä lähetettävässä komennossa. Näin ollen arvot voivat olla ’s’, ’0’ tai ’1’, joista ’s’ käynnistää ”start/stop”-bitin, ’0’ käynnistää 0-bitin ja ’1’ käynnistää 1-bitin.

```
//Initializes given command 9 times to increase reliability.
void initcommand(char command[])
{

    int length = 19;
    int j = 0;
    times = 0;
    while (times < 8)
    {
        j = 0;
        while (j < length)
        {
            char bit = command[j];
            if (bit == '0')
            {
                zero();
            }
            else if (bit == '1')
            {
                one();
            }
            else if (bit == 's')
            {
                startorstop();
            }
            ++j;
        }
        delay(100);
        ++times;
    }
}
```

***Kuva 14**. Metodin ”initcommand” toteutus.*

Kuten kuvasta 14 voidaan nähdä niin $initcommand$ käy lävitse parametrin $command$ bittejä sisällään pitävän taulukon arvot yksitellen ”while”-looppia käyttämällä ja sen arvojen mukaan käynnistää eri IR-signaalin osia lähettäviä funktioita ”$zero()$”, ”$one()$” tai ”$startorstop()$”. Tämän lisäksi ”initcommand” suorittaa sille parametrina annetun komennon yhdeksän kertaa lisätäkseen todennäköisyyttä siitä, että LEGO Passenger Train vastaanottaa yhden niistä onnistuneesti. Kokeiden pohjalta toistojen välillä oleva 100 millisekunnin viive osoittautui järjestelmätestauksessa parhaaksi junan toiminnan kannalta.

”$Controller$”-luokka pitää myös sisällään 15 eri julkisen rajapinnan metodia, joita kutsumalla ohjelma toteuttaa luvun 3.3 (Single output mode) taulukon 1 mukaisesti ”PWM forward step” 1–7, ”PWM backwars step” 1–7 tai jarruta ja pysähdy (Brake then float) komentoja. Lisäksi metodi pitää huolta luvussa 3.3 (Single output mode) mainitusta ”toggle”-bitistä T, jonka arvon tulee vuorotella, jotta LEGO Passenger Train erottaa uuden komennon vanhasta. Yksi tällainen metodi on kuvassa 15 näkyvä metodi $incseven()$ joka asettaa junan nopeuden eteenpäin nopeudelle seitsemän.

```
//inits PWM forward step 7
void incseven()
{
    if (toggle == 0)
    {
        toggle = 1;
        initcommand(inc70);
    }

    else if (toggle == 1)
    {
        toggle = 0;
        initcommand(inc71);
    }
}
```

Kuva 15. *”PWM forward step 7” käynnistävä ”incseven”-metodi.*

Kuten metodista ”$incseven()$” voidaan huomata, niin se kutsuu ”toggle”-bitin T mukaan metodia $initcommand$ parametrilla ”$inc70$” tai ”$inc71$”, jotka sijaitsevat luokan yksityisessä rajapinnassa. Esimerkiksi parametrin ”$inc70$” rakenne on seuraavanlainen: ”$char\ inc70[19] = "s0000010101111101s"$” kuten luvun 2.3 taulukosta (Taulukko 1) huomataan niin kyseinen parametri ”$inc70$” pitää sisällään ”PWM forward step 7”-komennon ”toggle”-bitin T ollessa nolla.

