Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Juha Airikkala

Torninosturin automaatiojärjestelmän vianhavaitsemis- ja pysäytyslaitteen suunnittelu ja toteutus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Sähkö- ja automaatiotekniikka

Insinöörityö

13.11.2019

Tekijä Otsikko Sivumäärä Aika	Juha Airikkala Torninosturin automaatiojärjestelmän vianhavaitsemis- ja pysäytyslaitteen suunnittelu ja toteutus 34 sivua + 1 liite 13.11.2019
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	sähkö- ja automaatiotekniikka
Ammatillinen pääaine	automaatiotekniikka
Ohjaajat	tuotekehitysinsinööri Simo Simolin lehtori Kristian Junno

Insinöörityön tavoite oli kehittää Trenox Oy:n kehityksessä olevaan torninosturin automaattiseen ohjausjärjestelmään erillinen vianhavaitsemis- ja pysäytyslaite. Laitteen haluttiin havaitsevan automaatiojärjestelmän laitteiden vikatiloja ja vian esiintyessä pysäyttävän nosturin hallitusti ilman hätäjarrutusta. Laitteen haluttiin toimivan myös manuaalisella etälaukaisulla ja näin korvaavan nosturinjuurelle vedetyn käsikäyttöisen virtakytkimen, jota oli käytetty aiempana turvajärjestelynä. Työn oli tarkoitus toimia automaatiojärjestelmän turvajärjestelmän perustana, jonka kehitystä pystyttäisiin myös jatkamaan tulevaisuudessa.

Työn aikana kartoitettiin erilaisten vianhavaitsemistapojen toteutusmahdollisuuksia ja tarvetta. Tuloksena oli laite, jolla automaattinen ohjausjärjestelmä pystytään kytkemään irti nosturin ohjauselektroniikasta aiheuttaen nosturin rauhallinen pysähtyminen. Laitteella pystytään myös laukaisemaan nosturin alkuperäinen hätä-seis-järjestelmä, joka aiheuttaa hätäjarrutuksen. Laitteeseen onnistuttiin luoda joitain automaattisia vianhavaitsemistoimintoja ja siihen tehtiin web-selain -pohjainen käyttöliittymä, jonka kautta laitteen manuaalinen laukaisu ja muu hallinta suoritetaan.

Laitteen alustana käytettiin Kunbuksen Revolution Pi -teollisuustietokonetta ja I/O-moduuleja. Kommunikaatiossa automaatiojärjestelmän eri osien kanssa hyödynnettiin XBee-radiota, TCP/IP-yhteyksiä ja sarjaliikennettä. Automaatiojärjestelmän irti kytkeminen ja hätäseis-järjestelmän laukaiseminen toteutettiin releiden avulla. Ohjelmoinnissa käytettiin Node RED -ohjelmointityökalua, sekä Python- ja JavaScript-ohjelmointikieliä. Laite asennettiin Trenox Oy:n testauskäytössä olleeseen torninosturiin.

Työstä saatiin hyödyllistä tietoa automaatiojärjestelmän kehitystyöhön, sekä nopeampi ja turvallisempi pysäytysjärjestelmä jo olemassa olevaan järjestelmään. Valmistuneesta laitteesta laadittiin myös englanninkielinen dokumentaatio Trenox Oy:n käyttöön.

Avainsanat	automaattiohjaus, Node-RED, Revolution Pi, torninosturi

Author	Juha Airikkala
Title	Design and Build of Error Detection and Stopping Device for Tower Crane's Automation System
Number of Pages	34 pages + 1 appendix
Date	13 November 2019
Degree	Bachelor of Engineering
Degree Programme	Electrical and Automation Engineering
Professional Major	Automation Engineering
Instructors	Simo Simolin, R&D Engineer Kristian Junno, Senior Lecturer

The purpose of this thesis work was to develop an external error detection and stopping device to the automatic tower crane control system under Trenox Oy's development. The device was wanted to detect malfunctions of the automatic control system's devices and to stop the crane in a controlled manner without emergency braking. The device was also wanted to function with manual remote launch and thus to replace the manual power switch on the crane's base, which was used as the former safety arrangement. The work was meant to be a base also for the safety system, the development of which could be continued in the future.

During the work possibilities and needs for different malfunction detection methods was surveyed. The work resulted in a device with which the automatic control system can be decoupled from the crane's control electronics and so to cause the crane's easy stopping. The crane's original emergency stop can also be launched with the device. Some automatic malfunction detection features were created, and web-browser based user interface was made for controlling the device.

The device was based on Kunbus GmbH's Revolution Pi industrial PC and I/O-modules. XBee-radios, TCP/IP-connections and serial communication were utilized in communication with devices of the automation systems. Decoupling of the automation system and launching of the emergency stop was done with a set of relays. Node-RED programming tool and Python and JavaScript programming languages were used for programming. Finally, the device was installed to a tower crane in Trenox Oy's test use.

The work resulted in a quicker and safer stopping system for the automation system and during the work useful knowledge was gathered for the future development of the system. A documentation about the device was written in English for Trenox Oy's use alongside this thesis.

Keywords	Automatic control, Node-RED, Revolution Pi, Tower crane

Sisällys

Lyhenteet ja käsitteet

CRC	*Cyclical Redundancy Check.* Tietoliikenneyhteyksissä käytetty laskennallinen virheidentarkastusmetodi.
GPS	*Global Positioning System.* Maailmanlaajuinen paikallistamisjärjestelmä.
GPIO	*General Purpose Input Output.* Yleiskäyttöinen sisääntulo ulostulo, Viittaa mikrokontrollerien ja yhdenpiirintietokoneiden käyttämiin I/O-portteihin.
IPC	*Industrial PC.* Teollisuustietokone.
I/O	*Input/Output.* sisääntulo ulostulo. Lyhenne, jolla viitataan portteihin, jotka vastaanottavat tai lähettävät jännite- tai virtasignaaleja.
IoT	*Internet of Things.* esineiden Internet.
Link Box	Ohjauslaite joka toimii torninosturin ja tietokoneen välisenä rajapintana. mahdollistaen minkä tahansa torninosturin tietokoneohjauksen.
Node-RED	Graafinen ohjelmointityökalu, joka on suunnattu IoT-sovelluksiin.
TCP	*Transfer Control Protocol,* TCP/IP-protokolla perheeseen kuuluva tiedonsiirto protokolla.
USB	*Universal Serial Bus.* Tietokoneissa yleinen sarjaväyläarkkitehtuuri.
WLAN	*Wireless Local Area Network.* Langaton lähiverkko.
XBee-radiot	Teollisuuden tarpeisiin suunnattuja luvasta vapaita taajuuksia käyttäviä tiedonsiirtoradioita.

1 Johdanto

Nykyaikana automaatiota löytyy kaikkialta ympäriltämme. Tuotantolaitoksissa prosessit säätyvät ilman ihmisen aktiivista puuttumista, rakennuksissa lämmitys ja ilmanvaihto säätyvät itsestään, logistiikkakeskuksissa tavarat siirtyvät liukuhihnoilla ja robottitrukeilla. Automaation avulla laitteiden ja prosessien hyötysuhteita on saatu parannettua ja turvallisuutta lisättyä.

Kone, jolle kuitenkaan ei vielä ole automaatioratkaisuja saatavilla, on torninosturi. Torninosturi on kone, joka on työmaan keskiössä ja joka pitää työmaan liikkeessä. Nosturin pysähtyessä pysähtyy myös työmaa. Nykyaikaisella tekniikalla olisi mahdollista automatisoida torninosturi ja näin saada aikaan nosturi, jonka käyttäminen ei vaatisi kuskin paikalla oloa. Nosturi olisi kenen tahansa käytettävissä mihin kellon aikaan tahansa. Ilman huolta henkilöstöjärjestelyistä. Tähän puutteeseen Trenox Oy on vastaamassa kehityksessä olevalla torninosturin automatisointijärjestelmällä.

