jamk.fi

Moottoroitu time lapse-ohjain

Mikko Leppänen

Opinnäytetyö
Toukokuu 2018
Tekniikan ja liikenteen ala
Insinööri (AMK), Ohjelmistotekniikan koulutusohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Leppänen, Mikko	Opinnäytetyö, AMK	Toukokuu 2018
	Sivumäärä 50	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi

Moottoroitu time lapse-ohjain

Tutkinto-ohjelma

Ohjelmistotekniikan koulutusohjelma

Työn ohjaaja(t)

Hannu Luostarinen, Raija Hämäläinen

Toimeksiantaja(t)

Tiivistelmä

Opinnäytetyön tavoitteena oli tehdä time lapse-ohjain, joka muuttaisi kameran paikkaa ja kulmaa ja käskisi kameraa ottamaan kuvia tasaisin välein. Käyttäjä voisi ottaa yhteyttä laitteeseen älypuhelimella weppikäyttöliittymän kautta. Käyttöliittymästä voisi asettaa time lapse-ajon tarvitsemat arvot ja laittaa ajon päälle. Tämän jälkeen käyttöliittymä näyttäisi ajon edistymistä ja ajon voisi myös pysäyttää kesken stop napista. Työn laajuutta oli rajattu siten, että vaatimuksena oli vain yhdenlainen ajo-ohjelma, sekvenssiajo, jossa kameraa liikutettaisiin haluttu matka, pysäytetään liike, otetaan kuva ja jatketaan liikettä seuraavaan kuvanottokohtaan.

Työn toteutuksen keskipisteenä oli Raspberry Pi, missä tehdyt ohjelmat pyörivät. Pi:llä oli C-kielellä kirjoitettu ajo-ohjelma, joka huolehti kameraa liikuttavien servojen ohjauksesta erillisen PWM-servo-ohjainkortin avulla. C-ohjelma huolehti myös kuvanotosta antamalla käskyjä työhön valitulle GoPro 3-kameralle. Käyttöliittymäpuolen palvelinohjelma oli tehty Node.js ja Express.js teknologiota hyödyntäen. Node-ohjelma keskusteli ajo-ohjelman kanssa välittäen sille käyttöliittymän puolelta halutut ajoarvot ja vastaanottaen tilannetietoa ajossa olevan ohjelman etenemisestä. Käyttöliittymän asiakaspäässä käytettiin Vue.js ohjelmistokehystä, jonka avulla yhden sivun html-näkymää päivitettiin. Vue-osio myös tarkistaa käyttäjän antamat syötteet sekä keskusteli Node-serverin kanssa.

Mekaanisesti laitteen osat pakattiin koteloon, joka asennettiin kiskon päälle. Liike kiskoa pitkin tapahtui sen yläpuolelle viritetyn vetohihnan avulla. Kamera asennettiin kotelon päälle.

Lopputulos täytti vaatimusmäärittelyn suurimmalta osalta. Vain vertikaalinen kameran kulman säätö jäi pois.

Avainsanat (asiasanat)

Time lapse, Raspberry Pi, PWM, Servo, C, JavaScript, Node.js, Express.js, Vue.js

Muut tiedot (salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Leppänen, Mikko	Bachelor's thesis	May 2018
		Language of publication: Finnish
	Number of pages 50	Permission for web publication: x

Title of publication

Motorized time lapse-controller

Degree programme

Software Engineering

Supervisor(s)

Luostarinen Hannu, Hämäläinen Raija

Assigned by

Abstract

The goal of the thesis was to make a time-lapse-controller that would change the position and angle of the camera and make it take pictures at set intervals. The user could connect to the device with a smartphone using a web application. The parameters needed for a time-lapse could be set from the UI followed by program start. After this, the UI would show time-lapse progress and the program could be halted with the stop button. The scope of the thesis was narrowed with the requirement that there would need to be only one kind of time-lapse program, a "move shoot move" functionality where the camera takes one picture, then moves and takes another with a set interval.

The center of the implementation was a Raspberry Pi unit that runs the programs needed. A time-laps-program made with C was responsible for moving the servos via a PWM servo controller. It also took care of taking the pictures by giving commands to the GoPro 3 camera that was used. On the UI side, the server was implemented using Node.js together with Express.js. Node-program would send user-defined parameters to the time-lapse-program and receive progress updates while a program was running. The UI client side was implemented using Vue.js. Its job was to update the single HTML page used, validate user-defined parameters and send and receive messages from Node-server.

Mechanically all components were put in a case that sat on top of the rail. A drive belt fastened to the ends of the rail was used to move the rig along the rail. The camera sat on top of the case.

The result met the set requirement specifications except for one thing; namely only the vertical camera angle adjustment was left out.

Keywords/tags (subjects)

Time lapse, Raspberry Pi, PWM, Servo, C, JavaScript, Node.js, Express.js, Vue.js

Miscellaneous (Confidential information)

Sisältö

Kuviot

1 Johdanto

1.1 Yleiskatsaus

Harrasteprojektit ovat opettavaisia. Aina ei projektin tarvitse edes valmistua ja silti voi oppia jotain hyödyllistä. Itse asiassa epäonnistuneista projekteista yleensä oppii enemmän. Tämä ei kuitenkaan ollut sellainen projekti.

Olen kuvaillut GoPro-kameroilla niin sanotun normaalin kuvauksen lisäksi videota nopealla kuvanottotaajuudella. Videokuvaa esitetään normaalisti 25 ruutua sekunnissa. Kun kuvataan vaikka 100 ruutua sekunnissa, voidaan jälkikäsittelyssä kuvaa hidastaa muuttamalla video 25 ruutua sekunnissa nopeuteen. Näin saadaan aikaan hidastusvideon, esimerkiksi frisbeegolfin heittotekniikan analysointiin. Entäpä jos tehdään päinvastoin, kuvataan vaikkapa ruutu sekunnissa ja katsotaan syntynyt video 25 kuvaa sekunnissa nopeudella. Näin saadaan liikettä hitaisiin tapahtumiin, kuten auringonlaskuun, pilvien tai tähtien liikkeeseen. Hidastamalla kuvaustaajuutta vielä lisää, voidaan katsella vaikka kasvin kasvamista tai kukan avautumista normaalinopeutta sulavammin.

Periaatteessa tämä on helppoa. GoPro-kameroissa on sisäänrakennettu time lapse-ominaisuus, jossa valitaan kuvataajuus, kuvakulma ja resoluutio ja sitten kuvaus käyntiin. GoPro-valmistajan sovelluksilla voidaan PC:ssä kasata ruuduista valmis videoklippi. Aika nopeasti huomaa, että tällä tavalla saa kyllä sinänsä laadukasta time lapse-videota, mutta paikallaan olevan kameran takia otokset ovat helposti tylsän näköisiä. Luontodokumentit ovat opettaneet katsojan liian hyvään kuvaan. Näissä time lapse-videoihin liittyy korkean teknisen laadun lisäksi kamera ajo, jossa kameraa liikutetaan hitaasti kuvaussekvenssin aikana. Heti tulee mielenkiintoisemman näköistä jälkeä. Tästä lähti idea toteuttaa tällainen kameranohjauslaite itselleni.

1.2 Tavoitteet

Toimivan laitteen lisäksi henkilökohtaisena tavoitteena oli oppia sekä sulautettujen järjestelmien tekoa kuin myös weppikehitystä. Vaikka puhtaasti ohjelmiston tekeminen on aina ollut lähellä sydäntä, on jotain erityisen hienoa siinä kokemuksessa, että

saa aikaiseksi fyysisen laitteen, minkä voi nähdä toimivan itse tekemällään koodilla. Suuri osa ohjelmistokehityksestä teollisuuden ulkopuolella on siirtynyt weppikehitykseen. Tästä oli itselläni ennen työn aloitusta hyvin suppea kuva. Nämä olivat isoja tekijöitä työn aiheen ja sen toteutukseen vaadittavien teknologioiden valinnassa.

2 Time lapse-kuvaus

Time lapse-kuvaus (eli intervallikuvaus tai sekvenssikuvaus) on elokuvauksen muoto, jossa kuvia otetaan pienemmällä taajuudella kuin millä se näytetään. Jos esimerkiksi ottaa yhden kuvan sekunnissa ja näyttää kuvista koostetun videon elokuville tyypillisellä 24 ruutua sekunnissa nopeudella, on tuloksena tuotettu video 24 kertaa tavallista nopeampi. Tekniikka sopii hyvin tavallisesti hitaasti tapahtuvien muutosten seuraamiseen. Muun muassa luontodokumenteissa käytetään usein time lapse-kuvausta vaikkapa vuodenaikojen vaihtumisen tai kasvien kasvamisen näyttämiseen nopeutettuna.

Time lapse kuvaus juontaa juurensa jo 1800-luvulle. Ensimmäinen tekniikkaa käyttänyt täyspitkä elokuva oli Georges Meliesin Carrefour De L'Opera (1987). 1909 Jean Comandon ja Pathe Freres tekivät uraa uurtavaa työtä käyttäessään time lapse-kuvausta biologisen ilmiön tutkimiseen. (Chowdhurry Farabee. 18.01.2013.)

Tekniikan popularisoinnista kunnia kuuluu tohtori John Ottille, joka käytti menetelmää kasvien kasvuun vaikuttavien tekijöiden seuraamiseen. Ott, joka oli ammatiltaan pankkiiri, jatkoi erilaisten sekvenssikuvauslaitteiden rakentamista niin pitkälle, että lopulta hänellä oli kokonainen kasvihuone täynnä kasveja, joita kuvattiin jatkuvasti. Hän jopa rakensi itse suunnittelemiaan moottoreita, joilla ohjasi kameroiden liikettä kasvien kasvun mukaan. Ott selvitti, että asiat kuten veden määrä ja kasvihuoneen valojen väri vaikuttivat kasvien liikkeisiin. Tätä tietoa käyttäen hän kuvasi time lapse-tekniikalla koreografioituja animaatioita kasvien ”tanssista”. Ajan dokumentit kuten Disneyn Secrets Of Life (1956) käyttivät Ottin time lapse-kuvaa nostaen tekniikan suosiota. (Chowdhurry Farabee. 18.01.2013.)

3 Vaatimusmäärittely

3.1 Yleistä

Time lapse kuvaus vaatii vakaata kuvaustapaa, sillä aikaa nopeutettaessa kaikki kameran liikkeet korostuvat. Tämän vuoksi ko. tekniikkaa käytettäessä on hyvä laittaa kamera paikalleen jalustalle. Jos elokuvaan on haluttu liikettä ilman, että kuvasta tulee huojuvaa, kannattaa kameran liikkeen ohjaus automatisoida. Tästä syystä time lapse-kuvaukseen on tehty erilaisia kameran kiskoajoja, jollainen myös tämän työn lopputuote on.

Työssä oli tarkoitus luoda kameran ohjauslaite, jossa kamera kulkee kiskoa pitkin kulkevan kelkan päällä, kääntyy ja ottaa kuvia tasaisin väliajoin. Kun työtä rajattiin, päädyttiin toteuttamaan kuvausmenetelmä puhtaasti intervallikuvauksena eli siten, että kamera ei ota liikkuessaan kuvia, vaan liikkuu ensin tarvitun matkan, kääntää kameraa haluttuun uuteen kulmaan ja ottaa vasta sen jälkeen paikoillaan ollessaan kuvan. Tällä tavoin poistetaan myös mahdollinen kiskon ja kelkan välisestä kitkasta johtuva tärinä.

3.2 Servo-ohjaus

Servo-ohjaukselta vaadittiin kolmen servon hallintaa. Ensimmäinen liikuttaa kameraa kiskoa pitkin, toinen ja kolmas muuttavat kameran kulmaa. Servoja ohjaavalle minitietokoneelle annettavat arvot olivat yhden sekvenssin kuvanottoväli(ms), kameran kulman muutos vaaka- ja pystysuunnassa, matka kiskoilla (mm) ja sekvenssien määrä. Näillä ohjeilla servo-ohjaus liikuttaa kameraa, kunnes kaikki sekvenssit on ajettu, käyttäjä pysäyttää ohjelman käyttöliittymästä tai kelkka saavuttaa kiskon pään. Molempiin päihin kiskoa oli asennettu valokennot, joiden katkaisusta seuraa keskeytys servo-ohjelmassa.

3.3 Kameraohjaus

Kuvaamiseen haluttiin käyttää GoPro3-kameraa, koska sellainen oli jo valmiiksi käytössä. Servon-ohjausohjelmalta vaadittiin myös kameran ohjausta. Ohjelman täytyi

pystyä kameran asetusten muuttamiseen ja kuvan ottoon sekä kameran paluuviestien vastaanottoon GoPro:n oman rajapinnan kautta.

3.4 Käyttöliittymä

Käyttäjän piti pystyä syöttämään servo-ohjauksen vaatimat arvot käyttöliittymään, minkä jälkeen käyttöliittymän tuli validoida saadut arvot, ja jos ne olivat hyväksyttyjä, lähettää ne servo-ohjaukselle. Tämän jälkeen käyttöliittymästä piti pystyä seuraamaan, missä vaiheessa ohjelmaa servo-ohjaus on menossa, sekä myös tarvittaessa keskeyttämään ohjelman ajo.

4 Käytetty laitteisto ja teknologiat

4.1 Mekaniikka

4.1.1 Lineaarijohde

Lineaarijohde on kisko, jota pitkin kamerakelkka liikkuu. Käyttämällä lineaarijohdetta saadaan helposti tehtyä kamera-ajosta tasaista. Lineaarijohteena projektissa käytettiin metrin pituista Igus Drylin WS-10-40-johdetta. Johteen päällä kulkee 100mm pituinen WW-10-40-10-kelkka. Osat valittiin, koska ne ovat riittävän tukevia, vaikka kamerana käytettäisi GoPro:n sijasta järjestelmäkameraa. Valintaan vaikutti myös se, että kyseistä kiskoa käytetään myös vastaavissa kaupallisissa tuotteissa.