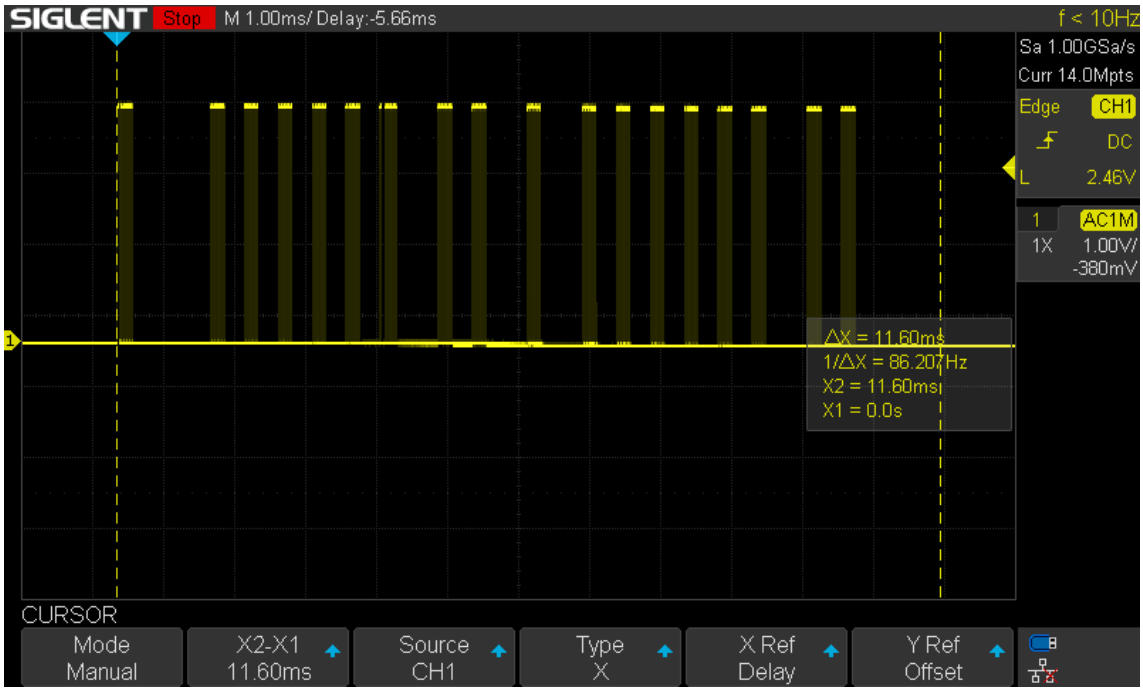
Tämän lisäksi koska tässä kandidaatintyössä ei vielä integroida Raspberry Piillä toimivaa infrapunaohjainta Beckhoffin automaatiojärjestelmään niin ”cpp”-tiedoston ”$Controller$”-luokan ulkopuolelle tehtiin yksinkertainen funktio ”question()”, joka ohjelman käynnistyessä kysyy mitä komentoa halutaan kutsua. Annetun arvon pohjalta $question$ funktio kutsuu ”$Controller$”-luokan julkisia ohjauskomentoja, jotka suorittavat halutun komennon. Tämä on tarkoitettu lähinnä testausmielessä, eikä sitä ole suunniteltu käytettäväksi mahdollisissa myöhemmissä ohjelmistoversioissa.

5. LOPPUTULOS JA JATKOKEHITYS

Kandidaatintyössä onnistuttiin tutkimaan reaaliaikaisen sovelluksen toteuttamista ja saatiin toteutettua reaaliaikainen ohjausjärjestelmä teoriaosuuden pohjalta. Käytännön osuuden lopputuloksessa eniten haasteita aiheuttaa virtapiirin rakenne ja ohjelman reaaliaikaisuuden aiheuttamat haasteet. Ohjelmakoodia testattaessa käytännössä voitiin todeta, että LEGO Passenger Train vastaanotti käskyjä verrattain luotettavasti, mutta siitä huolimatta välillä se ei kyennyt vastaanottamaan signaalia oikein. Kandidaatintyössä opitun perusteella tähän on kaksi merkittävintä selittäjää. Ensinnäkin kandidaatintyössä käytettävän LL-503IRT2E-2AC ledin teho ei ole riittävä. Kyseinen ledi välkkyi puhelimen kameralla katsottaessa huomattavasti himmeämmin kuin LEGOn valmistamassa alkuperäisessä LEGO Passenger Trainin ohjaimessa oleva infrapunaledi. IR-signaali osoittautui niin heikoksi, ettei LEGO Passenger Train kyennyt vastaanottamaan signaalia mistään muusta suunnasta kuin suoraan junan yläpuolelta.