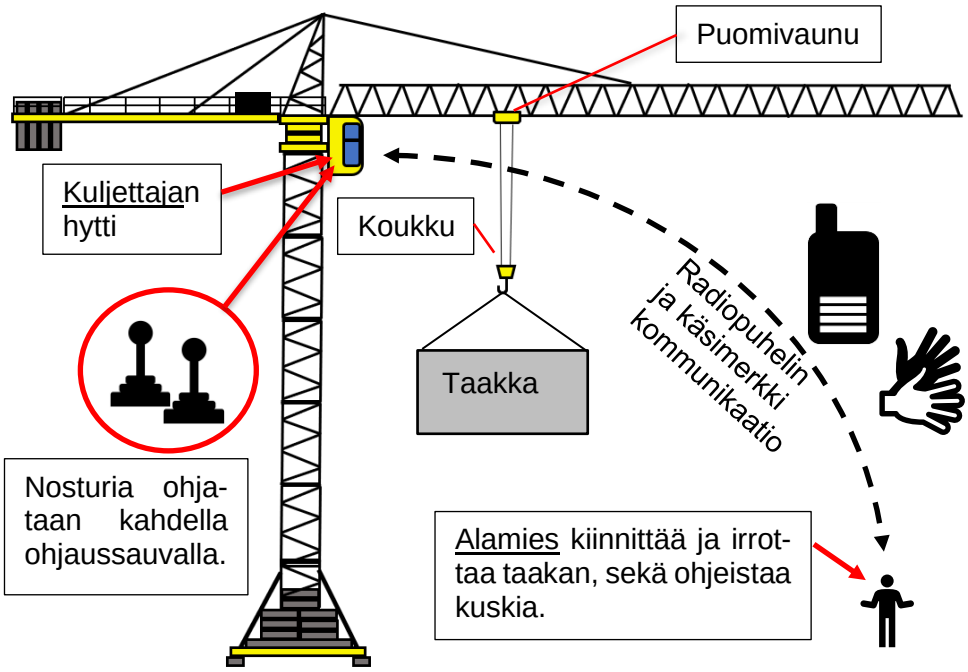
Tämän opinnäytetyön tavoitteena on luoda tähän kehitteillä olevaan automaatiojärjestelmään yksi osajärjestelmä. Kyseessä on turvalaite, joka valvoo torninosturin muun automaatiojärjestelmän toimintaa ja vian ilmentyessä pysäyttää nosturin. Laitetta halutaan käyttää myös manuaalisena pysäytysjärjestelmänä, ja pysäytyksen tulee tapahtua rauhallisesti. Nostureiden normaalit hätä-seis-pysäytysjärjestelmät lukitsevat nostureiden tuulijarrut aiheuttaen rajuja pysäytyksiä, jollaisia halutaan välttää. Opinnäytetyön on määrä toimia perustana automaatiojärjestelmän turvajärjestelmälle. Työ tehdään Trenox Oy:lle ja on osa Trenox Oy:n kehitysprojektia torninostureiden automatisoimiseksi.

Työn alussa käsitellään torninosturin ja automaatiojärjestelmän toimintaa sekä turvalaitteen roolia siinä ja määritellään vaatimukset laitteen toiminnalle. Sen jälkeen pohditaan, kuinka laitteen voi teknisesti toteuttaa ja tuodaan esille eri vaihtoehtoja hardwaren ja softwaren suhteen. Tämän jälkeen käsitellään työn käytännön etenemisvaiheita ja esitellään saadut käytännön tulokset. Viimeinen luku sisältää analyysia projektista ja sen tuloksista kokonaisuudessaan.

2 Teknistä taustatietoa

2.1 Perinteisen torninosturin toiminta

Nykyaikaisen torninosturin ohjaus on toteutettu kahdella ohjaussauvalla, jotka on sijoitettu nosturin ohjaamoon. Nosturin kuljettajalla voi olla nosturin varustelusta riippuen lisäksi apuvälineinä kameroita ja koukun korkeuden ilmaiseva näyttö. Noston yhteydessä mukana on aina myös niin kutsuttu alamies, joka kiinnittää ja irrottaa taakan, sekä antaa nosto-ohjeita kuskille. Viestintä kuljettajan ja alamiehen välillä tapahtuu radiopuhelimitse ja käsimerkein. Nosturin kuljettajalta vaaditaan aina nosturinkuljettajakoulutus. Kuvaan 1 on koottu torninosturilla operointiin liittyvät peruskäsitteet.



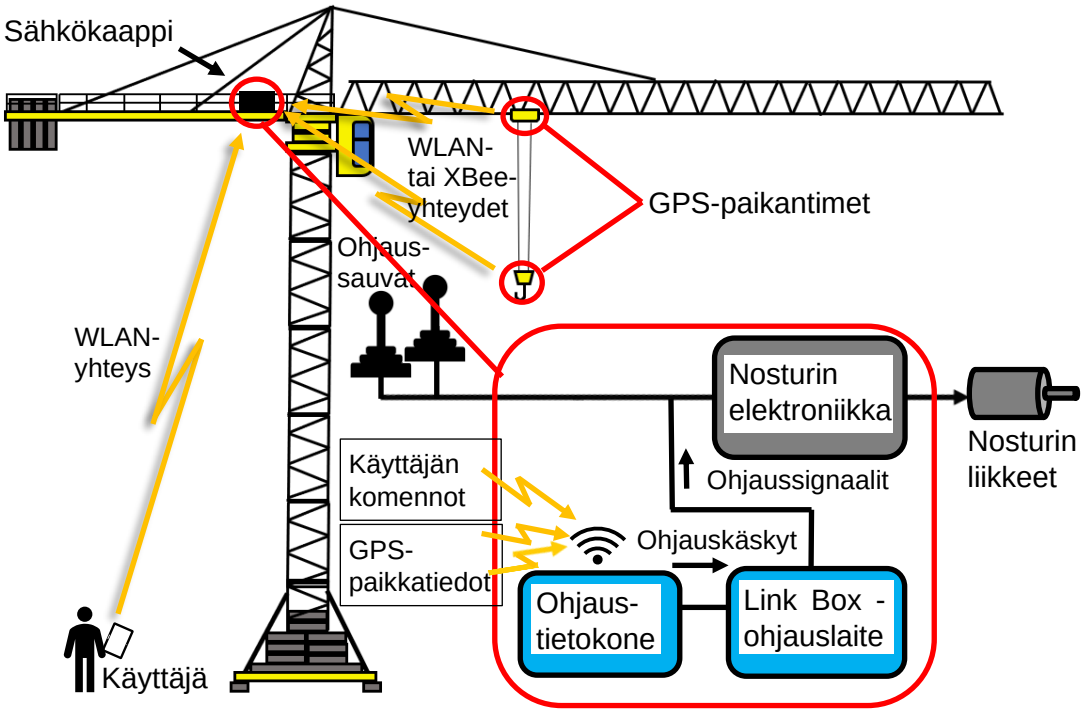
Kuva 1. Torninosturin peruskäsitteet.

Käytännössä kaikissa torninostureissa on nykyään myös kauko-ohjain. Kauko-ohjaimella voi tehdä pieniä nostoja maasta käsin, mikä helpottaa taakan maahan laskemista, kun kuskilla on huono näkyvyys ohjaamosta. Näkyvyys ohjaamosta onkin usein nostoja vaikeuttava tekijä, koska ohjaamo on yleensä useiden kymmenien metrien päässä taakasta ja työmailla on usein myös paljon näköesteitä, kuten rakennuksia ja työkoneita (1.)

2.2 Trenox Oy:n automaattinosturi

Automaattinosturin ajatuksena on automatisoida nosturikuljettajan osuus nosturin ohjauksesta, jolloin alamies kykenee suorittamaan koko noston yksin. Automaattinosturia on tarkoitus ohjata kannettavalla päätelaitteella, jolla alamies, eli kuka tahansa nosturinkäyttäjä, voi pyytää koukkua tulemaan luokseen. Päätelaite on GPS (Global Positioning System) -seurattu, joten käyttäjän ei tarvitse huolehtia sijaintinsa määrittelystä. Koukku saapuu päätelaitteen koordinaatteihin automaattisesti. Nosturia voi myös käskeä siirtymään kartalta valittuun pisteeseen. Saavuttaessaan halutun pisteen nosturi jättää koukun riippumaan muutaman metrin korkeuteen, josta nosturinkäyttäjä laskee sen maahan päätelaitteen kauko-ohjauksella.