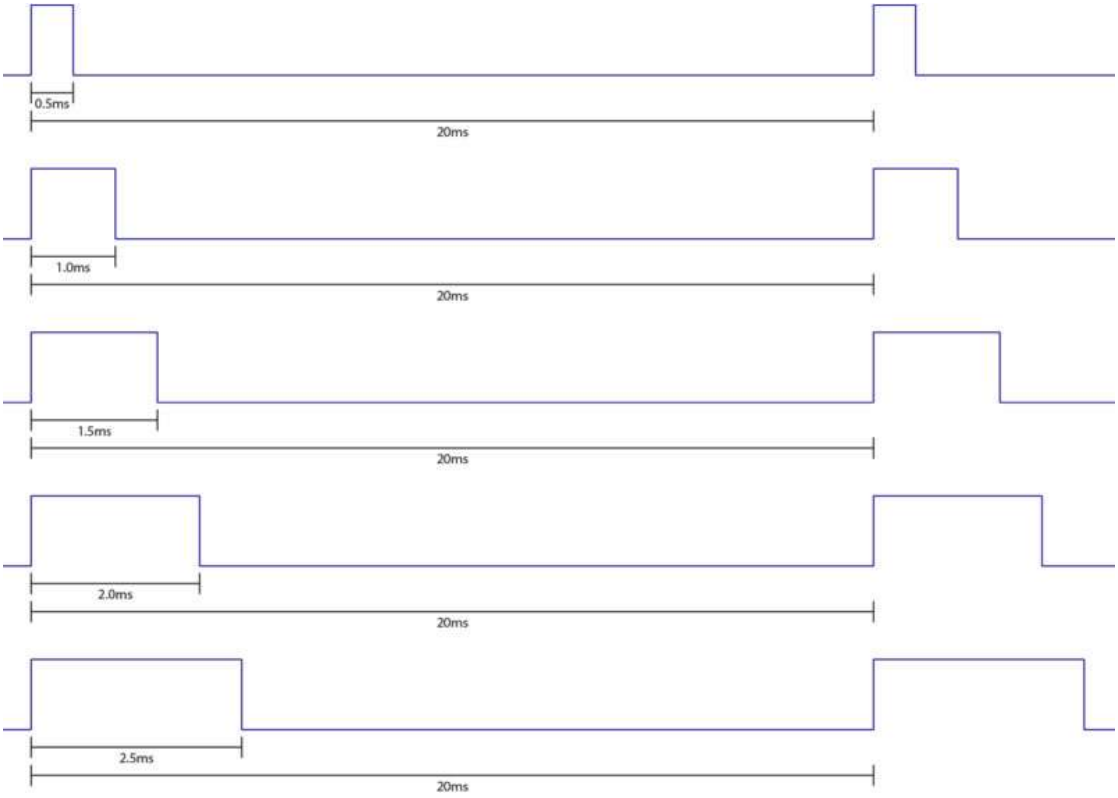
4.1.2 Servomoottorit

Servomoottori on tarkkaan asemoinnin ohjaukseen tarkoitettu moottori. Tavallinen servo sisältää vaihteiston moottorin ja ulostulevan akselin välillä, jonka liikekulmaa on rajoitettu. Servomoottorin sisäinen elektroniikka selvittää moottorin tämänhetkisen kulman, ja moottori muuttaa kulmaa sisään tuleva ohjaussignaalin mukaan. Suurin osa servoista vastaanottaa PWM- (Pulse width-modulation), eli pulssinleveysmodulaatiosignaalia servon ohjaukseen.

Pulssinleveysmodulaatiossa ohjauspulssin pulssisuhdetta muutetaan eli säädetään sitä, kuinka suuren osan pulssin kestosta lähtö on aktiivinen. Tyypillisesti servoon

ohjataan aallonpituudeltaan 20 ms signaalia (kuvio 1). Servon signaalin ollessa aktiivinen 1,5 ms ajan servo on keskiasennossa ja 1 ms arvolla servo on myötäpäivään ääriasennossa ja 2 ms arvolla vastapäivään ääriasennossa. (What is a Continuous Servo? N.d.)

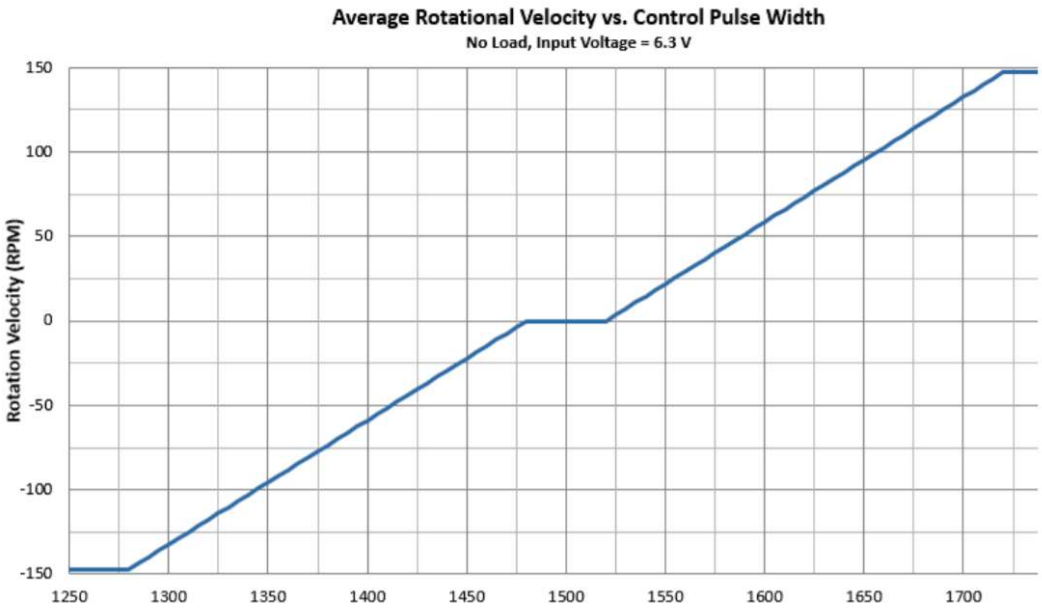
Ympäripyörivän servon kohdalla pulssin pituus määrittelee moottorin asennon sijasta sen nopeutta ja suuntaa. Tällöin 1 ms aktiivinen signaali pyörittää servoa maksiminopeudella myötäpäivään, 1,5 ms signaalilla servo on paikallaan ja 2 ms signaali pyörittää servoa täyttä vauhtia vastapäivään. (What is a Continuous Servo? N.d.)



Kuvio 1. Pulssinleveysmodulaatiosignaali. (What is a Continuous Servo? N.d.)

Kiskoa liikuttavaksi servomoottoriksi valittiin edullinen amerikkalainen servo Parallax 900-00360. Servomoottori on ympäripyörivää mallia, mutta koska siitä löytyy takaisinkytkentäsignaali, joka palauttaa servonohjaukselle servon senhetkisen kulman, voi sitä myös halutessaan käyttää tavallisena servona. Tässä työssä servoa kuitenkin käytettiin vain ympäripyörivänä, eikä paluusignaalia täten tarvittu. Servon nopeus muuttuu signaalin pulssinpituuteen nähden lineaarisesti, mikä helpottaa nopeuden laskentaa (kuvio 2). Lisäksi kyseisessä servossa on kuollut alue keskellä pulssinpituuskaaviota, millä tarkoitetaan siis sitä, että servo on pysähdyksissä pulssinpituuden säätöalu-

een keskellä +-20 mikrosekunnin alueella. (Parallax Feedback 360° High-Speed Servo (#900-00360) N.d.)



Kuvio 2. Parallax 900-00360 servon nopeuden suhde pulssinleveyteen. (Parallax Feedback 360° High-Speed Servo (#900-00360) N.d.)

Servon datalehden mukaan, servoa ohjataan 50 Hz PWM-signaalilla, ja servon pitäisi pyöriä täyttä vauhtia myötäpäivään pulssinpituudella 1,28 ms ja vastapäivään pulssinpituudella 1,72 ms. Kuollut alue, jolla servo on pysähdyksissä, pitäisi olla 1,48-1,52 ms. Testattaessa kuitenkin vaikutti siltä, että säädön keskikohta olisi ollut 1,58 ms. Kokeellisesti todettua säädön keskikohtaa käytettäessä servon nopeus muuttui molempiin suuntiin mentäessä samalla tavalla. (Parallax Feedback 360° High-Speed Servo (#900-00360) N.d.)

Kameran kääntämiseen käytettiin Futaba S3003 servoa. Alkuperäinen suunnitelma oli käyttää Adafruit pan-tilt kittiä kameran kääntämiseen vaaka- ja pystysuunnassa kahdella SG 92 mikroservolla, mutta se osoittautui liian kevyeksi ratkaisuksi GoPro-kameralle. Kyseinen laite lieneekin suunniteltu Raspberryn vain muutaman gramman painoisten kameramoduulien liikutteluun. Samalla luovuttiin kameran kääntämisestä pystysuunnassa. Näin saavutettiin myös mekaanisesti tukevampi ratkaisu. Futaba S3003 servo kiinnitettiin kotelon kanteen ja GoPro-kiinnike ruuvattiin kiinni suoraan servon akselin kiinnitettyyn kääntöpyörään.

4.2 Elektroniikka

4.2.1 Raspberry Pi Zero v1.3

Projektin ohjaimena käytettiin Raspberry Pi Zero-pienoistietokonetta. Tietokonetta valmistaa Raspberry Pi-säätiö, Cambridgelainen hyväntekeväisyysjärjestö, jonka tavoitteena on opettaa ohjelmointia ja muita digitaalisen maailman taitoja ihmisille, erityisesti nuorille, ympäri maailmaa. Raspberry Pi-koneet ovat tehokkaita ja erittäin edullisia, mikä on tehnyt laitteista hyvin suosittuja. Alkuperäinen Raspberry Pi 1 Model B julkistettiin vuonna 2012. Noin 35 dollarin hinnalla sai 85.60 mm × 56.5 mm kokoisen tietokoneen, jossa oli ARM11 prosessori, 256 MB SDRAM-muistia, kaksi USB liitäntää, HDMI, 10/100 Ethernet jne. Lisäksi kortilta löytyi 8 GPIO linjaa, jotka voitiin konfiguroida tuloiksi tai lähdöiksi, UART, I2C-väylä, SPI väylä, +3,3 V, +5 V ja maa (Conor Lyons. 2015.). Erilaisia Raspberry Pi-malleja on myyty tähän mennessä yhteensä 19 miljoonaa kappaletta kouluihin, koteihin, konttoreihin ja tehtaisiin (Eben Upton. 2018). Opiskelun lisäksi kyseisiä minitietokoneita käytetään nykyään myös kaupallisissa tuotteissa kuten näytöissä (NEC Display Solutions. 2016). (Raspberry Pi Foundation. N.d.)

Tässä työssä käytetty Raspberry Pi Zero versio julkaistiin vuonna 2015. Kooltaan vain 65 mm kertaa 30 mm ja 5 dollarin hinnaltaan se on ryhmän edullisin ja pienin tuote. Kuitenkin kortilta löytyy 1 GHz CPU, 512 MB RAM, mini HDMI, mikro USB portti, mikro USB virransyöttö ja muiden korttimallien kanssa yhteensopiva 40 nastainen I/O liityntärima.

Raspberry Pi-minitietokoneiden hinta ja levinneisyys tekivät siitä hyvän valinnan projektiin. Koneen yleisyys ja sen käyttö opetuksessa tarkoittivat sitä, että alustan käyttöön oli todella helppo löytää ohjeita ja valmiita kirjastoja, kuten esimerkiksi työssä servon ja valokennojen ohjaukseen käytetty kirjasto. Myös se, että Raspberry Pi tukee laajaa kirjoa ohjelmointikieliä, oli tärkeää ottaen huomioon, kuinka monta eri teknologiaa työssä jouduttiin käyttämään. Koneen käyttöjärjestelmänä toimi Raspbian, joka pohjautuu Debian Linuxiin.

4.2.2 Servo-ohjainkortti

Vaikka Raspberry Piistä löytyy yksi PWM-lähtö, haluttiin silti käyttää erillistä PWM-ohjainta. Koska Piillä on käytettävissä I2C-väylä kommunikointia varten, valittiin servoja ohjaamaan Adafruitin 16-kanavainen PWM-ohjain, josta myös löytyy I2C-väylä. I2C on Philips Semiconductor-yhtiön kehittämä synkroninen pakettikytkentäinen sarjaväylä. Väylä on omiaan yksinkertaisiin tiedonsiirtotarpeisiin, joissa ei tarvita suurta nopeutta. Työssä käytettiin wiringPi-kirjaston rutiineja I2C-kommunikointiin.

PWM-kortilla on myös erillinen 5 V jännitteensyöttö servomoottoreita varten, eli sitä voidaan ohjata 3,3 V mikrokontrollerilla ja silti ajaa servoja 5 V jännitteellä.

Varsinainen ohjauspiiri on PCA9685, joka toimii itsenäisesti omalla kellollaan. Tällä saadaan aikaan ohjelman kiertoajasta riippumaton tarkka PWM-säätö. Kortin kello voidaan ohjelmoida toimimaan 24-1526 Hz taajuudella. PWM-lähdöillä on 12-bittinen resoluutio (4096 askelta), joten työssä käytetyllä 50 Hz taajuudella pulssin leveyttä voidaan säätää n. 5 mikrosekunnin tarkkuudella.

Kortilla on myös 6 osoitepinniä, mikä mahdollistaa useiden korttien kytkemisen samaan I2C-väylään. Tässä työssä käytettiin vain yhtä korttia, joten osoitepinnit jätettiin oletusasentoon. (PCA9685 Product Datasheet N.d.)

4.2.3 GoPro-kamera

GoPro-kamerat ovat "action"-käyttöön suunniteltuja, eli ne ovat sääolosuhteita ja kolhuja kestäviä laitteita. Kameroissa on vedenkestävä kotelo, joka mahdollistaa kameran upottamisen jopa kymmenien metrien syvyyteen ilman ongelmia. Uusimmat mallit ovat vedenpitäviä 10 m syvyyteen ilman erillistä koteloa. Kameroihin on saatavissa loputon määrä kiinnitystarvikkeita ja muita lisävarusteita, joilla katetaan suuri määrä kuvaustarpeita. Kameroissa on kiinteä linssi, eli linssin polttoväliä ei voi muuttaa, mutta parametreilla on mahdollista muuttaa kuvakulmaa eli sitä, miten laajalta alueelta kuvailmaisimesta lopullinen kuva muodostetaan. Diagonaalinen kuvakulma vaihtelee mallista riippuen n. 75-150 asteen välillä, eli melko laajakulmainen linssi on käytössä.

GoPro-perheen kameroilla pystyy kuvaamaan yksittäiskuvia, kuvasarjoja ja videota useilla eri kuvaustaajuuksilla ja resoluutioilla. Lisäksi niissä on sisäänrakennettu time lapse-kuvausmahdollisuus, jossa parametreilla valitaan kuvataajuus, kuvakulma ja resoluutio. Sitten sekvenssi käynnistetään ja kamera ottaa kuvia halutulla taajuudella, kunnes muistikortti tulee täyteen tai akku loppuu.

GoPro 3-versiosta lähtien kameroissa on ollut kaukokäyttö wlan-verkon yli älypuhelimeen saatavalla applikaatiolla. Järjestelmä toimii niin, että kamera muodostaa wlan verkon ja puhelimen appi kytkeytyy tähän verkkoon ja kykenee lähettämään komentoja kameralle ja saa paluuarvoina tietoja kameran toiminnasta. GoPro ei ole julkaissut ohjauskoodeja, mutta monet käyttäjät ovat tehneet tutkimustyötä ja internetistä löytyykin useita lähteitä, joista koodit ja niiden käyttö käyvät selville. Tämän työn toteutuksessa käytettiin Konrad Iturben wlan komentolistoja GiHubista (Konrad Iturbe 2017.). Tässä työssä käytettiin juuri GoPro 3 Black -mallia, koska sellainen löytyi ja vanhastaan ja ominaisuudet sopivat projektin tarpeisiin. (GoPro Hero3 Manual. N.d.)