Lisäksi kuten aiemmassa luvussa 4.3 (Ohjaimen toteutus ja haasteet) kuvissa 9 ja 12 nähdään, että ohjain luokan ”$Controller$” lähettämät signaalit kärsivät edelleen pienistä viiveistä ja joissain tilanteissa ne voivat heikentää ohjaimen luotettavuutta. Tämän saisikin varmaan helpointen korjattua iteroimalla oskilloskoopilla virheitä pienentäviä arvoja, joita käsiteltiin luvussa 4.3.3 (Lopullisen ohjelmakoodin toteuttaminen C++:lla) sivulla 26, sekä käyttämällä luvussa 2.4 (C++ teoriaa) käsiteltyä C++:n PWM-kirjastoa IR-syklien toteuttamiseen.

Kandidaatintyön lopputuloksena saatiinkin tutkittua reaaliaikaisen ohjelmiston toteuttamista niin teoriassa kuin käytännön tapausesimerkinkin kautta onnistuneesti. Teoriaosuudessa käsiteltiin mahdollisia keinoja ratkaista reaaliaikaisuuden haasteita sekä Raspberry Pi -piiritietokoneen että mahdollisten ohjelmointikielien näkökulmasta. Käytännön tapausesimerkkinä toteutettiin C++ ohjelmakielellä toimiva ohjain, joka osoittautui toimivan edellä käydyn infrapunaledin sijainnin ennakkoehtojen täyttyessä pääasiassa luotettavasti. Ohjaimen luotettavuutta saisi todennäköisesti lisättyä vaihtamalla ledin tehokkaammaksi. Lopullinen ohjelma toteuttaa myös ajallisesti hyvin tarkkaa signaalia mikä voidaan päätellä oskilloskoopilla otetusta kuvasta (Kuva 16) jossa näkyy lopullisen ohjelman toteuttama ”brake, then float”-komento, joka pysäyttää junan.



Kuva 16. *Lopullisen ohjaimen ”brake, then float”-komento.*

Kuvasta nähdään että ”brake then float”-komennon suorittaminen kesti noin 11.6 millisekuntia. Vastaavasti luvun 2.3 (Single output mode), taulukosta 2 nähdään, että kyseinen ”brake then float”-komento, jossa ”toggle”-bitti T on nolla, koostuu 12 kappaleesta ”0”-bittiä, neljästä kappaleesta ”1”-bittiä ja kahdesta ”s”-bitistä. Näin ollen ”brake then float”-komennon virheettömän pituuden tulisi olla: $12 \cdot 421\ \mu s + 4 \cdot 711\ \mu s + 2 \cdot 1184\ \mu s = 10264\ \mu s$. Verrattaessa tätä oskilloskoopilla mitattuun $11600\ \mu s$ saadaan virheeksi: $11600\ \mu s - 10264\ \mu s = 1336\ \mu s$, eli noin 13 %. Vaikka virhe onkin prosentuaalisesti melko suuri, niin yhtä kirjainta kohden virhe on $\frac{1336\ \mu s}{16} = 83.5\ \mu s$ mikä menee luvussa 4.1 (Järjestelmän vaatimukset) käsiteltyjen virhemarginaalien sisälle jokaisen bitin osalta. Tämä kuitenkin korostaa, että ohjausjärjestelmän reaaliaikaisuutta voidaan edellä mainituilla tavoilla kehittää, jotta se pysyisi paremmin virhemarginaaleissa.

Infrapunaohjaimen integroinnin Beckhoffin automaatiologiikkaan voisi toteuttaa useilla eri tavoilla, mutta yksi mahdollinen keino on liittää Beckhoffin logiikka suoraan Raspberry Piin ”GPIO”-pinneihin asettamalla haluttu ”GPIO”-pinni ”$output$” tilasta ”$input$” tilaan komennolla: ”$pinMode(pin, INPUT)$”. Tällöin kyseinen pinni tarkkailee sisään tulevaa jännitettä ja ohjelman voisi muokata ohjaamaan tässä kandidaatintyössä luotua luokkaa $Controller$ sisääntulevan jännitteen pohjalta. Toisena vaihtoehtona on myös käyttää Beckhoffin ohjelmoitavan logiikan ja Raspberryn välillä jotain julkista rajapintaa esimerkiksi ”OPC®”-palvelinta, jonka kautta Beckhoffin automaatiologiikka voisi ohjata suoraan Raspberrylla olevan ”$Controller$”-luokan julkisia metodeja. Kolmas ja mahdollisesti yksinkertaisin ratkaisu olisi käyttää SSH yhteyttä Beckhoffin automaatiojärjestelmän ja