Trenox Oy:n kehityksessä oleva nosturiautomaatiojärjestelmä on modulaarinen ja se on tarkoitus voida jälkiasentaa helposti mihin tahansa torninosturiin merkistä ja mallista riippumatta. Nosturin alkuperäiseen elektroniikkaan ei tarvitse tehdä mitään muutoksia, sillä järjestelmä asennetaan nosturin alkuperäisen elektroniikan rinnalle. Järjestelmän keskeiset laitteet ovat Link Box -ohjauslaite, ohjaustietokone, kaksi GPS-paikanninta ja päätelaite. Kuvassa 2 on esitetty automaattisen ohjausjärjestelmän toimintaperiaate ja kuinka sen eri laitteet ovat yhteydessä nosturiin ja toisiinsa.



Kuva 2. Automaattisen ohjausjärjestelmän toimintaperiaate.

Link Box -ohjauslaite toimii rajapintana nosturin automaatiojärjestelmän välillä. Sen ajatuksena on imitoida ohjaussauvojen tuottamia signaaleja releiden avulla ja se asennetaan sähköisesti rinnakkain alkuperäisten ohjaussauvojen kanssa. Link Box -ohjauslaite ei itsessään synnytä sähköisiä signaaleja, kuten ei ohjaussauvatkaan, vaan Link Box ainoastaan välittää lävitsensä nosturin ohjauselektroniikasta peräisin olevia signaaleja. Tämä tapahtuu yhdistelemällä ohjaussauvojen kaapeleiden johtimia releiden avulla. Ohjaussauvojen fyysiset liikkeet ovat kaikissa nostureissa samanlaiset, mutta käytetyt signaalityypit ja johdinyhdistelmät riippuvat kokonaan nosturin mallista ja jopa yksilöstä. Link Box -ohjauslaite ei suorita laskentaa, vaan liikeradat ja ohjauskäskyt lasketaan ohjaustietokoneessa. Link Boxin osana on vain toteuttaa ohjauskäskyt, jotka sille lähetetään sarjakaapelia pitkin.

Ohjaustietokoneena toimii IPC (Industrial PC) eli teollisuustietokone. Sen tehtävä on laskea liikeradat koukun ja puomivaunun päälle sijoitettujen GPS-paikantimien antamien paikkatietojen perusteella. Tietokone laskee nostolle tavoiteliikeradan ennen liikkeen aloittamista. Noston edetessä se ohjaa liikettä reaaliaikaisesti mahdollisimman lähelle tavoitetta.

GPS-paikantimista on olemassa kahta eri versiota. Ensimmäinen, vanha versio, toimii langattoman lähiverkon eli WLAN (Wireless Local Area Network) -verkon kautta, jossa tiedonsiirto tapahtuu TCP (Transfer Control Protocol) -protokollan mukaan. TCP-protokolla tarjoaa monipuolisen tiedonsiirron virheenkorjausjärjestelmän ja sitä on helppo hyödyntää valmiiden koodikirjastojen ansiosta. WLAN-verkossa on kuitenkin todettu esiintyvän viiveitä niin paljon, että sen käytöstä GPS-paikantimien kanssa halutaan luopua.

Tämän takia Trenox Oy:ssä ollaan saman aikaisesti opinnäytetyön kanssa kehittämässä uutta versiota GPS-paikantimesta, joka käyttää tiedonsiirrossa WLAN-verkon sijaan XBee-radioita. XBee-radiot ovat teollisuuden tarpeisiin suunnattuja radiolähetin/vastaanottimia ja ne käyttävät luvasta vapaita radiotaajuuksia. Toimivia koekappaleita on jo valmistettu ja niiden käyttöön siirrytään lähitulevaisuudessa. Kommunikaatio XBee-verkossa tapahtuu Trenoxissa kehitetyn protokollan mukaan.

2.3 Turvajärjestelmät opinnäytetyön aloitushetkellä

Opinnäytetyön aloitushetkellä turvajärjestelminä käytettiin nosturin alkuperäistä hätä-seis-järjestelmää sekä virtakytkintä, jolla pystyi kytkemään sähköt pois Link Box -ohjainlaitteelta. Hätä-seis-järjestelmän pystyy laukaisemaan nosturin ohjaamosta tai kauko-ohjaimelta. Kyseessä on siis nosturin alkuperäinen kauko-ohjain, joka ei liity automaatiojärjestelmään mitenkään.

Hätä-seis-järjestelmän laukaiseminen pysäyttää kaikki nosturin moottorit ja kytkee jarrut välittömästi päälle. Tällä tavoin nosturin liike pysähtyy lyhyimmässä mahdollisessa ajassa. Koska jarrut kytkeytyvät päälle riippumatta nosturin vauhdista, hätä-seis-pysäytykset voivat aiheuttaa nosturin rakenteisiin äärimmäisiä rasituksia. Toistuvasti käytettynä hätäpysäytykset voivat aiheuttaa jopa rakennevaurioita nosturiin.

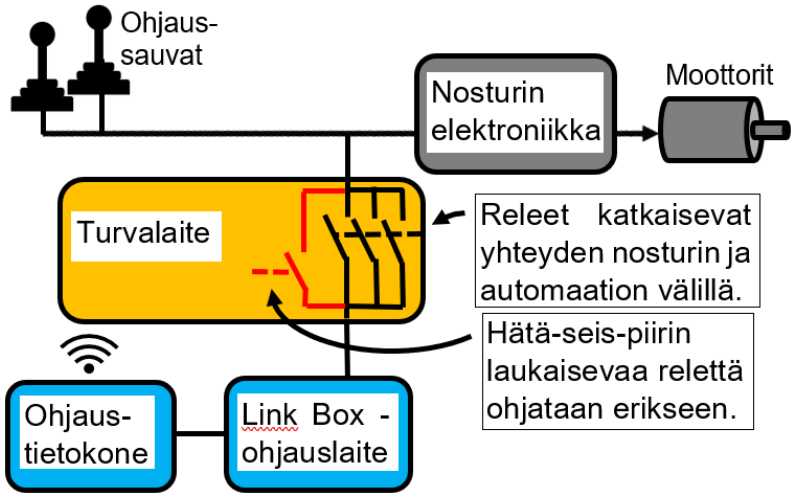
Sen takia automaatiojärjestelmän testeissä käytettyyn nosturiin oli tehty järjestely, jolla Link Box -ohjauslaitteen sähköt pystyi kytkemään pois. Sähköjen pois kytkeminen palauttaa Link Boxin releet auki-tilaan, jolloin ohjaussignaaleja ei enää välity nosturille ja liike pysähtyy luonnostaan kitkan vaikutuksesta. Tällöin nosturin rakenteisiin ei kohdistu normaalia suurempia rasituksia ja siksi Link Boxin sähköjen pois kytkemistä käytettiin ensisijaisena turvajärjestelmänä.

Virtakytkin oli sijoitettu nosturin juurelle, mistä siltä oli vedetty sähköjohdot ylös nosturin takapuomilla sijaitsevaan sähkökaappiin. Tässä sähkökaapissa sijaitsivat Link Box -ohjainlaite, ohjaustietokone ja nosturin ohjauselektroniikka. Tämä turvajärjestely oli yksinkertainen toteuttaa ja riittävä automaatiojärjestelmän kehitystyötä aloittaessa. Lopullisessa tuotteessa tällainen järjestely olisi kuitenkin riittämätön, sillä virtakytkimen käyttäminen vaatii fyysistä pääsyä nosturin luokse, mikä saattaa olla todellisissa työmaaolosuhteissa täysin mahdotonta riittävän nopeasti. Nosturin käyttäjä saattaa olla jopa jonkin rakennuksen katolla, mistä hänellä kestää minuutteja päästä nosturin juurelle. Järjestelyn toiminta myös nojasi täysin käyttäjän tarkkaavaisuuteen ja ymmärrykseen kääntää kytkimestä ajoissa. Lisäksi kytkimen johtojen vetäminen maasta nosturin puomille jokaisen nosturin pystytyskerran yhteydessä olisi epäkäytännöllistä.