4.2.4 Virtalähde

Virtalähteenä käytettiin USB lähtöistä varavirtalähdettä. Nämä ovat nykyään yleisiä laitteita ja helposti hankittavissa, koska niitä myydään lähinnä älypuhelinten lataamiseen. Raspberry + wlan-kortti + servo-ohjain-paketin yhteinen virrankulutus on n. 150-170 mA, joten elektroniikan osuus energiankulutuksessa ei ole ongelma. Kelkkaa kuljettava servo vie tyhjäkäynnillä n. 50 mA ja kuormitettaessa n. 100 mA. Koska servoa ajetaan vain lyhyitä nykäisyjä, sen energiankulutus jää jopa alle elektroniikan kulutuksesta. Käytössä oli n. 10000 mAh varavirtalähde, joten sen pitäisi hyvin riittää kuvaussessioihin. Todella hitaita time lapse-kuvauksia tehdessä saattaa olla tarpeen syöttää myös kameralle sähköä varavirtalähteestä, koska kameran ollessa päällä yhtäjaksoisesti akku kestää vähän yli tunnin.

4.3 Ohjelmisto

4.3.1 Git

Projektissa käytettiin Git-versionhallintaa. Git on ilmainen avoimen lähdekoodin versionhallinta, joka toimii hyvin niin pienissä, kuin erittäin suurissakin projekteissa.

Muista versionhallintajärjestelmistä Git eroaa edukseen erityisesti haarautumismallillaan (branching). Git antaa kehittäjän tehdä useita paikallisia ohjelmahaaroja, mitä voi muokata toisistaan riippumatta. Uusista ominaisuuksista tehdään usein omia haaroja. Tällöin saadaan yksittäisen ominaisuuden kehitys täysin erilliseksi kaikesta muusta, jolloin se ei sotke muuta tehtyä työtä. Jos tällöin ominaisuus todetaan myöhemmin turhaksi, voi haaran poistaa todella helposti. Valintaan työn versionhallinnaksi vaikutti myös se, että Git on nopea ja laajasti käytetty, joten omakohtaista kokemusta järjestelmästä löytyi usean projektin verran. (Git N.d.)

Git-palvelimena käytössä oli Bitbucket ja käyttöliittymänä Sourcetree, mitkä kummatkin ovat Atlassianin tuotteita. Bitbucket valittiin siksi, että se tarjoaa ilmaiseksi rajattoman määrän yksityisiä säilöjä (Atlassian N.d.).

4.3.2 VueJS

Vue on käyttöliittymän (frontend) tekoon tarkoitettu Javascript ohjelmistokehys (framework). Vue on suunniteltu otettavaksi käyttöön osissa, eli työhön ei tarvittu mukaan mitään ylimääräistä. Vuen ydinkirjasto keskittyy ainoastaan näkymäkerrokseen eli sen käyttöönotto oli helppoa ja nopeaa. Vuesta on kuitenkin myös isompien projektien kehitykseen laajemmalla kirjastojen ja työkalujen käytöllä. Ohjelmistokehys oli projektiin ihanteellinen, sillä työhön tehty yhden sivun sovellus vaati suhteellisen yksinkertaisia toimintoja, mitkä Vuella oli helppo toteuttaa. (Vue.js N.d.)

4.3.3 NodeJS ja ExpressJS

Node on asynkroninen, tapahtumapohjainen Javascript-runtime. Node on siis serveripuolen (backend) Javascript-kirjasto. Noden vahvuus onkin se, että sekä frontend-, että backend-puolella käytetään Javascriptiä. Tämä vähentää työmäärää, kun serveriä varten ei tarvitse opetella kokonaan uutta ohjelmointikieltä. Tapahtumapohjaisuus näkyy siinä, että alun skriptin suorituksen jälkeen Node jää yksinkertaisesti odottamaan tapahtumia (event). Kun tällainen eventti tapahtuu, suoritetaan vain sen sisältämä koodi. (About Node N.d.)

Express on Nodea varten suunniteltu javascript weppiohjelmistokehys. Siitä löytyy kaikki tarvittavat ominaisuudet kevyen weppiaplikaation tekoon. Käytännössä Expressin avulla työssä toteutettiin se, että käyttäjän weppiselaimen tekemiä http

pyyntöjä vastaanotettiin tietyssä portissa (listen) ja se että pyynnöt ohjattiin oikealle sivulle (routing). (Express N.d.)

4.3.4 Socket-yhteydet

”Soketti (Socket) on eräänlainen verkkokommunikaation päätepiste. Soketti on abstrakti apuväline verkkokommunikaatioon, jonka toiminta ja käyttötapa riippuu käytettävästä ympäristöstä (API, Application Programming Interface). Soketin avulla voidaan lukea ja kirjoittaa tiettyyn määriteltyyn verkkoyhteyteen. Tämän verkkoyhteyden kuvaa ja yksilöi käytettävä IP-osoite ja portti. Soketti kommunikaatio vaatii aina vastinparin, joka käyttää samaa porttia ja sijaitsee valitun ip-osoitteen spesifioimassa kohteessa.” (Marko Hassinen N.d.) Socket-yhteyksiä työssä käytettiin servokontrolleriohjelman ja Node-serverin, sekä Vue-käyttöliittymäohjelman ja Node-serverin välillä.

5 Kaupalliset vaihtoehdot toteutukselle

5.1 Yleistä

Tässä luvussa esitellään kaksi kaupallista toteutusta time lapse-kuvaukseen. Toteutukset edustavat mekaanisesti kahta hyvin erilaista ratkaisua. Pohjalla olevista ohjaustekniikoista tai laitteiden käyttämistä moottoreista ei ole juurikaan tietoa saatavilla, joten arviot on tehty ulkoisesti.

5.2 Edelkrone SliderPLUS

SliderPLUS on kameran jalkaan kiinnitettävä liukukisko. Kiskon saa ostaa sellaisenaan, tai siihen voi lisätä erilaisia osia. Tuote on rakennettu niin, että kummatkin kamera ja itse kisko liukuvat kameran jalan päällä. Tämä mahdollistaa pitemmän liikkeen lyhyemmällä kiskolla ja tekee tuotteesta helpommin mukana kuljetettavan. Kiskoa on saatavilla kahdessa eri pituudessa ja painossa. (Edelkrone SliderPLUS N.d.)

Kameran automaattista liikuttamista varten on Motion Kit, kolmesta moduulista koostuva tuotesarja. Slide-moduuli vastaa kameran ohjauksesta kiskolla ja sisältää myös kaikkien 3 moduulin tarvitseman akun. Slide-moduuli on sijoitettu kiskon pää-

hän. Head-moduuli hoitaa kameran käännöt pysty- ja vaakasuunnassa (pan ja tilt). Siihen on myös tehty kohteen seuranta. Tällöin kameralla seurataan moduulille annettua kohdetta tasaisen kameran käännön sijasta. Viimeisenä on Focus-moduuli joka tarkentaa kameran linssin automaattisesti seurattavaan kohteeseen. (Edelkrone SliderPLUS N.d.)

Kamera-ajon asetuksia hallitaan Bluetooth-yhteyden kautta toimivalla älypuhelinohjelmalla. Kamera-ajo voi tehdä myös käsin. Tällöin kamera asetetaan manuaalisesti kiskolla haluttuihin pisteisiin. Motion Kit muistaa pisteet ja tekee niiden välille tasaisen ajon. (Edelkrone SliderPLUS N.d.)

5.3 Syrp Genie

Syrp Genie on kehitetty hieman eri näkökulmasta, kuin edellinen tuote. Geniessa on yhteen koteloon rakennettu kaikki perus time lapse-kuvaukseen tarvittava laitteisto. Kotelon päälle asennetaan kamera ja sen pohjassa on moottori, joka kiinnitystavasta riippuen joko kääntää kamerakulmaa tai ajaa laitetta kiskoa pitkin vetonarun avulla. (Syrp Genie N.d.)

Monissa muissa kaupallisissa järjestelmissä, kuten yllä kuvatussa SliderPLUS:ssa moottori on paikallaan johteen päässä ja vain osa laitteesta liikkuu. Koska Geniessa koko järjestelmä liikkuu kameran mukana, se ei tarvitse mitään tiettyä kiskoa toimiakseen. Tällöin kisko voidaan vaihtaa vaikkapa vaijeriin. Kun liike ei ole rajoitettu mihinkään tiettyyn laitteistoon voi tuotteella tehdä aivan sen mittaisia ajoja kuin haluaa johdetta vaihtamalla. (Syrp Genie N.d.)

Pelkkä Genie ei voi yhtä aikaa tehdä kameran ajoa kiskolla ja sen kulman kääntöä. Tuotteeseen saa hankittua lisäosina kameran kääntömoduulin joko pelkästään vaakakäännöllä (pan), tai vaaka- ja pystykäännöllä (pan ja tilt). (Syrp Genie N.d.)

Tuotteen hallinta tapahtuu Geniessä olevan LCD-näytön ja painikkeiden avulla. Ajo-ohjelmia on saatavilla erilaisilla valmisasetuksilla tai sitten parametrit voi asettaa itse. Laitteeseen voi tallentaa omia ajo-ohjelmia myöhempää käyttöä varten. (Syrp Genie N.d.) (Edelkrone SliderPLUS N.d.)

5.4 Vertailua omaan toteutukseen

Työssä tehty laite muistuttaa enemmän Genieta toteutuksessaan. Genie on kompaktimman kokoinen ja siitä löytyy enemmän ominaisuuksia, kuten tuki sadoille eri kameroille. Geniesta ei kuitenkaan löydy laitteen etäohjausta puhelimella. Tämä taas löytyy SliderPLUS:sta. SliderPLUS:ssa rajoittavana tekijänä on kuitenkin se, että kamera-ajojen tekeminen on rajoitettu yhteen valmiiseen kiskoratkaisuun. Molemmille laitteille tulee hintaa kameran kääntömoduulien kanssa yli 1000 dollaria, joka myös vaikutti oman ratkaisun hakemiseen. (Syrp Genie N.d.)

6 Työn toteutus

6.1 Yleistä

Työstä tuli hyvin moniosainen. Työn keskipisteenä oli Raspberry Pi, joka pyöritti servonohjausohjelmaa ja weppikäyttöliittymää. Raspberry Pi:ssä oli suoraan johdoilla kiinni PWM-servokontrolleri ja valokennot. Lisäksi Pi oli GoPro:n luomassa langattomassa verkossa. Liittymällä haluamallaan laitteella, esimerkiksi älypuhelimella samaiseen verkkoon sai käyttöliittymän itselleen näkyville laitteen selaimella.

6.2 Raspberry Pi

6.2.1 Käyttöjärjestelmän asennus ja konfigurointi

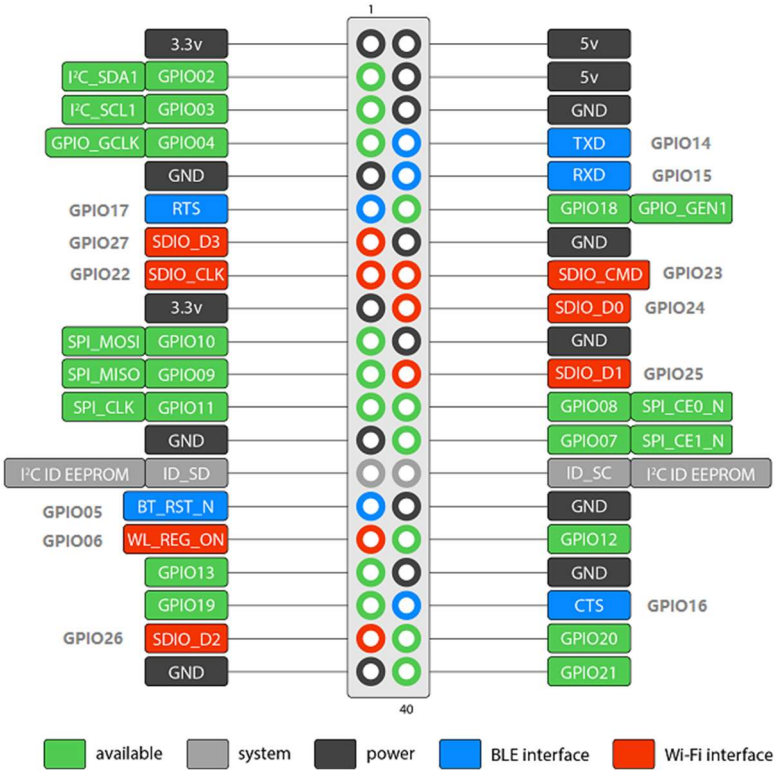
Raspberryn asennus alkoi käyttöjärjestelmän lataamisella. Käyttöjärjestelmäksi valikoitui Rasbian, joka on yleisin Pi korteille asennettava järjestelmä. Rasbianista löytyy suuri osa tarvittavista ohjelmista valmiina ja loput voi ladata kätevästi pakettihallinnan kautta. Valmis levykuva ladattiin suoraan Raspberry Pi säätiön sivuilta. Win32 Disk Imager-ohjelmalla levykuva siirrettiin mikro SD-kortille ja näin saatiin suoraan boottaava järjestelmä. Seuraavaksi aseteltiin Rasbianin asetukset sopiviksi. Rasbiconfig-ohjelmalla saatiin iso osa asetuksista tehtyä ilman asetustiedostojen käsittelyä.

Tämän jälkeen piti tehdä Redbearin IoT pHAT-kortin asetukset, jotta langaton verkko olisi käytettävissä. Koska wlan on käytössä kameran ohjauksessa ja ohjelmointiympä-

ristön pitää päästä kommunikoimaan Raspberryn kanssa, alustettiin USB liitäntä toimimaan paikallisverkon tavoin. Ensin kytkettiin normaali USB-kaapeli PC:n ja Pi:n väliin. Sitten varmistettiin, että Piillä on käytössä dwc2 USB ajuri. Nyt wlan voitiin asettaa kytkeytymään GoPro-kameran wlan verkkoon ja silti kommunikoida ohjelmankehityslaitteena toimivasta PC:stä niin kuin Pi olisi kytkettynä lankaverkkoon.

6.2.2 Redbear IoT pHAT

Raspberry Pi Zero-kortista puuttuvaa verkkoliitäntää paikkaamaan löytyi Redbearin IoT pHAT-lisäkortti, joka mahdollistaa langattoman verkon ja tarvittaessa Bluetooth yhteyden käyttämisen. IoT pHAT liitetään suoraan I/O rimaan, mutta käyttää vain osan nastoista jättäen loput vapaaksi muuhun käyttöön. Kortin käyttämät pinnit ovat kuviossa 3 punaisella.



Kuvio 3. Raspberry Pi ja Redbear IoT pHAT-korttien väliset kytkennät.