Raspberry Piin välillä ja käyttää sitä ajaakseen Raspberry Piillä sijaitsevan Controller luokan metodeja Beckhoffilta käsin. Käytännössä tämän voisi toteuttaa esimerkiksi käyttämällä puTTYn osana olevaa Plink ohjelmistoa [20].

6. YHTEENVETO

Tässä kandidaatintyössä tutkittiin, sitä miten toteutetaan reaaliaikaisuutta vaativa sovellus. Käytännön tapausesimerkkinä käytettiin mikrosekunnin reaaliaikaisuutta vaativa LEGO Passenger Trainin infrapunaohjainta, jota ohjataan käyttäen LEGO Power Functions RC -protokollaa. Työssä kerrottiin, mitä palveluita LEGO Power Functions RC -protokolla tarjoaa ja kuinka protokolla lähettää infrapunasignaalissa dataa. Työssä erityisesti keskityttiin protokollan ”single output mode”:en ja siihen miten sillä saadaan ohjattua LEGO Passenger Trainia.

Raspberry Piin ja sen ohjelmiston teorian osalta käsiteltiin rakenne, käyttöjärjestelmä, liittimet sekä mahdollisesti hyödylliset Python ja C++ kirjastot reaaliaikaisen sovelluksen suunnittelussa. Kandidaatintyön teoriassa keskityttiin erityisesti siihen, mitä Raspberry Piin ohjelmointikirjastoja voidaan käyttää reaaliaikaisen IR-ohjaimen kehittämiseen ja mahdolliseen jatkokehitykseen. Raspberry Piin osalta tarkasti käsiteltiin Raspberry Piin ”GPIO-header”, sen tarjoamat palvelut ja se miten Raspbian-käyttöjärjestelmä asennetaan Raspberry Piihin. Raspberry Piin ohjelmoinnin teoriassa kerrottiin reaaliaikaisen Raspberry Pi -järjestelmän ohjelmiston toteutuksessa hyödylliset kirjastot sekä Pythonin että C++ osalta. Teoriassa keskityttiin erityisesti C++:n tarjoamiin kirjastoihin ja ohjelmistoympäristöihin, joita kandidaatintyön toteutuksessa käytettiin ja mitä mahdollisesti voisi käyttää jatkokehityksessä. Kandidaatintyössä käytiin tarkasti läpi mahdolliset eri ohjelmistokomennot ja niiden käyttäminen käyttäen hyödyksi esimerkkejä ja mahdollisten ohjelmointikirjastojen rakennetta.

Työn toteutuksessa kandidaatintyön toteutus käytiin lävitse teorian pohjalta aina järjestelmän vaatimuksista, suunnitteluprosessiin ja siihen mitä haasteita ohjaimen toteutus aiheutti ja mitä työkaluja käyttämällä haluttuun lopputulokseen päädyttiin. Työssä keskityttiin erityisesti siihen, miten reaaliaikaisuuden haasteet ratkaistiin, sekä miten Beckhoffin automaatiojärjestelmän ja Raspberry Piin välinen ohjelmistorajapinta huomioitiin järjestelmässä.