2.4 Lisättävä turvalaite

Lisättävän turvalaitteen ajatuksena oli toimia itsenäisesti ilman käyttäjän huomiota, havaita mahdolliset viat ja pysäyttää nosturi turvallisesti ennen kuin vaaratilannetta ehtii syntyä. Lisäksi laitetta haluttiin toimivan myös manuaalisena pysäytyslaitteena, jonka käyttäjä pystyy laukaisemaan helposti huomatessaan vaaratilanteen. Laitteen ensisijainen tarkoitus oli toimia automaatiojärjestelmän prototyyppiversion tukena. Lisäksi sitä aiottiin käyttää pohjana automaatiojärjestelmän lopullisen kaupallisen version turvajärjestelmälle.

Yksinkertainen ja varmin tapa pysäyttää nosturin liike turvallisesti on katkaista Link Box -ohjauslaitteen ja nosturin elektroniikan välinen sähköinen yhteys. Tämä vastaa samaa kuin ohjaussauvat päästettäisiin vapaaksi, jolloin nosturin liike pysähtyisi liukuen. Vaikutus on siis sama kuin sähköjen katkaisulla Link Boxilta, mutta tässä automaatiojärjestelmä tulee lisäksi täysin eristetyksi nosturinohjauksesta. Keino on siis varmempi ja takaa ettei mikään automaatiojärjestelmän vika pääse vaikuttamaan nosturiin. Kuvassa 3 on esitetty turvalaitteen sijoitus nosturin elektroniikan ja automaatiojärjestelmän välissä. Releitä on kuvassa vähemmän kuin todellisuudessa.



Kuva 3. Turvalaite katkaisee yhteyden nosturin elektroniikan ja automaatiojärjestelmän välillä.

Lisättävällä turvalaitteella ei haluttu tehdä hätä-seis-pysäytyksiä, mutta hätä-seis-järjestelmän käyttö haluttiin mahdolliseksi optiona tulevaisuuden käyttöä varten. Kuten jo mainittua hätäpysäytykset aiheuttavat äärimmäistä rasitusta nosturin rakenteisiin, jonka takia niitä ei kannata käyttää muulloin kuin tilanteissa, joissa siten pystytään ehkäisemään henkilö- tai materiaalivahinkojen syntymistä. Lisättävä turvalaite oli suunnattu laitteistovikojen havaitsemista varten, eikä sen tehtävänä ollut toimia törmäyksenestojärjestelmänä, minkä takia turvalaitteen ei ole tarpeen käyttää hätä-seis-pysäytystä. Törmäykseneston kehitys oli tämän projektin laajuuden ulkopuolella.

Koska turvalaitteella oli tarkoitus pystyä pysäyttämään nosturi myös manuaalisesti, täytyi sille kehittää myös jokin etäkäyttöliittymä, joka olisi nosturinkäyttäjän helposti saatavilla. Käyttöliittymän kautta täytyi pystyä pysäyttämään nosturi pehmeästi katkaisemalla Link Boxin ja nosturin välinen yhteys, tai laukaisemalla hätä-seis-järjestelmä. Tästä pehmeästä nosturin pysäytystavasta käytetään tästedes nimitystä turvapysäytys.

3 Alustavaihtoehtojen kartoitus

3.1 Alustan vaatimukset

Projektin alkuun kartoitettiin laitteita, joilla pystyttäisiin toteuttamaan turvalaitteen halutut toiminallisuudet. Laitteen täytyi toimia monien erilaisten protokollien (USB, IP, XBee) kanssa ja käsitellä dataa, joka osin ei ole minkään standardin mukaista. Siksi alustaksi kannatti valita laite, jota pystyy ohjelmoimaan vapaasti ja monipuolisesti. Toisin sanoen laite kannatti perustaa jollekin tietokonepohjaiselle alustalle, eikä perinteiselle ohjelmoitavalle logiikalle. Tietokoneella tarkoitetaan tässä tapauksessa laitetta, joka käyttää jotain tietokoneen käyttöjärjestelmää, esimerkiksi Linuxia tai Windowsia.

Ohjelmoitavia logiikoita käytetään yleisesti nostureiden ohjausjärjestelmissä, minkä takia sellaisen käyttäminen olisi vaikuttanut luontevalta ratkaisulta. Toiminnaltaan ne kuitenkin vastaavat enemmänkin mikrokontrolloreita kuin tietokoneita. Logiikoita voi ohjelmoida vain niitä varten tehdyillä ohjelmistoilla ja lisäksi logiikoiden prosessoritehot ovat melko rajallisia (2, s.3.) Tarvittava laskentateho ei ollut tiedossa projektin alussa, joten helpointa ja varminta oli hankkia laite, jossa on tehoa ainakin enemmän kuin keskimääräisessä logiikassa. Lisäksi ohjelmistolisenssien hankkimisesta logiikoiden ohjelmointia varten olisi koitunut lisäkuluja, logiikoiden muutoinkin korkeiden hintojen ohella.

Logiikan vahvuutena olisi kuitenkin ollut I/O (Input Output) -portit, joita on helppo käyttää ja asentaa lisää. Logiikkojen I/O-portit myös kestävät melko suuria virtoja. Digitaalilähtöjen virran kesto on yleensä 0,5 A ja analogisignaalit ovat 0…10 V tai 4…20 mA. Digitaalisignaalien jännite on yleensä 0…24 VDC, mutta myös 230 VAC signaaleja käytetään. Tällaisista I/O-porteista saatetaan käyttää myös nimitystä teollisuus-I/O. (2, s.9.)

Tietokone puolella oli nykyään saatavilla monia laitteita, joissa on GPIO (General Purpose Input Output) -portteina tunnettuja I/O-portteja. GPIO-porttien signaalit kulkevat kuitenkin tietokoneiden USB- ja sarjaporteista tutulla jännitetasolla eli 0…5 V tai 0…3,3 V signaaleina (3). GPIO-porttien käyttämien Link Box -ohjainlaitteen välittäminen ohjaussignaalien havaitsemiseen olisi välttämättä vaatinut jonkinlaisen välipiirin, joka olisi

muuntanut signaalit GPIO-porteille sopiviksi. Logiikoiden I/O-porteilla sen sijaan mahdollisesti pystyi havaitsemaan joitain signaaleja ilman välipiiriä ja releiden ohjaus pystyttiin joka tapauksessa toteuttamaan suoraan digitaalisilta I/O-porteilta.

GPIO-portteja varten oli saatavilla erillisiä relekortteja, joiden releitä pystyy ohjaamaan suoraan GPIO:lla. Nämä kortit olivat kuitenkin hyvin halpoja harrastekäyttöön tarkoitettuja, eikä niiden toimintavarmuudesta ole takeita. Siksi jokin muu ratkaisu on turvallisempi ja vastuullisempi. (4; 5.)

3.2 Eri alustavaihtoehdot

Tässä luvussa listataan parhaat löytyneet vaihtoehdot turvalaitteen alustaksi. Jokainen vaihtoehto on esitelty lyhyesti ja luvun lopussa on taulukko, johon on vielä koottu avainominaisuuksia kaikista vaihtoehdoista. Perinteisiä ohjelmoitavia logiikoita ei otettu vaihtoehtojen joukkoon niiden pienempien prosessoritehojen, rajoittuneemman ohjelmoinnin ja korkeampien hintojen takia.

Raspberry Pi

Ensimmäinen vaihtoehto, Raspberry Pi, on harrastelijoiden keskuudessa suuren suosion saavuttanut Linux-pohjainen yhden piirin tietokone (Single Board Computer). Suosion taustalla ovat laitteen laajat sovellus mahdollisuudet ja huokea hinta (6.) Laitteen saa tilattua netistä 30–50 eurolla veroineen riippuen mallista (7).

Raspberry Pi 3 model B+ olisi ollut huokea ja käyttökelpoinen vaihtoehto turvalaitetta varten. Releiden ohjaus ja signaalien havaitseminen GPIO-porteilla on luultavasti hieman monimutkaisempaa toteuttaa, kuin teollisuus-I/O-porteilla, mutta varmasti toteutettavissa. Toisaalta Raspberry Pi:lle löytyi myös paljon ohjeita Internetistä, laajan harrastajajoukon ansiosta. Laitteen luotettavuus herätti pientä epäilystä. Raspberryjen käyttämät SD-muistikortit tunnetusti saattavat korruptoitua ja kuluttajakäyttöön suunnatussa elektroniikassa saatetaan muutenkin käyttää vähempilaatuisia komponentteja. (8; 9.)