Työn alussa oletettiin, että myös käyttöliittymän palvelin voisi käyttää GoPro-kameran langatonta verkkoa. Työn edetessä kävi kuitenkin ilmi, että verkko on tarkoitettu vain viestinvälitykseen GoPron kanssa. Tämä aiheutti ongelman, koska nyt Raspberryn pitikin käyttää kahta verkkoyhteyttä. Yhtä GoPron kanssa keskusteluun ja toista verkkoa käyttöliittymää varten. Nyt myös Raspberryn piti olla tukiasema mah-

dollistaakseen yhteydenotot älypuhelimista tai vastaavista laitteista. Onneksi tämä ominaisuus löytyi verkkokortilta jo valmiiksi. Toinen verkko saatiin aikaan luomalla Raspberryn käynnistyksen yhteydessä virtuaalinen tukiasema.

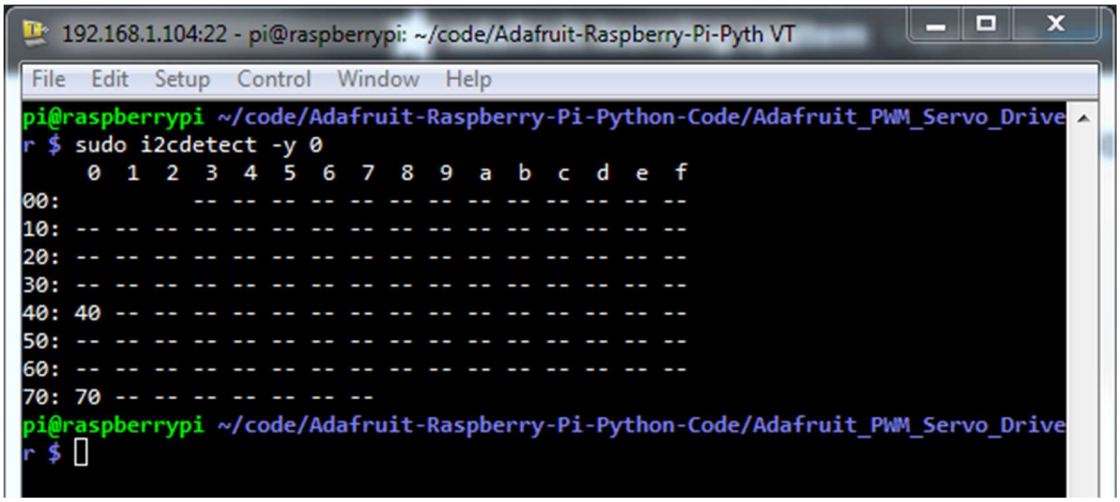
6.2.3 WiringPi-kirjasto

WiringPi-kirjasto otettiin alun perin käyttöön, koska sen avulla saatiin toteutettua näppärästi PWM-kortin rekisteriin kirjoitukset I2C-väylän kautta. Kirjastoa käytettiin lopulta hieman laajemminkin. Siitä löytyi muun muassa helppo tapa tehdä keskeytyksiä. Myös millisekunnin tarkkuudella toimivat millis- (aikavälin laskeminen millisekunnin tarkkuudella) ja delay-aliohjelmat (ohjelman suorituksen pysäytys annetun millisekuntiarvon ajaksi) olivat paljon käytössä.

6.2.4 I2C-asetukset

Raspberry Pi:n ja servokortin välinen liikenne tapahtui I2C-väylän läpi. Jotta I2C-väylä saatiin käyttöön, ladattiin Raspberryyn pari aiheeseen liittyvää apuohjelmaa, python-smbus-kirjasto ja python kieliset i2c-tools ohjelmat.

Kun PWM-kortti kytkettiin (3,3V, GND, I2C-SDA1 ja I2C-SCL1) ja Raspberry oli käynnissä, voitiin komennolla ” sudo i2cdetect -y 1” tarkastaa löytyikö kortti I2C-väylästä (kuvio 4). Kun kortin osoitepinnit olivat oletusasennossa, eli mitään niistä ei ole juotettu umpeen, kortti löytyi oletusosoitteesta 0x40.



Kuvio 4. Löydetyt I2C-väylän laitteet.

6.2.5 Valokennot

Työssä käytetyt kaksi valokennoparia olivat Adafruitin IR Break Beam sensoreita. Valokennot olivat yksinkertaisin ja nopein tapa havaita milloin ollaan kiskon laidassa. Kun kiskon pää saavutettiin, katkaisi laitaan asennettu haitta valokennon signaalin. Tämä aiheutti ohjelmassa keskeytyksen, joka pysäytti servon. Valokenno voi käyttää 3,3- tai 5 voltin jännitettä. Vastaanotin on open collector-lähtö, eli se tarvitsee ylösvetovastuksen, jotta signaalin voi lukea. RaspberryPi:ltä vastus löytyy valmiiksi, sen käytöstä tarvitsee vain kertoa Pi:n alustusten yhteydessä. Kuviossa 5 kerrottiin signaalin ylösvedosta tietyille GPIO pinneille, joita valokennot käyttävät.

```
pullUpDnControl(SERVO_PHOTOCELL_LEFT, PUD_UP);
pullUpDnControl(SERVO_PHOTOCELL_RIGHT, PUD_UP);
```

Kuvio 5. Valokennojen ylösvetovastuksen alustus.

Ohjelmassa keskeytykset alustettiin wiringPi-kirjaston avulla kuvion 6 mukaisesti. Aliohjelmalle annettiin muuttujina valokennon pinni RaspberryPi:llä, tehdäänkö keskeytys nousevasta vai laskevasta laidasta (silloin kun signaali menee poikki vai vasta kun valokenno saa uudestaan yhteyden) ja funktio jonka keskeytys suorittaa. Valokennojen arvoja luettiin RaspberryPi:ltä kuvion 3 alalaidassa olevista GPIO pinneistä 20 ja 21.

```
wiringPiISR(SERVO_PHOTOCELL_LEFT, INT_EDGE_FALLING, &endOfRailReached);
wiringPiISR(SERVO_PHOTOCELL_RIGHT, INT_EDGE_FALLING, &endOfRailReached);
```

Kuvio 6. Valokennojen keskeytykset.

Koska servoja ajavan säikeen haluttiin pysäytystilanteessa ajavan itsensä hallitusti alas, ei keskeytysfunktio tee muuta kuin muuttaa globaalia ajomuuttujaa, jota säie seuraa (kuvio 7).

```
//Ajetaan kun tulee keskeytys jommasta kummasta laidasta.
void *endOfRailReached(void)
{
  printf("laita saavutettu\n");

  timeLapseRunning = false;
}
```

Kuvio 7. Keskeytysfunktio.

6.3 Adafruit PWM-ohjain

Kun Raspberry Pi kytkettiin PWM-ohjaimeen, jossa servot olivat kiinni, voitiin servoja liikuttaa kirjoittamalla Pi:ltä PWM-ohjaimen muistiin halutut arvot I2C-liikennöinnin avulla. I2C tukee yksi- ja kaksisuuntaista 8-bittistä tiedonsiirtoa. Kuviossa 8 esimerkki PWM-kortille kirjoittamisesta. ServoContId oli yhteyden alustuksessa luotu, PWM-kortin osoite ja MODE1 oli vakio, jonka arvo oli 0x00.

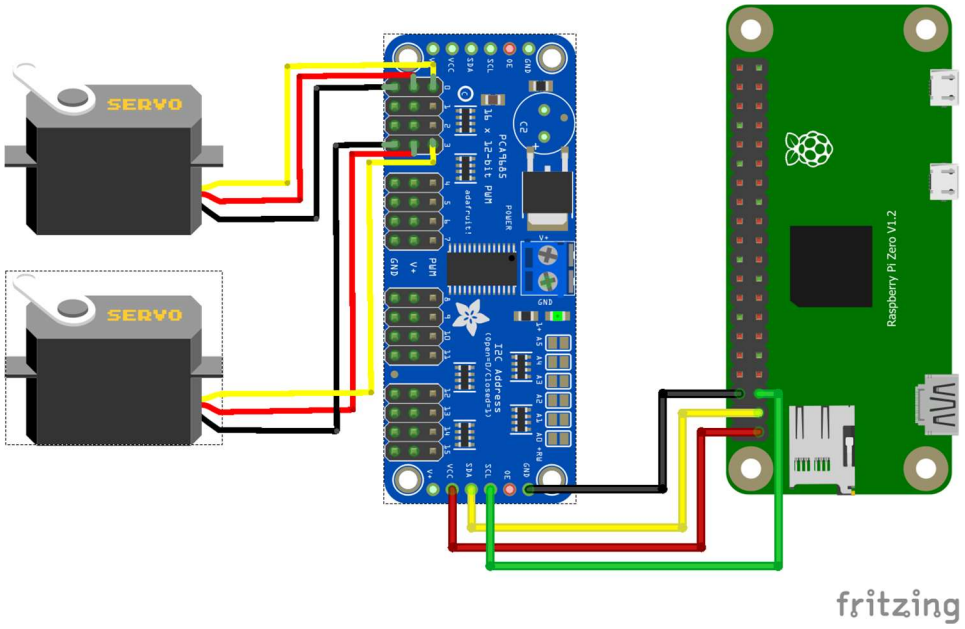
```
// Alustetaan servo-ohjainkortti, ensin kortille reset
wiringPiI2CWriteReg8(servoContId, MODE1, 0x0);
```

Kuvio 8. PWM-kortin reset.

6.3.1 Kytkentä

PWM-ohjainkortti kytkettiin servojen ja Raspberry Pi:n väliin kuvion 9 osoittamalla tavalla. Punainen piuha oli 3,3 voltin ohjausjännite ja musta piuha oli maa. Raspberry Pi:ltä tulevat kaksi muuta piuhaa olivat I2C-liikennöintiin vaadittavat data- ja kellolinjat (järjestyksessä keltainen ja vihreä). (I2C-bus specification and user manual. N.d.)

Servoja varten kortille tuotiin erikseen 5 voltin jännite (kortti tukee maksimissaan 6 volttia). Punainen ja musta piuha vastasivat 5 volttia ja maata. Keltainen piuha kuljetti servon ohjaussignaalia, PWM-kortin tuottamaa kanttiaaltoa, jonka mukaan servot liikkuvat.



Kuvio 9. PWM-kortin kytkennät.

6.3.2 Servot

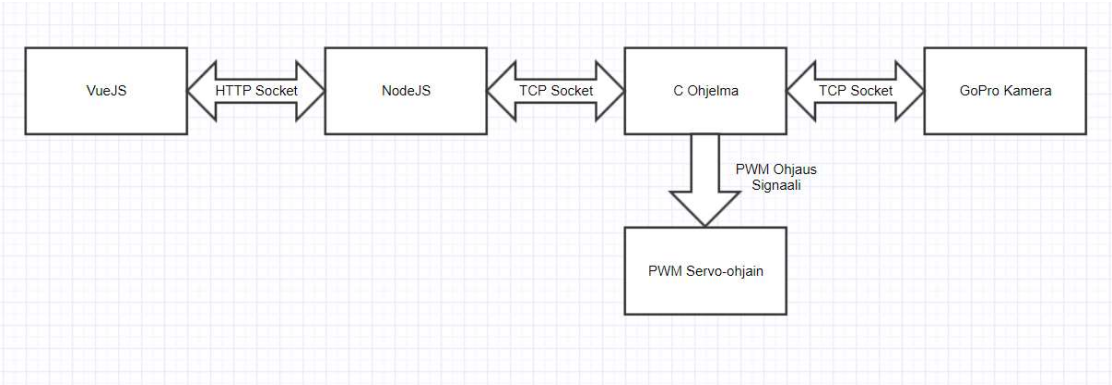
Servoja käskytettiin signaalilla, jonka aallonpituus oli 20 ms. Kanttiaallon nouseva reuna, eli kohta missä signaali vaihtuu nollasta ykköseksi, oli aina 0 ms kohdalla. Servosta riippuen sen nopeutta tai kulmaa vaihdettiin muuttamalla signaalin laskevan reunan paikkaa 1-2 ms välillä.

Normaalin servon tapauksessa signaalin laskiessa 1,5 ms kohdalla servo osoitti eteenpäin, 1 ms kohdalla 90 astetta keskilinjasta oikeaan ja 2 ms 90 astetta vasempaan. Ympäripyörivä servo taas oli 1,5 ms kohdalla paikallaan. 1- ja 2 ms taas pyörittivät servoa maksiminopeudella myötä ja vastapäivään.

6.4 Ohjelmointi

6.4.1 Yleistä

Projektissa luodusta järjestelmästä tuli hyvin moniosainen. Käyttöliittymän päässä weppisivun front-end toteutettiin Vue JS- ja back-end Node JS-kirjastolla. Projektin keskipisteenä oli C-kielellä tehty ohjelma, jonka tarkoitus oli hallinnoida servoja ja GoPro-kameraa sekä keskustella käyttöliittymärajapinnan kanssa. Kuviossa 10 on esitetty yhteydet, joiden avulla projektin eri osat kommunikoivat keskenään. Kommunikointi tapahtuu suurimmaksi osin socket-yhteyksien yli.

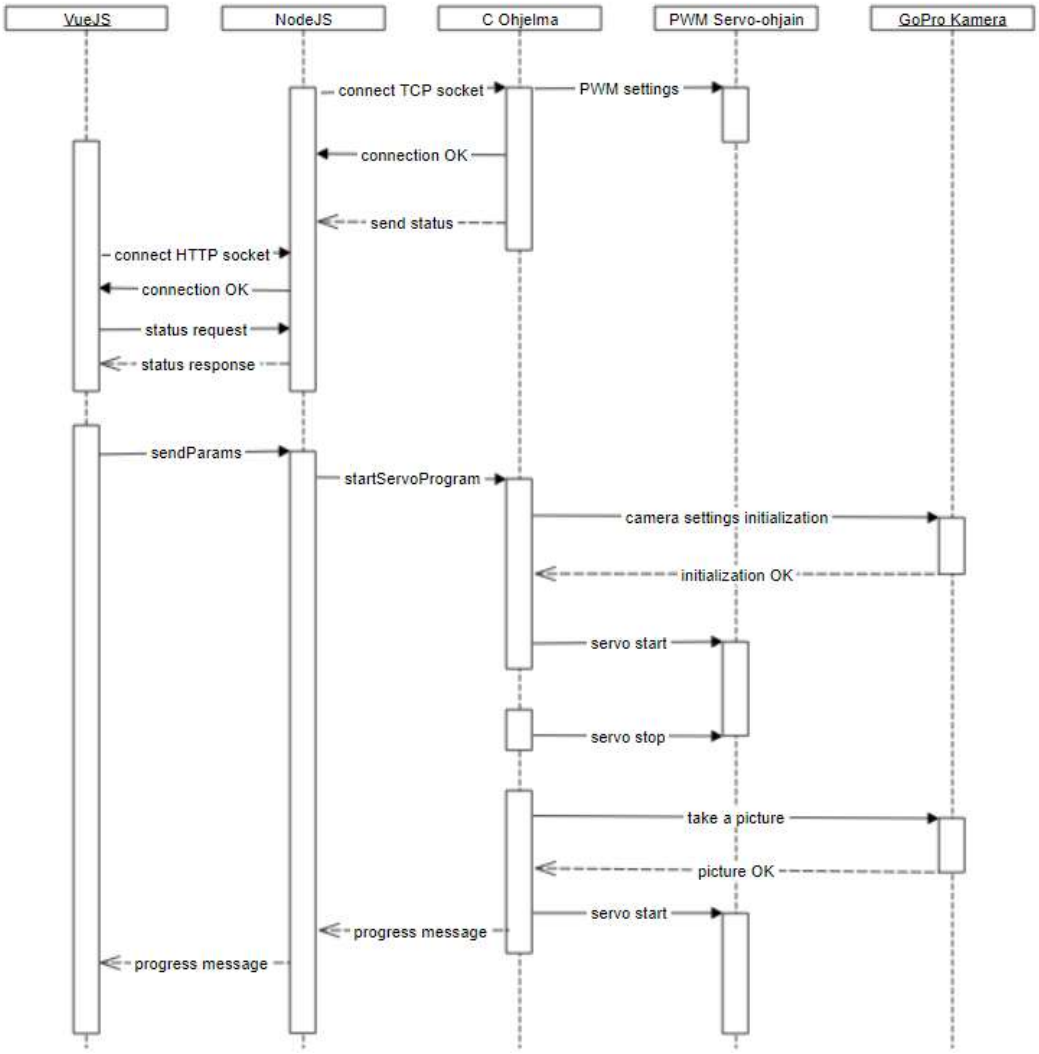


Kuvio 10. Työn eri moduulien suhde toisiinsa.