Lopulta käsiteltiin työn lopputulos järjestelmän toteutus ohjelmakoodin osalta, sekä kuinka reaaliaikaisen järjestelmän toteutus onnistui käyttäen perusteluna oskilloskoopilla tehtyjä mittauksia. Lisäksi kerrottiin IR-ledin tehottomuuden aiheuttama ongelma signaalin kantomatkalle ja miten mahdollisesti kandidaatintyötä voidaan jatkokehittää LEGO Passenger Trainin osalta ja kuinka sen voisi mahdollisesti liittää Beckhoffin automaatiojärjestelmään tulevaisuudessa. Lisäksi kerrottiin, että lopullinen LEGO Passenger

Trainin -infrapunaohjain vastasi pääasiassa järjestelmän vaatimuksia. Ohjain saatiin toimimaan ja sen reaaliaikaisuusehto toteutui, lisäksi järjestelmälle asetetut luotettavuusvaatimukset pääosin täyttyivät vaikkakin vain tiettyjen ennakkoehtojen toteutuessa.

LÄHTEET

[1] LEGO, LEGO Power Functions RC, 2010, Saatavissa: https://nishioka.com/train/LEGO%20Power%20Functions%20RC%20v120.pdf (Viitattu: 28.3.2021)

[2] J. Lucas, What is Infrared, LiveSciece, 2019, Saatavissa: https://www.livescience.com/50260-infrared-radiation.html (Viitattu: 28.3.2021)

[3] Raspberry Pi Foundation, Raspberry Pi 4 B, Saatavissa: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/ (Viitattu: 28.3.2021)

[4] Raspberry Pi Foundation, GPIO, Saatavissa: https://www.raspberrypi.org/documentation/usage/gpio/ (Viitattu: 28.3.2021)

[5] B. Ward, How Linux Works: What Every Superuser Should Know, No Starch Press, 2015, Saatavissa: https://learning.oreilly.com/library/view/how-linux-works/9781457185519/

[6] Python.org, What is python?, Saatavissa: https://www.python.org/doc/essays/blurb/ (Viitattu: 28.3.2021)

[7] Raspberry Pi Foundation, Raspberry Pi 4 Tech Specs, Saatavissa: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/ (Viitattu: 28.3.2021)

[8] Raspberry Pi Foundation, Raspberry Pi 4 Official power supply, Saatavissa: https://www.raspberrypi.org/products/type-c-power-supply/ (Viitattu: 28.3.2021)

[9] Raspberry Pi Foundation, Raspberry Pi Imager, Saatavissa: https://www.raspberrypi.org/software/ (Viitattu: 28.3.2021)

[10] Raspbian.org, Raspbian, Saatavissa: https://www.raspbian.org/ (Viitattu: 28.3.2021)

[11] M. Mitchell et al., Advanced Linux Programming, New Riders Publishing, 2001, Saatavissa: https://learning.oreilly.com/library/view/advanced-linux-programming/0735710430/ (Viitattu 10.4.2021)

[12] Pinout, Raspberry Pi GPIO-pins, Saatavilla: https://pinout.xyz/

[13] ElectronicWings, Raspberry Pi PWM Generation using Python and C, Saatavilla: https://www.electronicwings.com/raspberry-pi/raspberry-pi-pwm-generation-using-python-and-c (Viitattu: 28.3.2021)

[14] Raspberry Pi Foundation, GPIO in Python, Saatavissa: https://www.raspberrypi.org/documentation/usage/gpio/python/README.md (Viitattu: 28.3.2021)

[15] Koder, How to control LEGO Power Functions trains with Raspberry Pi, 2017, Saatavissa: https://www.koder.be/2017/12/06/lego-power-functions-trains-raspberry-pi.html (Viitattu: 27.3.2021)

[16] Pinout, Pinout Wiringpi, Saatavissa: https://pinout.xyz/pinout/wiringpi (Viitattu 28.3.2021)

[17] Wiringpi, Wiringpi Blink, Saatavissa: http://wiringpi.com/examples/blink/ (Viitattu 28.3.2021)

[18] Geany, Geany-The Flyweight IDE, Saatavissa: https://www.geany.org/ (Viitattu: 28.3.2021)

[19] Raspberry Pi Foundation, Raspberry Pi Operating Systems, Saatavissa: https://www.raspberrypi.org/software/operating-systems/ (Viitattu: 28.3.2021)

[20] Documentation.Help!, 7.2 Using Plink, Saatavissa: https://documentation.help/PuTTY/plink-usage.html (Viitattu 11.8.2021)