Digi ConnectCore

Toinen vaihtoehto oli Digi ConnectCore, joka oli Raspberry Pi:tä vastaava yhdenpiirin tietokone. ConnectCore oli suunnattu nimenomaan teollisuuskäyttöön ja se oli Raspberryä hieman hintavampi. Siksi sen saattoi arvioida olevan korkealaatuisempi ainakin komponenttien suhteen. Harmillisesti ConnectCoressa tosin oli vähemmän prosessoritehoa. (10; 11.)

Revolution Pi

Revolution Pi kuului kanssa Raspberry Pi:tä vastaavien laitteiden kategoriaan, kuten sen nimestäkin jo pystyi päätellä. Merkittävin ero Raspberry Pi:hin verrattuna siinä oli, se ettei siinä ollut GPIO-portteja. Sen sijaan siihen pystyi liittämään teollisuus-I/O-moduuleita. Myös Revolution Pi oli suunnattu teollisuuskäyttöön ja sitä markkinoitiin teollisuustietokoneena (IPC). Se käytti SD-kortin sijaan sisäistä eMMC-muistia, mikä oli luotettavuuden kannalta erittäin hyvä. Lisäksi Revolution Pi:ssä oli DIN-kiskoon asennettava kotelointi. (12.)

Siemens IOT-2040

Siemens IOT-2040 vaikutti mielenkiintoiselta vaihtoehdolta, sillä se oli uudenlainen laite perinteikkäältä logiikkavalmistajalta. Sitä markkinoitiin nimensä mukaisesti teollisuuden IoT (Internet Of Things) -sovelluksiin. IOT-2040:n Arduino-malliset GPIO-portit sai muutettua teollisuus-I/O-porteiksi lisämoduulilla, jossa oli viisi digitaalista sisääntuloa, kaksi digitaalista ulostuloa ja kaksi analogista sisääntuloa. I/O:ta ei harmillisesti kuitenkaan pysty lisäämään tästä yhtään enempää. Lisäksi laite häviää prosessoritehossa selkeästi muille ja on silti kalliimmasta päästä. (13; 14.)

Beckhoffin CX-sarja

Beckhoffin CX-sarjaan kuului monia eritehoisia ja eri liitännöillä varustettuja laitteita. Ne vastasivat näistä vaihtoehdoista eniten Revolution Pi:tä ja olivat ominaisuuksiltaan parempiakin. Teollisuus-I/O:ta ja liitäntöjä pystyi lisäämään Beckhoffin logiikoiden kanssa

käytettävillä moduuleilla. Prosessoriksi sai jopa hyvin tehokkaan 2,1 GHz:n 12-ydinprosessorin. Muista vaihtoehdoista poiketen CX-sarjan laitteet käyttivät Windows Embedded -käyttöjärjestelmää. Mikä oli oikeastaan negatiivinen ominaisuus, koska sille löytyi vähemmän ohjeita Internetistä, eikä minulla ollut siitä lainkaan käyttökokemusta. Beckhoffin tuotteet olivat hyvin hintavia, eikä hintatietoja ollut helposti saatavilla Internetistä, mikä sekin jo kertonee korkeasta hintaluokasta. Muiden laitteiden kohdalla hinnat löytyivät helposti netistä. Taulukkoon 1. on vielä koottu kaikkien mainittujen vaihtoehtojen avainominaisuuksia. (15; 16).

Taulukko 1. Alusta vaihtoehdot ja niiden keskeisimpiä ominaisuuksia.

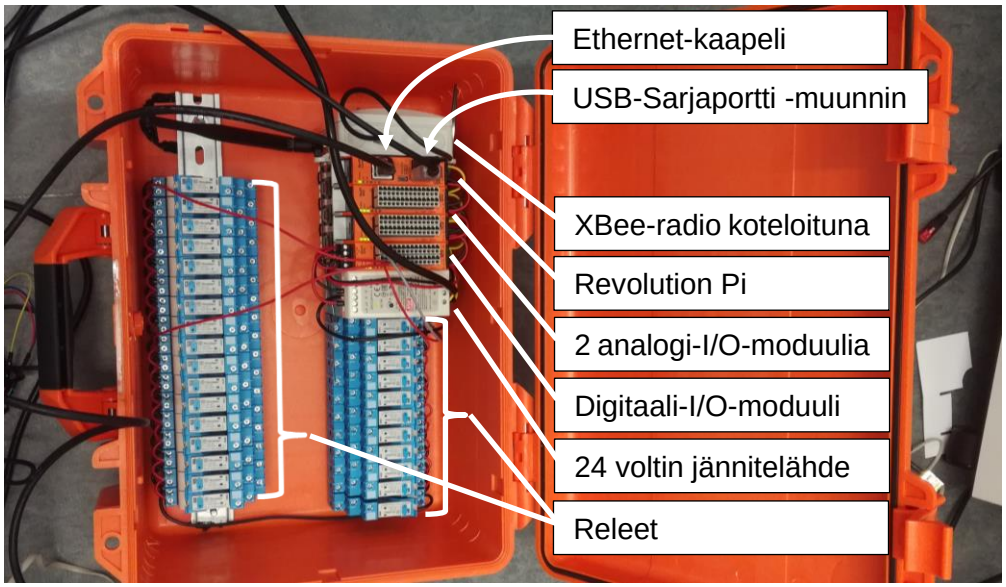
Laite	**Liitännät**	**Käyttöjär.**	**Prosessori**	**Hinta**
Raspberry Pi 3 Model B+	GPIO, 4 x USB, Ethernet	Linux Raspbian	1,4 GHz Cortex-A53	<50 € sis. ALV.
Digi ConnectCore 6UL SBC Express	GPIO, 2 x USB, Ethernet	Linux Yocto	528 MHz, Cortex-A7	Noin 100 € sis. tulli
Revolution Pi Core 3	Teollisuus I/O:ta lisättävissä, 2 x USB, Ethernet	Linux Raspbian	1,2GHz, Cortex-A53	204 € veroton
Siemens IOT-2040	1 x USB, 2 x Ethernet, 2 x sarja-portti, MPCIe, Arduino-mallinen GPIO (14 x DIO + 6 x AI)	Linux Yocto	400 Mhz Intel Quark	225 € veroton
Beckhoff CX-series	Teollisuus I/O:ta lisättävissä, USB, Ethernet, sarja-portti	Windows Embedded	0,5…2,1 GHz riippuen mallista	600…>1000 € hinta tiedot vaikeasti saatavilla

4 Toteutunut fyysinen kokoonpano

4.1 Valitut komponentit

Toteutuneeseen fyysiseen kokoonpanoon valittiin Revolution Pi -teollisuustietokone. Revolution Pihin päädyttiin, koska sen fyysinen kokoonpano vaikutti luotettavalta muistiratkaisun suhteen kuin yleisestikin. Toinen hyvä syy valintaan oli mahdollisuus liittää siihen teollisuus-I/O:ta. Ainoastaan monta kertaa kalliimmassa Beckhoffissa oli samanlainen mahdollisuus. Lisäksi Revolution Pissä oli tehokas prosessori ja DIN-kiskoon sopiva kotelointi vaikutti käytännölliseltä ratkaisulta.