Vue JS:n tehtävänä oli tarkistaa missä tilassa ohjelma on ja näyttää sen mukaan toisen kahdesta vaihtoehtoisesta sisällöstä sivulla. Silloin kun time lapse-ohjelma oli pysähdyksissä, näytettiin lomake, jolle käyttäjä pystyi syöttämään uuden sekvenssiajon vaatimat arvot ja jos ajo oli jo käynnissä, näytettiin ohjelman edistymispalkki. Lomakenäkymässä otettiin vastaan käyttäjän antamat syötteet, validoitiin ne ja lähettiin sitten NodeJS-serverille. Edistymis-näkymässä taas javascript odotti serveriltä tulevaa tilannetietoa ohjelman edistymisestä ja päivitti sen näkymään näytöllä.

NodeJS serverin tehtävä oli yksinkertaisuudessaan toimia välikätenä käyttöliittymän ja servonohjausohjelman välissä, sekä palauttaa html sivu weppiselaimen sitä pyytäessä.

Projektin ytimenä toimiva C-ohjelma odottaa ensin yhteydenottoa weppiserveriltä, palauttaa oman tilansa ja jää sitten odottamaan seuraavaa käskyä. Saadessaan käskyn ja intervalliajoon tarvittavat parametrit käynnistää ohjelma toisen säikeen, jossa itse ajon suoritus tapahtuu. Samalla pääohjelma jatkaa viestien vastaanottamista ja tarvittaessa keskeyttää ajo-ohjelman. Ajo-ohjelma liikuttaa servoja haluttuun uuteen asentoon, ottaa yhteyden kameraan, sekä lähettää sille kuvanottamispyynnön ja ilmoittaa ajetusta sekvenssistä weppiohjelmaan, jossa edistymistä seurataan. Tätä silmukkaa ohjelma suorittaa siihen asti, kunnes haluttu määrä intervalleja on ajettu, tullaan kiskon päähän, tai käyttäjä lopettaa ohjelman weppikäyttöliittymästä. Sen jälkeen ohjelma jää jälleen odottamaan uutta käskyä. Kuviossa 11 on esitetty ohjelman ja intervalliajon käynnistys sekvenssikaaviona.



Kuvio 11. Sekvenssikaavio ohjelman käynnistämisestä ja time lapse-ajon aloittamisesta.

6.4.2 Ohjelmointiympäristön käyttö

Visual Studio Code:n asennuksen jälkeen editoriin täytyi asentaa kaksi lisäosaa, C/C++ c-ohjelman lukemisen selkeyttämiseksi ja SFTP ohjelman siirtämiseen Raspberry Pi-kortille. SFTP:n käyttöönotto vaati ensin ssh-avaimen luonnin. GitBash:lla luotiin ssh-avain, jota käytettiin yhteydenottoon Raspberry Pi:n ja editorin välillä. Koodia testattiin gdp:llä(GNU Project Debugger).

6.4.3 C-ohjelman rakenne

Ohjelmiston ydin ja sen suurin osa oli C:llä kirjoitettu yksi noin 700 riviä pitkä tiedosto, joka vastasi kaikesta servojen ja kameran hallintaan liittyvästä. Ensimmäisenä ohjelman main-funktiossa olivat alustukset. wiringPiSetupGpio-funktio on työssä käy-

tetyn wiringPi-kirjaston vaatima alustus funktio. Alustusfunktion Gpio-versio takaa sen, että Raspberry Pi:ssä kiinni olevia komponentteja kutsuttaessa voitiin koodissa käyttää Broadcom GPIO pinni numeroita, jotka löytyvät kuviosta 3. (Gordon Henderson. N.d.)

Seuraavaksi alustettiin käytössä oleva mutex. Sen käyttö todettiin kuitenkin loppujen lopuksi kohtuullisen turhaksi, sillä ohjelmassa ei moniajosta huolimatta tule kilpailutilanteita (kuvio 12).

```
if (pthread_mutex_init(&lock, NULL) != 0)
{
  printf("\n mutex init failed\n");
  return 1;
}
```

Kuvio 12. Mutex alustus.

Kuviossa 13 on esitetty servokontrollerin alustus. Käskyn jälkeen servoContId muuttuja sisälsi PWM-kontrollerin osoitteen, jota myöhemmin käytettiin kaikkialla ohjelmassa.

```
// Alustetaan I2C liikennöinti
if ((servoContId = wiringPiI2CSetup(SERVO_CONT_I2C_ADDR)) == -1)
{
  fprintf(stderr, "servoContId: Unable to initialise I2C: %s\n", strerror(errno));
  return 1;
}
```

Kuvio 13. I2C-yhteyden alustus.

Seuraavaksi ohjainkortti resetoitiin ja laskettiin sisäisen kellon taajuus, joka tämän jälkeen kirjoitettiin rekisteriin (kuvio 14).

```
// Alustetaan servo-ohjainkortti, ensin kortille reset
wiringPiI2CWriteReg8(servoContId, MODE1, 0x0);
// Seuraavaksi lasketaan sisäisen kellon skaalaus. Tässä 20ms pwm ohjeukselle sopivat arvot
freq = UPDATE_RATE;
prescaleval = (OSCILLATOR_CLOCK / RESOLUTION / freq) - 1;
prescale = floor(prescaleval + 0.5);
```

Kuvio 14. PWM-reset ja taajuuslaskenta.

Kuviosta 15 löytyy servokontrollerin alustuksessa käytetyt vakiot. 50 Hz taajuus tuottaa 20 ms aallonpituuden. 12 bitin resoluutio tarkoittaa, että kortille annetut ohjausarvot olivat välillä 0-4096.

```
// Adafruit 16-channel PWM Driver
#define OSCILLATOR_CLOCK 25000000 //kortin oma kellotaajuus 25MHz
#define RESOLUTION 4096           //12-bittiä
#define UPDATE_RATE 50            //PWM taajuus
```

Kuvio 15. PWM-kortin vakiot.

Kuviossa 16 on esitetty socket-serveri yhteyden luonti. Socket-käskyllä luotiin socket sockfd muuttujaan, joka toimii IP osoitteistuksella (AF_INET), ja on tyyppiä SOCK_STREAM.

```
//Socket serverin luonti.
//listen komento blokkaa kunnes client ottaa yhteyden.
int initSocketServer()
{
  int sockfd, newsockfd, portno, clilen;
  struct sockaddr_in serv_addr, cli_addr;

  sockfd = socket(AF_INET, SOCK_STREAM, 0);
  if (sockfd < 0)
    error("ERROR opening socket");
```

Kuvio 16. Socket-käsky.

Setsockopt ja SO_REUSEPORT vaadittiin ohjelmalta, jotta virheen tapahtuessa samaa porttia voidaan käyttää yhteyden uudelleen luontiin (kuvio 17).

```
  //Antaa serverin käyttää osoitetta vaikka se olisi jo käytössä.
  if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &(int){1}, sizeof(int)) < 0)
    error("setsockopt(SO_REUSEADDR) failed");

#ifdef SO_REUSEPORT
  if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEPORT, &(int){1}, sizeof(int)) < 0)
    perror("setsockopt(SO_REUSEPORT) failed");
#endif
```

Kuvio 17. Setsockopt ja SO_REUSEPORT.

Serv_addr structiin sijoitettiin yhteyden vaatimat tiedot, osoitteistus tyyppi, mitä yhteyksiä sallittiin (tässä tapauksessa kaikki IP-yhteydet), sekä kuunneltavan portin numero (kuvio 18).

```
bzero((char *)&serv_addr, sizeof(serv_addr));
portno = 2050;
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(portno);
```

Kuvio 18. Serv_addr struct.

Bind käskyllä tiedot sidottiin sockfd muuttujaan. Listen aloittaa socket-yhteyden kuuntelun. Käskylle annettu numero tarkoittaa, että jonossa odottamassa yhteyttä voi olla maksimissaan 5 clientia. Luvulla ei sinällään ollut työssä sen suurempaa merkitystä, sillä vain yksi client, NodeJS-moduuli, otti ohjelmaan yhteyttä ja jonosta hyväksyttiin kerralla muutenkin vain yksi yhteys (kuvio 19).

```
if (bind(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
  error("ERROR on binding");

listen(sockfd, 5);
```

Kuvio 19. Bind ja listen.

Accept komento odottaa yhteydenottoa ja hyväksyy sen. Komento siis estää ohjelman etenemisen, kunnes porttiin otetaan yhteyttä. Yhteyden hyväksymisen jälkeen funktio palautti socket-muuttujan, jota voitiin nyt käyttää viestien lukemiseen ja kirjoittamiseen käyttöliittymän kanssa (kuvio 20).

```
clilen = sizeof(cli_addr);
newsockfd = accept(sockfd, (struct sockaddr *)&cli_addr, &clilen);
if (newsockfd < 0)
  error("ERROR on accept");
else
  printf("Client found\n");

return newsockfd;
```

Kuvio 20. Accept ja socket-muuttujan palautus.

Kun socket-yhteys on kerran saatu aikaiseksi, alustettiin ohjelman toinen säie ja siirryttiin pääohjelmassa silmukkaan, jossa oltiinkin loppu ohjelman suorituksesta. Silmukassa luettiin ensin käsky käyttöliittymältä ja toteutettiin sitten käskyn haluama toimenpide. Siinä tapauksessa, että socket-yhteys meni poikki, luotiin se uudestaan

yllä kuvaillulla socket-yhteyden muodostamisfunktiolla ja aloitettiin sitten silmukan alusta (Kuvio 21). Yhteyden uudelleenluonnin yhteydessä myös lähetettiin sen tämänhetkinen tila, eli onko time lapse-ajo päällä vai pois, takaisin käyttöliittymälle.

```
while (true)
{

  if (!readSock(newsockfd, params))
  {
    //Ota uusi yhteys
    newsockfd = initSocketServer();
    //Lähetä sekvenssiajon status käyttöliittymälle kun yhteys saadaan.
    sendStatusToWebApp(newsockfd);
  }
  else
  {
```

Kuvio 21. Silmukan alku, socket-yhteyden luku ja yhteyden uudelleenotto.

Kun käyttöliittymältä saatu viesti oli tallennettu params-taulukkoon, tarkastettiin minkä tyyppinen viesti oli kyseessä (kuvio 22). Viestejä oli kolmen tyyppisiä, time lapse-ajon pysäytys ja käynnistys, sekä koko ohjelman sammuttaminen. Käskyn tyypin määritteli params-taulukon ensimmäinen muuttuja. Pysäytyskäskyn saadessaan ohjelma lukitsi mutex:lla globaalin timeLapseRunning-muuttujan ja muutti sen arvoa. Mutex:n käyttö oli tässä tilanteessa tosin hieman turhaa, sillä kyseinen globaali muuttuja on vain boolean-arvo, eikä sitä moniajon aikana muuteta kuin todesta epätodeksi. Myöhemmin esitelty time lapse-ajon toteuttava säie vahtii tätä boolean arvoa ja ajaa itsensä alas, kun arvo muuttuu epätodeksi. Pthread_join pysähtyi odottamaan, että säie suoritti itsensä loppuun ennen kuin antaa käyttäjän aloittaa uutta ajoa.

```
else
{

  if (params[0] == StopMessage && timeLapseRunning == true)
  {
    pthread_mutex_lock(&lock);
    timeLapseRunning = false;
    pthread_mutex_unlock(&lock);
    //Odotetaan kunnes sekvenssiajo säie on lopettanut.
    pthread_join(servoMovementThread, NULL);
  }
```

Kuvio 22. Pysäytyskäsky.

Käynnistyskäskyn saadessaan ohjelma asetti parametrit args-structiin. Pthread_create loi servoMovementThread nimisen säikeen, joka suoritti universalThread nimisen aliohjelman, ja jolle annettiin args-tietorakenne parametrina (kuvio 23).

```
  //Sijoitetaan säikeelle menevät parametrit structiin.
  args->type = ServoControl;
  args->sockfd = newsockfd;
  args->seqLength = params[1];
  args->seqTime = params[2];
  args->seqCount = params[3];
  args->servoContId = servoContId;
  args->startSideRight = (params[4] == Right);
  args->cameraAngleStart = params[5];
  args->cameraAngleStop = params[6];

  if (pthread_create(&servoMovementThread, NULL, universalThread, args))
  {
    printf("Servo-ohjaus säie ei toiminut!");
  }
}
```

Kuvio 23. Käynnistyskäsky ja säikeen luonti.

Sammutus-komennolla ohjelma lopetettiin. Jos silmukan suoritus jatkuu, tarkistettiin, oliko ohjelman tila vaihtunut viime kierroksesta ja jos oli, lähetetään tilapäivitys käyttöliittymälle. Silmukan lopussa oli 10 millisekunnin viive, sillä aikaisempi socket-luku funktio ei pysäyttänyt ohjelman suoritusta, joten silmukka kiersi niin nopeasti kuin koneen resurssit mahdollistivat, mikä ei erityisesti moniajon kannalta ollut hyvä asia (kuvio 24).

```
  else if (params[0] == ShutdownMessage && timeLapseRunning == false)
  {
    //Tullaan ulos silmukasta ja ohjelma loppuu.
    break;
  }

  //Lähetä statusviesti jos tila on vaihtunut.
  if(lastIsStoppedState != timeLapseRunning)
  {
    sendStatusToWebApp(newsockfd);
    lastIsStoppedState = timeLapseRunning;
  }
}
delay(10);
```

Kuvio 24. Viimeinen ehto ja tilan päivitys käyttöliittymään.

Kuviossa 25 on alku readSock-funktiosta. Ensin alustettiin puskuri, johon sitten luettiin dataa socketista recv-käskyllä. Ohjelman suoritus pysähtyy recv-komentoon, kunnes serveri vastaanottaa viestin.

```
//Luetaan socket serveriin tullut viesti.
//Ottaa vastaan socketin id:n ja params pointerin, jonne saadut parametrit palautetaan.
//Palauttaa onnistuiko luku.
bool readSock(int newsockfd, long *params)
{
  char buffer[256];
  int n;
  int speed;
  char *message;
  int i;

  bzero(buffer, 256);
  n = recv(newsockfd, buffer, 256, 0);
  message = buffer;
```

Kuvio 25. readSock-aliohjelman alku.

Recv palauttaa numero arvon, joka kertoo, onnistuiko viestin luku. Jos luku on suurempi kuin nolla viesti saatiin onnistuneesti ja sen sisältämät arvot pilkotaan params-tauluun (kuvio 26). Viesti ei tällä hetkellä sisällä muuta kuin välimerkillä erotettuja numeroarvoja ja tässä olisikin yksi jatkokehityksen kohde. Jos työn viimeistelyyn olisi ollut enemmän aikaa, olisi viestiin saanut selkeyttä tekemällä siitä esimerkiksi JSON-muotoisia avain-arvopareja. Nykyisellään c-ohjelmaa tehdessä täytyi vain muistaa ulkoa missä järjestyksessä parametrit käyttöliittymältä saapuvat.

```
//Puretaan saatu viesti
for (i = 0; i < SERVO_PARAMETER_COUNT; i++)
{
  long temp = strtol(message, &message, 10);
  if (errno == ERANGE)
  {
    printf("range error, got ");
    errno = 0;
    return false;
  }
  else
  {
    params[i] = temp;
  }
```

Kuvio 26. readSock-aliohjelman viestin pilkkominen.

Viestinvälitys käyttöliittymään tehtiin send-funktiolla. Viestissä lähetettiin kaksi numeroa. Ensimmäinen kertoi minkä tyyppisestä viestistä oli kyse (ajon tilaviesti, tai ajon edistymisviesti) ja ajon senhetkisen tilan (päällä tai pois) (kuvio 27).

```
void sendStatusToWebApp(int sockfd)
{
  char *timeLapseStatusStr;

  if(!timeLapseRunning){
    timeLapseStatusStr = "0 0";
  }else{
    timeLapseStatusStr = "0 1";
  }
  if (send(sockfd, timeLapseStatusStr, sizeof(timeLapseStatusStr), 0) < 0)
  {
    printf("status send failed :(\n");
  }
}
```

Kuvio 27. Time lapse-ajon tilan päivitys käyttöliittymään.

6.4.4 Prosessiohjaus

Kun pääohjelmassa haluttiin käynnistää time lapse-ajo, luotiin sitä varten oma säie. Säie aloittaa universalThread funktiosta (kuvio 28). Aliohjelmaa käytettiin loppujen lopuksi vain kerran, mutta se suunniteltiin toimimaan tarvittaessa aloituskohtana kaikille ohjelman tarvitsemille säikeille. Säiefunktio tulee c-kielessä olla aina tyyppiä void * (void), mikä tekee argumenttien antamisesta funktiolle hieman erikoista. Koska ohjelma ei tiedä mitä tyyppiä void pointer on, täytyy sen tyyppi määritellä. Työssä tämä tehtiin funktion ensimmäisellä rivillä määrittelemällä oikean tyyppinen actual_args niminen osoitin funktiolle tuotuun tietueeseen. Sitten jatkettiin seuraavaan aliohjelmaan sen mukaan, mikä tyyppi säikeelle oli annettu argumenteissa. Kuten sanottu nykyisellään aliohjelmasta jatketaan aina intervalTimelapseProgram-funktioon. Kun säikeen suoritus viimein loppuu, vapautetaan viimeisenä argumenttien viemä muisti. intervalTimelapseProgram-funktion toiminta on kuvattu tilakaaviona liitteessä 1.

```
//Säie joka voidaa laittaa tekemään saatujen argumenttien mukaan mitä halutaan.
void *universalThread(void *args)
{
  threadArgs *actual_args = args;

  switch (actual_args->type)
  {
  case ServoControl:
    //intervalliajo funktio
    intervalTimelapseProgram(actual_args->servoContId, actual_args->sockfd, actual_args->seqLength,
                             actual_args->seqCount, actual_args->seqTime, actual_args->startSideRight,
                             actual_args->cameraAngleStart, actual_args->cameraAngleStop);
    break;
```

Kuvio 28. Uuden säikeen aloituskohta.

Ensimmäisenä intervalTimelapseProgram aliohjelmassa laskettiin normaalin servon aloituskulma ja kulman muutos ja muunnettiin ne PWM-kortin ymmärtämään muotoon. PWM-kortin 12-bittinen resoluutio tarkoittaa käytännössä sitä, että annetut arvot voivat olla välillä 0-4096 (kuvio 29). Kun taajuus on 50 Hz lukuarvo 4096 tarkoittaa 20 millisekuntia. kun signaalin laskevan kulman halutaan olevan välillä 1-2 ms tarkoittaa tämä PWM-kortin lukuarvoina väliä 205-410.

```
unsigned float cameraAnglePWMCurrent =
              (cameraAngleStart * SERVO_ANGLE_TO_PWM) + SERVO_MIN_ANGLE_PWM;
unsigned float cameraAnglePWMIncrement =
              ((cameraAngleStop - cameraAngleStart) / seqCount) * SERVO_ANGLE_TO_PWM;
```

Kuvio 29. Normaalin servon kulman laskenta.

Kuviossa 30 on laskettu, kuinka pitkään servoa pidetään käynnissä per sekvenssi. Käyttäjä määritteli käyttöliittymässä sekvenssin pituuden eli kuinka pitkän matkan kelkka liikkuu kahden kuvan välissä. Muut laskutoimitukseen tarvittavat arvot olivat ohjelman vakioita. Myös nopeus oli aina vakio. Se oli asetettu suhteellisen hitaaksi, jotta kelkan liike olisi mahdollisimman tarkkaa. Ensiksi laskettiin servon akselissa olevan pyörän kehän pituus. Kehän ja kierrosluvun tulosta saatiin kelkan nopeus minuutissa, joka muutettiin millisekunneiksi. Lopuksi käyttäjän haluama matka jaettiin kelkan nopeudella, josta saatiin ajoaika.

```
// Servon kierrosnopeus halutulla arvolla
float servorpm = SERVO_DRIVE_SPEED * SERVO_SPEED_TO_RPM;
// Pyörän kehän pituus
float pulleyCircumference = M_PI * SERVO_PULLEY_DIAMETER;
int milliseconds = 60 * 1000;
//Laske kuinka kauan servon täytyy pyöriä päästäkseen seqLength verran eteenpäin
//(matkan pituus / (RPM * KEHÄ)*60(muunnos sekunteihin))
int servoDriveTime = round(seqLength /((float)(servorpm * pulleyCircumference) / milliseconds));
```

Kuvio 30. Ympäripyörivän servon ajoajan laskenta.

Seuraavaksi alustettiin kameran asetukset. Käytössä olleen GoPro 3-kameran rajapinta mahdollisti asetusten tekemisen HTTP GET-pyyntöjen avulla. Tätä varten luotiin socket-client jolla kameraan otettiin yhteyttä. Yhteydenotto ja viestinvälitys tehtiin goProController-nimisessä funktiossa, joka otti vastaan lähetettävän käskyn. Aliohjelma oli hyvin samannäköinen kuin socket-serverin luonti, paitsi että kuuntelun sijasta käytetty käsky oli connect. GoPro:n ip-osoite oli aina 10.5.5.9 ja se vastaanottaa yhteyksiä portissa 80 (kuvio 31).

```
if (inet_pton(AF_INET, GOPRO_IP, &serv_addr.sin_addr) <= 0)
{
  printf("\n inet_pton error occured\n");
  return 0;
}
if (connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
{
  printf("\n Error : Connect Failed \n");
  return 0;
}
```

Kuvio 31. Socket-client connect.

Kuviossa 32 on GET-pyynnön muodostus. Yksinkertaisin mahdollinen pyyntö sisälsi vain yhden kuvan mukaisen rivin ja yhden tyhjän rivin sen alla. Tyhjä rivi on myös pakollinen. Cmd muuttujassa on haluttu käsky, jotka olivat kaikki muotoa http://10.5.5.9/param1/PARAM2?t=PASSWORD&p=%OPTION (Konrad Iturbe. 23.03.2017.).

```
/// GET-pyynnön muodostus
snprintf(sendline, 1024,
         "GET %s HTTP/1.0\r\n"
         "\r\n",
         cmd);
```

Kuvio 32. GET-pyynnön muodostus.

Kuviossa 33 GET-pyyntö lähetetään kameralle ja kameralta luetaan vastaus. Jos vastauksen ensimmäinen rivi oli muotoa ” HTTP/1.0 200 OK” oli kamera vastaanottanut viestin ja toteuttanut halutun käskyn. Puskuriin tuleva viesti on kokonaisuudessaan melko iso eikä siitä tarvita kuin edellä mainittu ensimmäinen rivi, joten yhteys suljetaan jokaisen viestinlähetyksen lopuksi ja avataan sitten taas uudestaan seuraavassa.

Tämä siksi että muuten ei voitaisi olla varmoja, onko puskuri tyhjä ja luettu data voisi näyttää väärältä, vaikka viesti olisikin mennyt oikein perille.

```
/// GET-pyynnön lähetys
if (write(sockfd, sendline, strlen(sendline)) >= 0)
{
  //Luetaan vastaus
  read(sockfd, recvline, 1024);
  if (strcmp(recvline, GOPRO_CMD_OK))
  {
    ret = true;
  }

  close(sockfd);
}
return ret;
```

Kuvio 33. GET-pyynnön kirjoitus ja vastauksen luku.

Kuviossa 34 näkyvät kaikki kameran vaatimat asetukset. Esimerkiksi GP_INIT_PHOTO_MODE asettaa kameran valokuvatilaan, eli kamera ottaa yksittäisiä kuvia. Viesti näyttää seuraavalta: ”camera/CM?t=salasana3&p=%01”. Kaikkien asetusten on mentävä läpi, jotta kuvaaminen voidaan aloittaa.

```
//Kaikki gopron tarvitsemat asetukset ennen kuvausta.
//Palauttaa onnistuivatko kaikki asetukset.
bool goProSettings()
{
  return goProController(GP_INIT_PHOTO_MODE) && goProController(GP_INIT_ORIENTATION_UP)
      && goProController(GP_INIT_RESOLUTION) && goProController(GP_INIT_PROTUNE_OFF)
      && goProController(GP_INIT_AUTO_OFF_NEVER);
}
```

Kuvio 34. GoPro-kameran asetukset.

Time lapse-kuvaus aloitetaan aina kiskon päästä. Ennen varsinaisen sekvenssiohjelman aloitusta ajetaan siis kisko käyttäjän käyttöliittymässä määrittelemään reunaan. Kuviossa 35 on esitetty ohjelman toiminta, jos halutaan aloittaa vasemmalta, mutta vasemman puolen valokennon signaali ei ole poikki. Tässä tapauksessa laitetaan servo ajamaan vasemmalle, kunnes laita saavutetaan. DigitalRead lukee sille annettua Raspberry Pi:n pinniä. Funktio palauttaa epätosi, jos valokennon signaali on poikki.

```
//Jos ohjelma halutaan aloittaa vasemmalta mutta ei olla vasemmassa
//laidassa ajetaan kunnes laita saavutetaan.
if (!startSideRight && digitalRead(SERVO_PHOTOCELL_LEFT))
{
  servoMovement(servoContId, Left);
  while (digitalRead(SERVO_PHOTOCELL_LEFT))
  {
    delay(10);
  }
  servoMovement(servoContId, Stop);
```

Kuvio 35. Kelkan ajo laitaan.

Myös ympäripyörivän servolle käytetyt PWM-arvot ovat väliltä 205-410. Kyseisillä raja-arvoilla servo pyörisi täyttä vauhtia eri suuntiin, joten lukuarvojen puolessa välissä, arvolla 307(SERVO_STOP_SPEED), servo on pysähdyksissä. Oikeasti käytetty nopeus ei ole lähellekään maksimia vaan liikutukseen käytetty SERVO_DRIVE_SPEED poikkeaa keskeltä 15 pykälää (kuvio 36).

```
case Left:
  pwmOn = 0;
  pwmOff = SERVO_STOP_SPEED - SERVO_DRIVE_SPEED;
  setServoXPWM(0, pwmOn, pwmOff, servoContId);
  break;
}
```

Kuvio 36. Servoajon suunta.

Kuviossa 37 on PWM-kortin rekisteriin kirjoitus. Funktiossa käytettiin wiringPi-kirjaston 8 bitin rekisteriinkirjoitusta. Koska PWM-kortin rekisteri oli 16 bittiä, tehtiin kirjoitus kahdessa osassa. Kaksi ensimmäistä riviä kirjoittavat signaalin nousevan reunan paikan rekisteriin ja kaksi viimeistä taas vastasivat laskevan reunan arvon kirjoituksesta.

```
// Aliohjelma, jolla kirjoitetaan nopeus servolle ohjainkortin rekistereihin
void setServoXPWM(int servo, int pwmOn, int pwmOff, int servoContId)
{
  wiringPiI2CWriteReg8(servoContId, LED0_ON_L + servo * 4, pwmOn & 0xFF);
  wiringPiI2CWriteReg8(servoContId, LED0_ON_H + servo * 4, pwmOn >> 8);
  wiringPiI2CWriteReg8(servoContId, LED0_OFF_L + servo * 4, pwmOff & 0xFF);
  wiringPiI2CWriteReg8(servoContId, LED0_OFF_H + servo * 4, pwmOff >> 8);
}
```

Kuvio 37. Servon nopeuden kirjoitus rekisteriin.

Liitteessä 2 on esitetty sekvenssiajon logiikka tilakaaviona. Ohjelma pyörii silmukassa, joka on delay-funktiolla laitettu kiertämään 10 millisekunnin välein. Kierroksella tar-

kastellaan kahta kelloa, jotka ilmoittavat kauanko servo on ollut päällä ja kauanko edellisen kuvan otosta on ollut aikaa. Servo pysäytettiin, kun haluttu aika oli ajettu. Kun toinen kello ylitti käyttäjän määrittelemän ajan, otettiin kuva ja laitettiin servo uudestaan päälle. Kuvan otto tapahtuu samalla logiikalla kuin GoPro-kameran asetusten muuttaminen, mutta nyt annettu käsky on muotoa "bacpac/SH?t=salasana&p=%01".

Aina kun yksi kuvaa saatiin otettua, päivitettiin tilannetietoa web ohjelmaan samaan tyyliin kuin tilaviestin lähetyksessä. Ainoana erona oli se, että viestiä varten laskettiin prosenttiarvo ohjelman etenemisestä kuvion 38 osoittamalla tavalla.