Turvalaitteessa on yhteensä 28 relettä, joilla saadaan aikaan hätä-seis ja turva-seis-pysäytykset. Releet ovat Finderin DIN-kiskoon asennettavia teollisuusreleitä. Releet ovat ryhmitelty kahteen eri ryhmään. Ensimmäinen ryhmä on hätä-seis-piirin laukaisemista varten ja siinä on vain yksi rele. Toisessa ryhmässä on kaikki loput releet ja ne ovat turva-seis-pysäytystä varten, eli niillä katkaistaan Link Box -ohjauslaitteen yhteys nosturiin. Kaikki laitteen osat saavat sähkön Mean Wellin 24 voltin jännitelähteeltä, joka on kytketty verkkovirtaan. Revolution Pi:n USB-porteille on kiinnitetty XBee-radio ja USB-sarjaportti -muunnin. XBee-radiota tarvitaan vain, jos GPS-paikantimet käyttävät XBee-verkkoa. USB-sarjaportin kautta turvalaite saa tietoon ohjaustietokoneen Link Boxille antamat ohjauskäskyt. Kuvasta 4 näkee kaikki komponentit käytännössä asennettuna. Kuvassa turvalaite on ennen sen asentamista ja siitä vielä puuttuu releiden kontaktori puolen johdotus sekä johtojen läpiviennit.



Kuva 4. Turvalaitteen kokoonpano ennen kontaktoripuolen johdotuksia.

Kuvassa 4 oranssi laite salkun sisällä on Revolution Pi -teollisuustietokone, johon on liitetty yksi digitaalinen I/O-moduuli ja kaksi analogista I/O-moduulia. Digitaalimoduulissa on 14 sisääntuloa ja 14 ulostuloa, ja kummassakin analogimoduulissa neljä sisääntuloa ja kaksi ulostuloa. Digitaaliulostulolla ohjataan releitä ja digitaali- sekä analogisisääntulot ovat tarkoitettu Link Box -ohjainlaitteen välittämien ohjaussignaalien havaitsemista varten. Kotelointina on säänkestävä muovisalkku, jonka pohjaan on pultattu kaksi DIN-kiskoa, joihin osat on kiinnitetty. Kaikki osat ovat DIN-kiskoon sopivia, lukuun ottamatta XBee-radiota, jolla on oma pieni kotelo, joka kiinnitettiin nippusiteillä paikoilleen.

Alkuperäinen suunnitelma oli asentaa turvalaite nosturin sähkökaapin sisään. Automaatiojärjestelmän testaamiseen käytetyn Betox 150 -torninosturin sähkökaapissa tila oli kuitenkin todella vähissä, jolloin todettiin helpommaksi ja turvallisemmaksi asentaa turvalaite erikseen kaapin ulkopuolelle ja tarkoitukseen valittiin säänkestäväsalkku. Salkku on suunniteltu kestämään ulkoilmaympäristöä ja on täysin vedenpitävä. Johtojen ulos saattamiseksi salkusta tehtiin kaksi vedenpitävää läpivientiä kuvan 5 mukaisilla holkkitiivisteillä. Sähkökaapin sisään johdot vedettiin kaapin alakulmassa olleesta aukosta.

Kuva 5. Holkkitiiviste (17).

Umpinaisessa salkussa kuitenkin huolenaiheena oli jäähdytyksen riittävyys. Siksi salkun kanssa tehtiin lämpötilatestejä ennen sen lopullista valintaa. Testeissä turvalaitetta käytettiin salkkuineen sisätiloissa käytössä vaadittavalla prosessorin käyttöasteella ja releet vedossa. Prosessorilämpötila vakiintui noin 65 °C:sen, kun Ilman salkkua sama lämpö vakiintui noin 60 °C:sen. Ero ei siis ollut kovinkaan suuri. Myöhemmin toteutetussa testissä turvalaite asetettiin salkkuineen 24 °C helteellä suoraan auringon paisteeseen. Tällöin korkein mitattu prosessorin lämpötila ylsi 86 °C:sen. Salkun sisälämpötilaksi mitattiin 50 °C:ta. Salkun kylkiin olisi voinut porata tuuletusreikiä, jotka olisi sitten suojattu sateelta ja tuulelta. Se olisi kuitenkin merkittävästi riskeerannut veden pitävyyden ja tuottanut lisätyötä. Revolution Pi:n valmistajan nettisivuilta kävi ilmi, että prosessorin lämpötila yltäisi normaalistikin 80 asteeseen kovan rasituksen alla, joten tilanteen nähtiin olevan siedettävä eikä lisätuuletusta tarvittavan. (18.)

4.2 Link Box -ohjainlaitteen tarkkailusta

Toiveena oli välittää Link Box -ohjauslaitteen signaalit turvalaitteelle, jotta Link Boxin toimintaa oltaisiin pystytty tarkastelemaan vertaamalla ohjaustietokoneen antamia ohjauskäskyjä havaittuihin signaaleihin. Ratkaisun haluttiin olevan universaali eli sopivan kaikkiin mahdollisiin asennuskohteisiin. Link Boxin periaatteellinen toiminta perustuu laitteen kykyyn yhdistää mitkä tahansa sen 28 portista toisiinsa, mikä tehdään monimutkaisen relejärjestelyn kautta. Ainoa tapa todentaa mitkä, porteista ovat yhteydessä toisiinsa, oli porttien kautta kulkevien sähköisten signaalien havaitseminen.

Vaatimus kaikkialle sopivasta ratkaisusta teki tästä kuitenkin erittäin haastavaa, koska eri nosturit käyttävät sattumanvaraisesti eri teollisuuden signaalityyppejä. Havaittavat signaalit saattaisivat siis olla esimerkiksi 10 V:n tasavirtaa tai sitten 230 V:n vaihtovirtaa. Teoriassa ratkaisu olisi voitu saavuttaa jonkinlaisella virtamittauksella tai signaalinmuuntopiirillä. Käytännössä tällaisen virtamittarin tai muuntopiirin itse rakentaminen kuitenkin

olisi vaatinut erittäin syvällistä elektroniikan tuntemusta, eikä valmiita ratkaisuja ollut olemassa, kuten selvitystyön myötä kävi selväksi.

Rakenteilla olleeseen Link Box -ohjauslaitteen uuteen versioon oli tulossa näiden porttien lisäksi kuusi analogista -10…10 V:n I/O-porttia. Nämä analogiportit pystyttäisiin yhdistämään suoraan Revolution Pi -teollisuustietokoneen analogisisääntuloille, mutta Link Boxin uuden version valmistumisen viivästyttyä sitä ei päästy tekemään. Alkuperäinen suunnitelma oli asentaa turvalaite ja uusi Link Box yhdessä.

Nosturikohtaisen ratkaisun kehittäminen ohjaussignaalien tarkasteluun olisi ollut huomattavasti yksinkertaisempaa, mutta sellaiseen ratkaisuun tyytyminen lisäisi erittäin paljon järjestelmän uuteen kohteeseen asentamiseen vaadittavan suunnittelun määrää ja tekisi siitä paljon vaikeamman asentaa. Automaatiojärjestelmän kantava idea oli olla kaikkialle sopiva ja helppo asenteinen, eikä Link Box -ohjauslaitteen tarkkailu ollut edes välttämätöntä järjestelmän toiminnan kannalta. Siksi Link Boxin porttien tarkkailusta päätettiin luopua tämän projektin osalta.

5 Ohjelmointikielen valinta

Niin paljon kuin laite tarvitsee fyysiset komponenttinsa on yhtä tärkeää, että sillä on tarpeen mukainen ohjelmointi. Ennen ohjelmoinnin aloittamista otettiin selvää mikä olisi paras ohjelmointikieli turvalaitteen toimintojen toteuttamisen kannalta. Trenox ei asettanut rajoituksia kielen valinnan suhteen.

5.1 Logiikkaohjelmointikielet

Vaikkei projektissa käytettykään ohjelmoitavaa logiikkaa, olisi siinä voinut hyödyntää logiikkaohjelmointikieliä ja -ohjelmistoja. Logiikkaohjelmia pystyy käyttämään myös niin sanotuissa Soft-PLC:ssä eli tietokoneella toimivassa virtuaalilogiikassa. Logiikkaohjelmointikielet ovat luotu toimimaan turvalaitteen kaltaisissa tauotta päällä olevissa järjestelmissä. Lisäksi minulla oli aiempaa kokemusta logiikkaohjelmoinnista, jonka takia oli luonnollista harkita, pystyisikö sitä hyödyntämään. Revolution Pi:lle oli saatavilla Codesys- ja Logi.Cads 3 -logiikkaohjelmointiohjelmistot.