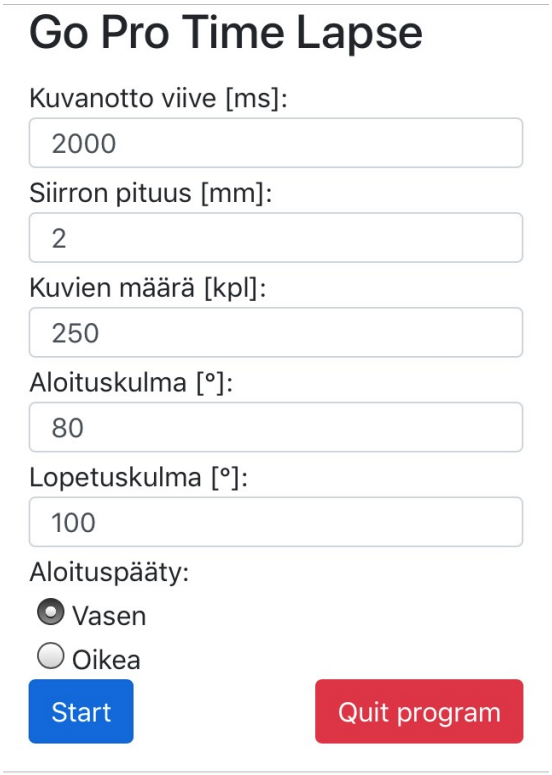
```
progress = 100 - (int)(((float)seqCount / startSeqCount) * 100);
sendProgressToWebApp(newsockfd, progress);
```

Kuvio 38. Edistymis-tiedon laskenta ja lähetys.

Tätä jatkettiin, kunnes käyttäjän määrittelemä määrä sekvenssejä on ajettu, saavutetaan kiskon pää tai käyttäjä lopettaa ajon kesken web ohjelmasta. Jos ohjelman suoritus loppuu kesken, säie lopettaa itsensä hallitusti, eli tulee ulos silmukasta, pysäyttää ympäripyörivän servon ja antaa tilapäivityksen web ohjelmaan. Tämän jälkeen säikeestä tullaan ulos ja se päättyy.

6.4.5 Käyttöliittymä

Ohjelman käyttöliittymäosio on huomattavasti lyhyempi. Kuviossa 39 on näkymä käyttöliittymässä ennen ajon aloitusta.



Kuvio 39. Käyttöliittymän parametrilomake.

Node serverin tehtävänä oli välittää viestejä käyttöliittymän ja C-ohjelman välillä, sekä ohjata http-pyynnöt oikein. Kuviossa 40 on esitetty socket yhteyden luominen. Yhteyden luominen oli huomattavasti yksinkertaisempi kuin C-ohjelmassa, missä kaikki oli tehtävä itse. Noden tapauksessa Net-kirjasto hoiti asian koodarin puolesta.

```
var client = new net.Socket();
connectClient();

function connectClient(){
    client.connect(2050, 'localhost');
}
```

Kuvio 40. Socket yhteys C-ohjelmaan.

Jos yhteydenotossa tapahtui virhe, odotettiin 3 sekuntia ja sitten yritettiin yhdistää uudestaan (Kuvio 41). Ilman tätä koodinpätkää Node-serveri piti käynnistää aina C-ohjelman jälkeen, tai se kaatui.

```
client.on('error', function(exception) {
        setTimeout(connectClient, 3000);
});
```

Kuvio 41. Socket virheentarkistus.

Jos C-ohjelma lähetti viestin Nodelle, se oli joko statusviesti (ajetaanko vai ollaanko pysähdyksissä) tai viesti ajon etenemisestä, jonka mukaan etenemispalkkia päivitettiin sivulla. Io viittaa Vuen ja Noden väliseen socket-yhteyteen, minne Node välitti C-ohjelman viestit (kuvio 42).

```
client.on('data', function(data) {

    var raspParams = data.toString().split(" ");
    if (raspParams[0] == 0)
    {
        servoStatus = parseInt(raspParams[1]);

        io.emit('statusResponse', servoStatus);
    }
    else if(raspParams[0] == 1)
    {
        progress = parseInt(raspParams[1]);

        io.emit('progress', progress);
    }
});
```

Kuvio 42. Viestin vastaanotto C-ohjelmasta.

Kuviossa 43 on näytetty mitä tapahtuu, kun yhteys Vuen puolelle oli saatu. Kun Vue-client lähetti Node-serverille ajoon tarvittavat parametrit serveri välitti ne eteenpäin.

```
io.on('connection', function(socket){

    socket.on('params', function(params){
        console.log(params);
        //Kirjoita raspberryyn
        client.write(StartMessage + ' ' +
                     params.seqLength + ' ' +
                     params.seqTime + ' ' +
                     params.seqCount + ' ' +
                     params.seqStartingEdge + ' ' +
                     params.cameraAngleStart + ' ' +
                     params.cameraAngleStop);
    });
```

Kuvio 43. Parametrien välitys C-ohjelmalle.

Kuviossa 44 on esitetty Express-ohjelmistokehyksestä käytetyt ominaisuudet. Listen asetti ohjelman kuuntelemaan portista 3000. Koska käyttöliittymä oli yhden sivun ohjelma, app.get-käsky ohjasi kaikki porttiin tulevat http-pyynnöt index-html-sivulle.

```
app.use(express.static(__dirname + '/'));

app.get('/*',function(req,res){
  res.sendFile(path.join(__dirname+'/index.html'));
});

http.listen(3000);
```

Kuvio 44. Routing ja portin kuuntelu.

Seuraavana oli Vue-ohjelman kirjoitus. Kuviossa 45 on socket-clientin luonti. Tämä toimi pelkällä käskyllä io(). Alempana kuvassa on esitetty mitä tehtiin, kun saatiin Nodelta ilmoitus ohjelman tilasta. Kyseinen pätkä oli tiedoston ainoa osa, mikä ei varsinaisesti liittynyt Vueen.

```
var socket = io();

socket.on("statusResponse", function(data) {
  app.status = data;
  if (app.status === 0) {
    app.progressPercent = 0;
  }
});
```

Kuvio 45. Socket.io yhteydenotto.

Vue otettiin käyttöön kuvion 46 mukaisesti. Vuesta luotiin ensin instanssi. Tämä instanssi sisälsi kolme kohta: el, data ja methods. El osoittaa Vuen root-elementtiin, data sisältää Vuen käyttämät muuttujat ja methods sen aliohjelmat eli metodit.

```
var app = new Vue({
  el: "#app",
  data: {
    status: 0,
    progressPercent: 0,
    params: {
      seqTime: 0,
```

Kuvio 46. Vuen luonti ja sen sisältämä data.

Käyttöliittymän tärkein tehtävä oli antaa käyttäjälle lomake, mihin syöttää Ajo-ohjelman tarvitsemat parametrit. Tavallinen html submit kuitenkin lataa koko sivun uudestaan mitä ei haluttu, joten kuvassa 47 on esitetty se, kuinka lomakkeen käskettiin lähettää start-nappia painettaessa (submit) lomakkeen tiedot Vuen sendParams-metodille. Weppi-osan tyylitiedostona käytettiin Bootstrappia.

```
<div v-show="status === 0">
    <form @submit.prevent="sendParams">
        <div>
            <label for="sequenceTime">Sekvenssin kesto:</label>
            <input type="number" v-bind:class="{ 'is-invalid': validations.seqTime }"
                class="form-control" v-model="params.seqTime" id="sequenceTime"
                required min="1">
            <div class="invalid-feedback"> Virhe: Sekvenssin kesto pitää olla pidempi kuin
        </div>
        <div>
```

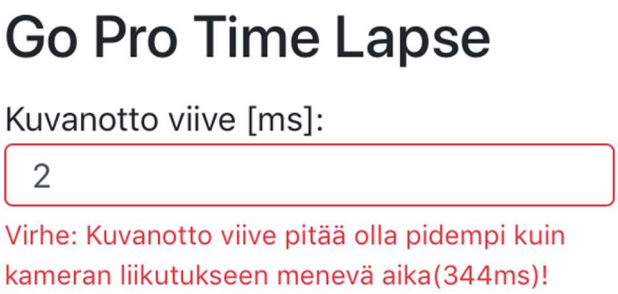
Kuvio 47. HTML-lomakkeen alku.

Kuviossa 48 on esitetty sendParamas-metodin alku. Kun käyttäjä oli painanut start-nappia, tarkastettiin että annetut syötteet olivat oikeanlaisia ja halutuissa rajoissa. Jos yksikin validations-objektin muuttuja oli arvoltaan tosi, eli validointi ei mennyt läpi, näytettiin virheelliset kentät lisäämällä niihin Bootstrapin luokka is-invalid.

```
methods: {
  sendParams: function() {
    //Tarkista syötteet
    app.validations.cameraAngleStart = app.params.cameraAngleStart < 60 ||
                                       app.params.cameraAngleStart > 120;
    app.validations.cameraAngleStop = app.params.cameraAngleStop < 60 ||
                                      app.params.cameraAngleStop > 120;
    app.validations.seqCount = app.params.seqCount < 25;
    app.validations.seqLength = app.params.seqLength < 1;
```

Kuvio 48. Syötteen validointi

Kuviossa 49 on esitetty miten virhe näkyi käyttöliittymäsivulla.



Kuvio 49. Virhe parametrien annossa.

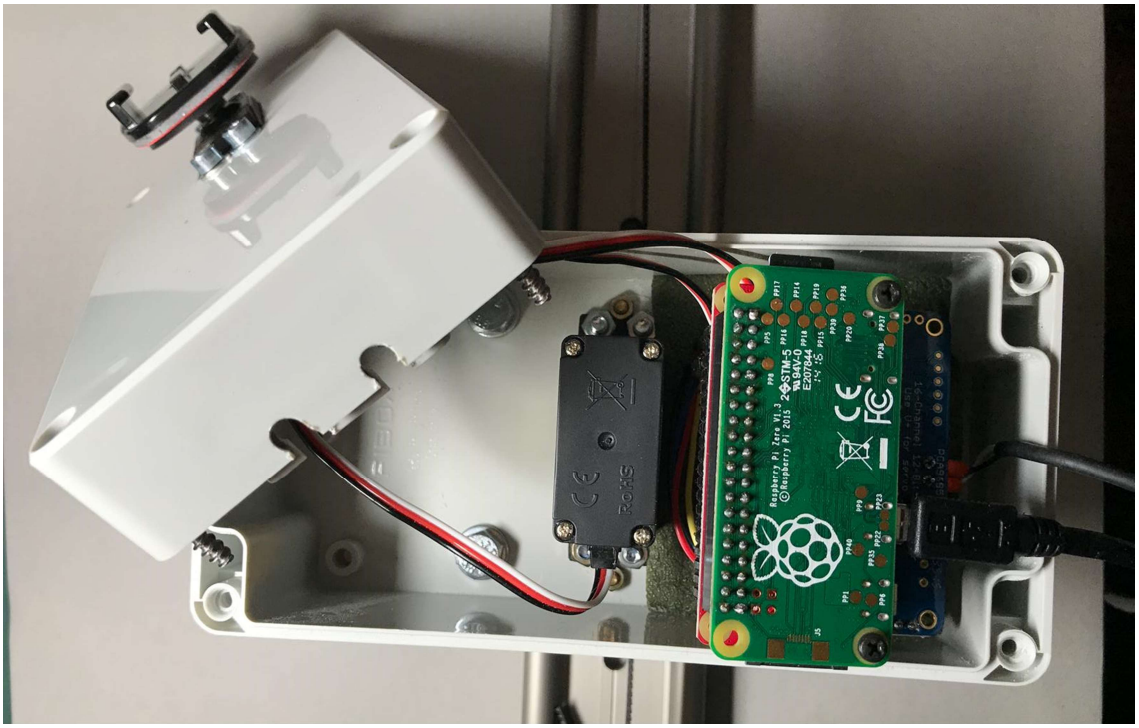
Validoinnin onnistuessa annetut parametrit lähetettiin Node-serverille käskyllä emit (Kuvio 50).

```
if (!hasErrors) {
  socket.emit("params", app.params);
}
```

Kuvio 50. Parametrien lähetys.

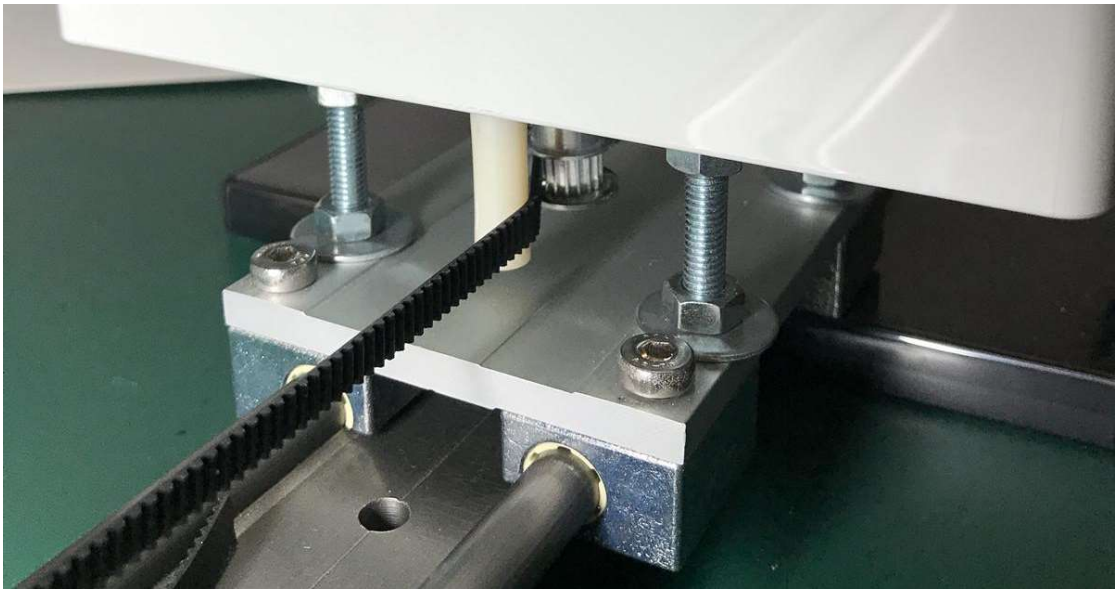
6.5 Mekaaninen kokoonpano

Kuviosta 51 näkyy, kuinka elektroniikka asennettiin Fibox Euronord-kotelon sisään. Myös virtalähteen oli tarkoitus olla kotelossa, mutta tilattu kotelo oli liian kapea tähän. Kotelo onkin tarkoitus vaihtaa isompaan heti kun se on mahdollista. Kotelon kanteen asennettiin kameran kääntöservo ja sen päälle kameran kiinnike ja siihen GoPro 3 Black-kamera. Kotelon pohjassa oli reikä ympäripyörivälle servolle. Se asennettiin akseli alaspäin, niin että se pyöritti kotelon alla olevaa vetopyörää.



Kuvio 51. Servot ja Raspberry Pi laatikon sisällä.