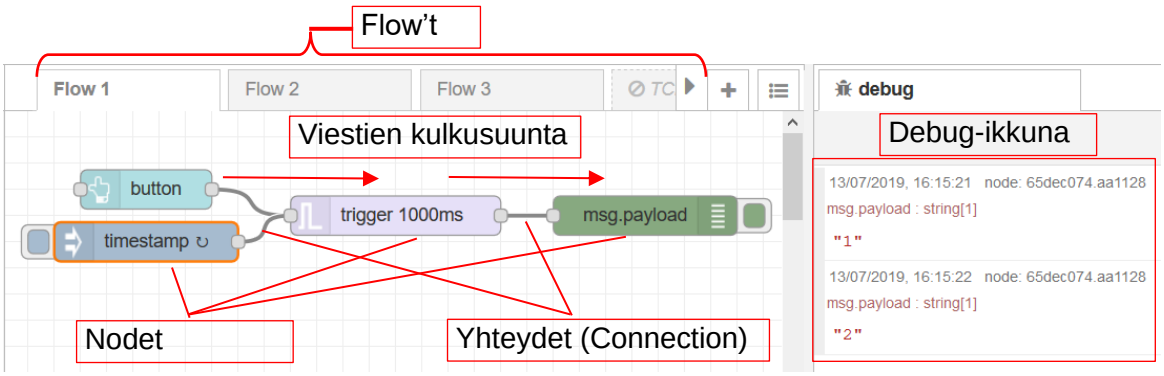
Logiikkaohjelmointikielien heikkoutena oli kuitenkin niiden rajoitteisuus. Ne ovat kehitetty spesifisti logiikoiden ohjelmointiin, minkä takia ne ovat ensisijaisesti tarkoitettu vain loogisten ohjaustoimintojen toteuttamiseen (19). Lisäksi molemmista ohjelmistoista olisi koitunut muutamien satojen eurojen lisenssikulut, mikä oli negatiivista (12; 20). Logiikkaohjelmointikielen käyttäminen olisi tuonut merkittäviä lisäkuluja eikä sellaisissa nähty olevan tässä projektissa mitään parempia ominaisuuksia perinteisiin kieliin nähden, kuten esimerkiksi Pythoniin tai C-kieleen, joita pystyy käyttämään ilmaiseksi. Näistä syistä opinnäytetyössä ei otettu käyttöön logiikkaohjelmointikieliä.

5.2 Node-RED-ohjelmointityökalu

Revolution Pi -teollisuustietokoneen Raspbian-käyttöjärjestelmässä oli tehtaalla valmiiksi asennettu Node-RED -ohjelmointityökalu (programming tool). Node-RED on suunnattu IoT (Internet Of Things) -järjestelmien ohjelmointiin. Node-RED:in ohjelmointitapa perustuu Flow-Based Programming (FBP) -ohjelmontikonseptiin ja se on käyttää Node.js'aa ajonaikaisena ympäristönä eli runtime'na. Node-RED perustuu Javascript-ohjelmointikieleen ja sitä ylläpitää OpenJS-säätiö. (21.)

Koska Node-RED-ohjelmointityökalu oli valmiiksi asennettu ja ilmainen, ei ollut syytä olla tutustumatta siihen. Node-RED on graafinen ohjelmointityökalu, jossa ohjelmointi perustuu niin kutsuttuihin nodeihin joiden välille luodaan yhteyksiä (Connection). Jokaisella nodella on jokin funktio, jonka se suorittaa saadessaan viestin (message) sisääntulolleen (input) tai saadessaan jonkin toisenlaisen herätteen, kuten esimerkiksi käyttäjäsyötteen käyttöliittymältä. Funktion suorittamisen jälkeen node voi lähettää edelleen viestin ulostuloltaan (output), josta viesti menee yhteyttä pitkin seuraavalle nodelle.

Ohjelmoinnin perusajatus on kuljettaa viestiä koodin läpi nodelta toiselle. Koodia pystyy jaottelemaan pienemmiksi kokonaisuuksiksi niin kutsuttujen flow'ien avulla, jotka ovat kuin välilehtiä tai kuin Function Blockit logiikkaohjelmoinnissa. Jokaiseen flow'iin pystyy tekemään haluamansa määrän koodia ja viestejä pystyy linkkaamaan niiden välillä. Kuvassa 6 on yksinkertainen Node-RED-koodiesimerkki, johon on merkitty koodin peruskäsitteet.



Kuva 6. Esimerkki Node-RED -koodista.

Kuvan 6 värilliset suorakaiteet ovat nodeja, joilla jokaisella on oma funktionsa, ja niiden välillä olevat viivat yhteyksiä, joita pitkin viestit kulkevat. Koodia luetaan vasemmalta oikealle. Koodissa timestamp-node on konfiguroitu lähettämään viestinä aikaleima kymmenen sekunnin välein. Trigger-noden funktio on lähettää viestin ”1” ja sekunnin jälkeen vielä viestin ”2”. Msg.payload-node näyttää kaikki sille saapuneet viestit debug-ikkunassa, mutta ei lähetä mitään viestiä eteenpäin. Debug-ikkunassa voi nähdä, kuinka trigger-node on aktivoitunut sen saatua viesti timestamp-nodelta. Button-node aktivoituu eli lähettää viestin, jos käyttöliittymässä olevaa nappia painetaan. Käyttöliittymä ei näy kuvassa.

5.3 Arvio Node-RED-ohjelmointityökalusta

Ensi vaikutelmana Node-RED-ohjelmointityökalu vaikutti helppokäyttöiseltä ja nopealta välineeltä luoda ohjelmia, ja lisäksi graafinen ohjelmointitapa oli luonnollinen logiikoista tuttua FBD (Function Block Diagram) -ohjelmointia aiemmin tehneelle. Siksi Node-REDiä päätettiin lähteä kokeilemaan opinnäytetyössä. Lopullinen päätös ohjelmointikielistä ja -työkaluista tehtäisiin vasta myöhemmin, kun ohjelmiston toteuttamisen asettamat vaatimukset olisivat paremmin kartoitetut ja toteutuskeinoista olisi kerätty enemmän kokemusta.

Opinnäytetyön aikana kerätyn kokemuksen perusteella voi todeta, että Node-RED-ohjelmointityökalu on varma toiminen eli se käynnistyy varmasti tietokoneen käynnistyessä ja alkaa suorittaa sille tallennettua koodia. Node-REDillä pystyy luomaan nopeasti monenlaisia ohjelmia vähäiselläkin kokemuksella, koodin debuggaus on helppoa ja koodia

on helppo muokata myös jälkikäteen. Miinuspuolena ovat rajalliset koodausmahdollisuudet, sillä kaikkiin tarkoituksiin ei välttämättä ole valmiita nodeja, jolloin joutuu tukeutumaan perinteiseen koodaukseen. Tästä syystä turvalaitteen Node-RED-koodissa on käytetty myös Python-funktioita joissain kohdin. Toisena miinuksena Node-REDin ohjelmointikäyttöliittymä on kömpelöhkö koodimäärän kasvaessa. Hyvää kuitenkin on koodin asynkroninen suorittaminen, jonka ansiosta koodin eri osat tulevat suoritetuiksi ilman huolta. Lisäksi koodiin on helppo palata jälkeen päin ja ohjelmointityökalun käyttö on helppo omaksua. Node-RED-ohjelmointityökalu paras vahvuus voisi olla pienehkössä IoT-projektissa, jossa on käytössä monia eri alustoja.