Kelkan päälle asennettiin kotelo pienillä korotuspaloilla niin, että kotelon ja kelkan väliin mahtui ajoservon vetopyörä. Vetopyörä oli hammasratas, mitä vasten vedettiin kiskon päistä kireälle asennettu hammashihna (kuvio 52). Kun ajoservoa pyöritettiin, saatiin kelkka liukumaan johteen päästä päähän. Kotelon kylkiin asennettiin valokennoparit, joiden säde katkaistaan johteen päätyyn asennetulla haitalla. Kun säde katkeaa, ajoservo pysäytetään.



Kuvio 52. Hammashihnaveto.

7 Tulokset

7.1 Lopputulos

Lopputuote täytti vaatimusmäärittelyn muilta paitsi yhdeltä osalta. Lopputuloksessa kameran kulmaa voi säätämään vain horisontaalisesti. Ominaisuuden jääminen pois ei kuitenkaan ollut suuri menetys. Vertikaalinen kulman säätö oli muutenkin todettu vähiten hyödylliseksi ominaisuudeksi käytännössä.

Tuloksena oli laite, mihin otetaan älypuhelimella yhteyttä. Käyttäjän asettaessa selaimeen laitteen osoitteen avautuu näytölle käyttöliittymä, mihin käyttäjä syöttää haluamansa ajon asetukset ja painaa sen jälkeen start-nappia. Tämän jälkeen kamerakisko aloittaa kuvanoton. Käyttäjälle päivitetään käyttöliittymään tiedot ajon edistymisestä. Ajo oli mahdollista keskeyttää käyttöliittymästä. Muutoin ohjelman suoritus loppuu joko silloin kun haluttu määrä kuvia on otettu, tai kun kiskon pää saavutetaan.

7.2 Jatkokehitys

Laitteelle löytyy useita varteenotettavia kehityspolkuja. Työhön voisi lisätä eri kameroiden, kuten uudempien Gopro-mallien ohjausta tai eri ajo-ohjelmia. Sen sijaan, että kameraa ohjattaisi GET-pyynnöillä, joissa olisi jokaisessa mukana kameran vaatima salasana, on GoPro:n kameroissa 4-versiosta lähtien vaadittu, että kamerasta ja ohjausohjelmasta tehdään pari. GoPro:n voisi myös vaihtaa isompaan järjestelmäkameraan, vaikkakin se luultavasti vaatisi parempia servoja.

Lisäksi ohjelmaan voisi lisätä videoajon. Tällöin liikutus ja kuvanottointervallien sijasta kamera ottaisi valmiiksi videokuvaa samalla, kun servot liikuttaisivat sitä hitaasti. Tämä muuttaisi servon ohjausta jonkin verran, sillä nykyisellään kameraa kääntävä servo voi välittömästi uuden sekvenssin alussa asettaa seuraavan kuvan vaatimiin asentoihin, siinä missä tasainen kamera-ajo vaatisi kulman muuttamista tasaisin väliajoin, jotta tuotettu elokuva näyttäisi vakaalta. Kelkkaa liikuttavaan servon ohjaukseen tulisi sen verran muutosta, että sitä pitäisi pystyä ajamaan eri nopeuksilla.

Koska kaikki työn elektroniikka on samassa kotelossa, voi kameran vaihtaa kulkemaan kiskon sijasta esimerkiksi vaijerien varassa samalla tavalla kuin Syrp Genie. Tällöin hammashihna vaihtuisi vetonaruun ja vaijerit vaatisivat uuden kiinnityksen koteloon, mutta elektroniikkaan tai ohjelmaan ei muutoksia tarvittaisi.

7.3 Haasteet

Projekti aloitettiin alun perin Netbeans-editorilla, koska ohjelman siirtäminen ja kääntö RaspberryPi:llä toimi suhteellisen helposti. Editori osoittautui kuitenkin lopulta huonoksi valinnaksi. Netbeans on tarkoitettu Java-ohjelmointiin, eikä siitä ollut juurikaan apua C:llä koodatessa. Ohjelma myös jumitti ja käytti tietyissä tilanteissa 90 prosenttia koneen prosessoritehosta, mikä on katastrofaalista ottaen huomioon, että ohjelma on pelkkä tekstieditori.

Suurimmaksi ongelmaksi kuitenkin muodostui koodin siirtäminen Raspberry Pi:lle, sillä tiedostoja etälaitteelle päivitettäessä Netbeans vertaa uuden ja vanhan tiedoston aikaleimaa toisiinsa eikä päivitä tiedostoa, jos uusi versio on sama tai vanhempi. Tämä ei normaalisti olisi ongelma, mutta Raspberry Pi.ssä ei ole kellopiiriä, joten se ei pidä aikaa silloin, kun kortti on pois päältä ja piirin ajan ollessa väärin Netbeans ei päivitä tiedostoja. Kortti ei myöskään saa päivitettyä aikaa internetin kautta, koska työn testausta varten Pi:n täytyy olla GoPron:n Wlan-verkossa, josta ei ole yhteyttä internettiin.

Tämän lisäksi haasteellista projektissa oli sen kompleksisuus. Kirjoitettua koodia ei sinällään tullut valtavia määriä (noin 1000 riviä), mutta lähes jokainen uusi ominaisuus vaati uuden asian opiskelua.

8 Johtopäätökset ja pohdinta

Työn tekeminen oli hyvin mielenkiintoista, mutta vaati myös pitkää pinnaa. Ohjelman tekeminen ei ollut missään vaiheessa helppoa, sillä jokainen uusi ominaisuus oli jotain uutta ja tuntematonta. Työ sisälsi myös paljon elektroniikkaa, joten mahdollisia ongelmakohteita riitti. Aina ei voinut olla varma johtuiko virhe itse tehdystä ohjelmasta vai siitä oliko esimerkiksi PWM-ohjainkortti päässyt sellaiseen tilaan, että sitä ei saanut resetoitua. Tämä saattoi joskus olla hyvin turhauttavaa.

Javascript-ohjelmointi oli työn kirjoittajalle täysin uutta, joten siinä riitti opiskeltavaa. Työhön harkittiin useita eri kirjastoja muun muassa React ja Angular 2, mutta loppujen lopuksi päädyttiin Vueen. Vue sopi projektiin hyvin, sillä se oli sopivan yksinkertainen ja nopeasti omaksuttavissa. Tämän aiheen opit tulevat varmasti tarpeeseen jatkossa.

Elektroniikka ja sulautetut järjestelmä eivät ole koskaan tuntuneet kovin kiinnostavalta aiheelta itselle. Työssä tuli kuitenkin opittua paljon uutta tästäkin aihealueesta ja perustiedon kasvattaminen on aina hyvä asia. Työn elektroniikkapuoli ei myöskään vaadi suurempaa jatkokehitystä, kun sen on kerran saanut toimimaan. Muutokset, kuten erilaisten kamera-ajo-ohjelmien teko tai tuki eri kameroille hoituisivat puhtaasti ohjelmistomuutoksilla. Näitä ominaisuuksia tulenkin lisäämään työhön jatkossa.

Lähteet

About Node N.d. Tietoa NodeJS javascript-runtime ympäristöstä. Viitattu 10.05.2018. https://nodejs.org/en/about/

Atlassian. Bitbucket. N.d. Tietoa Bitbucket versionhallintapalvelusta. Viitattu 09.05.2018. https://fi.atlassian.com/software/bitbucket

Conor Lyons. A History Of The Raspberry Pi. 2015. Raspberry Pi:n historia. Viitattu 30.04.2018. http://novadigitalmedia.com/history-raspberry-pi/

Chowdhurry Farabee. History of Time Lapse photography. 2013. Time lapse-kuvauksen historiaa. Viitattu 04.04.2018. http://web.colby.edu/am297/history-of-time lapse-photography/

Eben Upton. RASPBERRY PI 3 MODEL B+ ON SALE NOW AT $35. 2018. Raspberry Pi:n myyntilukuja. Viitattu 28.04.2018. https://www.raspberrypi.org/blog/raspberry-pi-3-model-bplus-sale-now-35/

Edelkrone SliderPLUS. N.d. Tietoa SliderPLUS-laitteesta Viitattu 13.05.2018. https://edelkrone.eu/products/sliderplus?gclid=CjwKCAjw8r_XBRBkEiwAjWGLlMyNcKXcz3f6tko1GxmikLCh9NENkLTo-EtLuF-3v-frFmzuLFg4jhoCJuMQAvD_BwE

Express N.d. Tietoa Express ohjelmistokehyksestä Nodelle. Viitattu 10.05.2018. https://expressjs.com/

Git. About. N.d. Tietoa Git-versionhallinnasta. Viitattu 09.05.2018. https://git-scm.com/about/branching-and-merging

GoPro Hero3 Manual. N.d. GoPro Hero 3 kameran käyttöopas. Viitattu 29.04.2018. https://gopro.com/content/dam/help/hero3-black-edition/manuals/HERO3_UM_Black_ENG_REVD_WEB.pdf

Gordon Henderson. Wiring Pi Setup. N.d. Wiring Pi-kirjaston asetukset. Viitattu 30.04.2018. http://wiringpi.com/reference/setup/

I2C-bus specification and user manual. N.d. I2C väylän määrittely- ja käyttöohjedokumentti. Viitattu 07.05.2018. https://www.nxp.com/docs/en/user-guide/UM10204.pdf

Konrad Iturbe. Wifi Commands for HERO3/3+. 2017. GoPro 3:n hallinta HTTP-pyynnöillä. Viitattu 29.04.2017. https://github.com/KonradIT/goprowifihack/blob/master/HERO3/WifiCommands.md

Marko Hassinen N.d. Mikä on soketti. Viitattu 10.05.2018. http://www.cs.uku.fi/~mhassine/VOH/Luennot/lu3.html

NEC Display Solutions. NEC Display Solutions announces collaboration with Raspberry Pi. 2016. Raspberry Pi minitietokone osana kuluttaja electroniikkaa. Viitattu 28.04.2018. https://www.nec-display-solutions.com/p/hq/en/news/dp/Products/Shared/News/2016/PressReleases/Company/RaspberryPi/RaspberryPi.xhtml

Parallax Feedback 360° High-Speed Servo (#900-00360). N.d. Paralax 900-00360 servon datalehti. Viitattu 22.04.2018. https://www.parallax.com/sites/default/files/downloads/900-00360-Feedback-360-HS-Servo-v1.2.pdf

PCA9685 Product Datasheet. N.d. PWM-kortin datalehti. Viitattu 29.04.2018. https://cdn-shop.adafruit.com/datasheets/PCA9685.pdf

Raspberry Pi Foundation. About us. N.d. Tietoa Raspberry Pi säätiöstä. Viitattu 28.04.2018. https://www.raspberrypi.org/about/

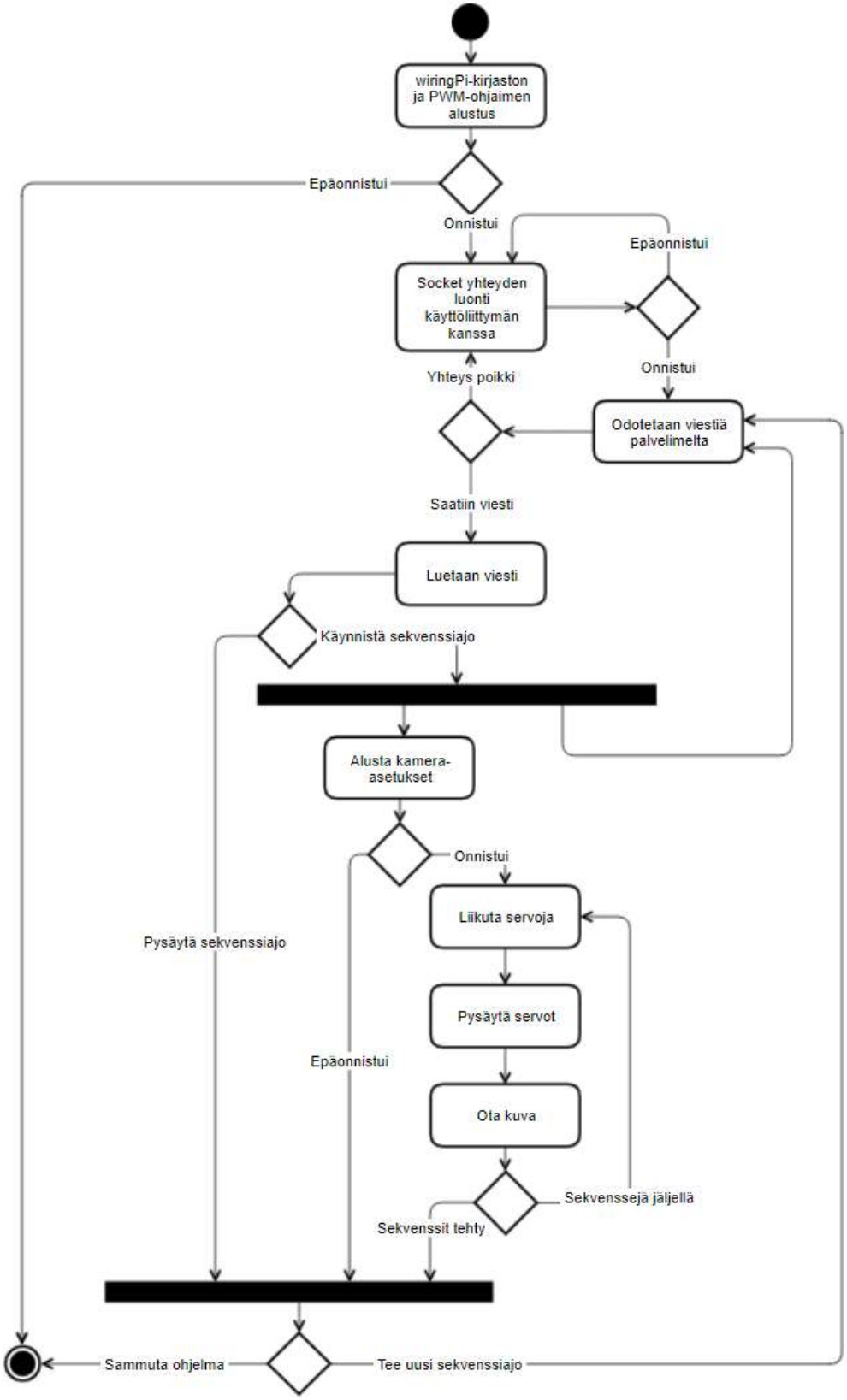
Syrp Genie N.d. Tietoa Genie-laitteesta. Viitattu 13.05.2018. https://syrp.co.nz/products/genie

Vue.js Introduction. N.d. Vue.js ohjelmistokehyksen esittely. Viitattu 10.05.2018. https://vuejs.org/v2/guide/

What is a Continuous Servo? N.d. Servon toimintaperiaatteen kuvaus. Viitattu 22.04.2018. http://education.rec.ri.cmu.edu/content/electronics/boe/robot_motion/1.html

Liitteet

Liite 1 C-ohjelman tilakaavio.



Liite 2 Sekvenssiajon tilakaavio.