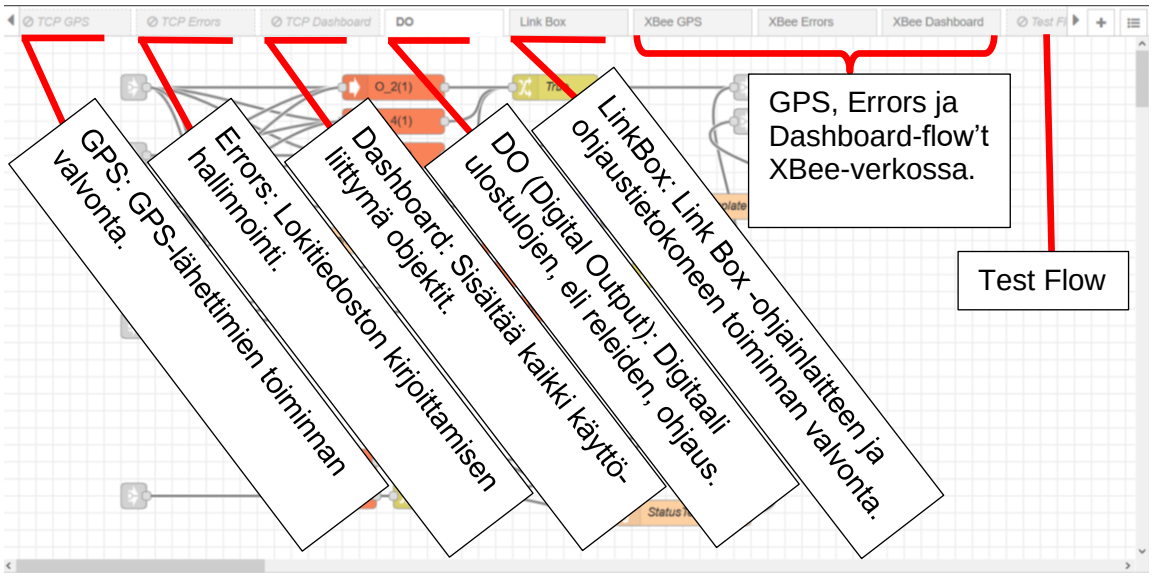
6 Turvalaitteen koodi

Lopullinen ohjelmointityö opinnäytetyössä tehtiin Node-RED-ohjelmointityökalulla sekä Python-ohjelmointikielellä. Node-RED valittiin käyttöön, koska se oli varmatoiminen ja helppokäyttöinen. Osa toiminnoista oli helpompi toteuttaa Pythonilla, minkä takia myös sitä käytettiin. Pääohjelmisto kuitenkin perustuu Node-RED-koodiin ja Python-koodi on vain shellejä, joita pääohjelmisto käyttää.

Kirjoitetun koodin tehtävä on valvoa ohjaustietokoneen ja Link Boxin, sekä GPS-lähettimien toimintaa ja luoda turvalaitteelle käyttöliittymä. Seuraavissa luvuissa käydään läpi koodin toiminta pääpiirteissään. Kaikki koodin flow’t on nähtävissä liitteessä 1. Joissain luvuissa on myös yksinkertaistettuja kuvia flow’eista havainnollistamassa niiden toimintaa.

6.1 Node-RED-koodin flow’t

Koodi on jaoteltu viiteen osaan flow’ien avulla. Jokaiseen osaan on pyritty keskittämään tietyn tyyppiset toiminnallisuudet. Flow’t ovat nimeltään GPS, Errors, Dashboard, DO ja Link Box. Jokainen nimi kuvastaa jollain tapaa flow’n tarkoitusta. Kuvassa 7 kaikki flow’t ovat esitelty lyhyesti. Kuva on otettu ohjelmointikäyttöliittymän näkymästä.



Kuva 7. Turvalaitteen flow't.

Tärkeänä huomiona, kuten myös kuvasta 7 voidaan huomata, todellisuudessa koodi sisältää yhdeksän eri flow'ta viiden sijaan, koska flow'eista GPS, Errors ja Dashboard on eri versiot XBee- ja TCP-yhteyksillä toimimiseen. Yhdeksäntenä flow'na on Test Flow, jota on käytetty vain testauksessa ja normaalissa käytössä se on aina pois käytöstä (disabled). Turvalaitteen toimiessa käytössä (enabled) on aina vain viisi flow'ta, riippuen käytössä olevista GPS-lähettimistä.

Flow'ien kahdentaminen oli käytännöllinen ratkaisu, koska niitä pystyy helposti poistamaan käytöstä (disable) ja ottamaan käyttöön (enable). Käytöstä poistetut flow't vastaavat samaa kuin ne olisi poistettu koodista kokonaan. Näin ei synny mitään vaaraa siitä, että koodin eri versiot synnyttäisivät konflikteja keskenään.

6.2 Link Box -flow

Link Box -flow tehtävä on valvoa Link Box -ohjauslaitteen ja ohjaustietokoneen toimintaa. Ohjaustietokoneen toiminnasta tärkeimpänä valvotaan, että se lähettää koko ajan ohjauskäskyjä Link Box -ohjauslaitteelle. Ohjauskäskyt lähetetään sarjakaapelia pitkin ja niitä lähetetään jatkuvasti myös silloin kun nosturi ei liiku.

Koodissa ohjauskäskyt otetaan vastaan serialRead.py -nimisellä Python-shellillä. Shellissä saapuvista käskyistä tarkastetaan, että niistä löytyy aloitus- ja lopetusmerkit eli C- ja E-kirjain, ja että niiden pituus on oikea. Ohjauskäskyn sisältönä on yhdeksän pilkuilla erotettua numeroa, joista jokainen vastaa yhtä nosturin liikkeistä. Jokaisesta saapuneesta oikean muotoisesta ohjauskäskystä Python-shell lähettää kuittauksen Node-RED -koodin puolelle. Viestin lähettäminen Pythonista Node-REDiin tapahtuu yksinkertaisesti print() -komennolla. Mikäli Pyhton-shell ei lähetä kuittauksia kolmeen sekuntiin, Node-RED -koodissa oleva ajastin laukaisee turvapysäytyksen.

Link Box -ohjainlaitteen toimintaa oli ajatus tarkkailla vertaamalla ohjainlaitteen antamia signaaleja ohjaustietokoneen lähettämiin ohjauskäskyihin. Jos signaalit ja ohjauskäsky täsmäisivät, voitaisiin Link Boxin todeta toimivan virheettömästi. Signaalit oli ajatus havaita Revolution Pi -teollisuustietokoneen digitaali- ja analogisisääntuloilla. Kuten jo aieminkin mainittua digitaalisignaalien osalta kaikkiin nosturimalleihin sopivan universaalin ratkaisun kehittäminen oli kuitenkin niin monimutkaista, että sen kehittämisestä luovuttiin tämän projektin osalta. Analogisignaaleja sen sijaa pystyttäisiin vastaanottamaan turvalaitteella, mutta harmillisesti Link Boxin uuden version valmistuminen viivästyi ja turvalaite asennettiin vanhan version kanssa, jossa ei vielä ole analogiportteja.

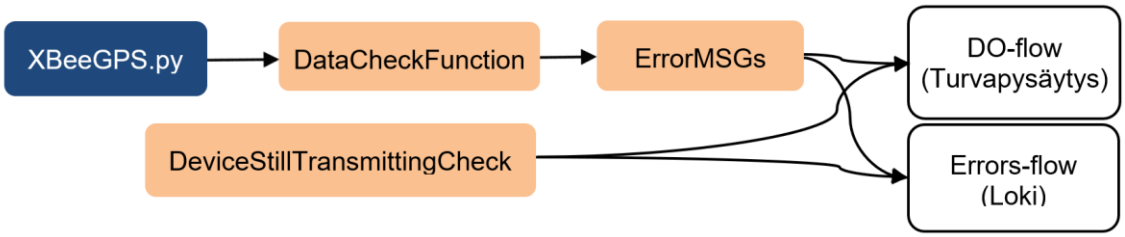
Tästä huolimatta sisääntulojen lukemista ja vertaamista ohjauskäskyihin kuitenkin testattiin Python-koodissa ja serialRead.py -tiedostoon jätettiin aihioksi koodia, jonka pohjalta ominaisuuden kehittämistä pystytään jatkamaan myöhemmin. Revolution Pin I/O-porttien tiloja luetaan piTest-funktiolla, joka palauttaa halutun portin tilatiedon digitaaliporteilla Boolen-arvona ja analogiporteilla 16-bittisenä arvona eli arvona 0:sta 65535:een.

6.3 GPS-flow

GPS-flow’n tarkoitus on huolehtia GPS-lähettimien lähettämän datan vastaanottamisesta ja tarkastamisesta. GPS-flow’sta on olemassa eri versiot XBee- ja TCP-yhteyksillä toimiville GPS-lähettimille. Kummankin flow’n toiminta on selitetty erikseen seuraavaksi.

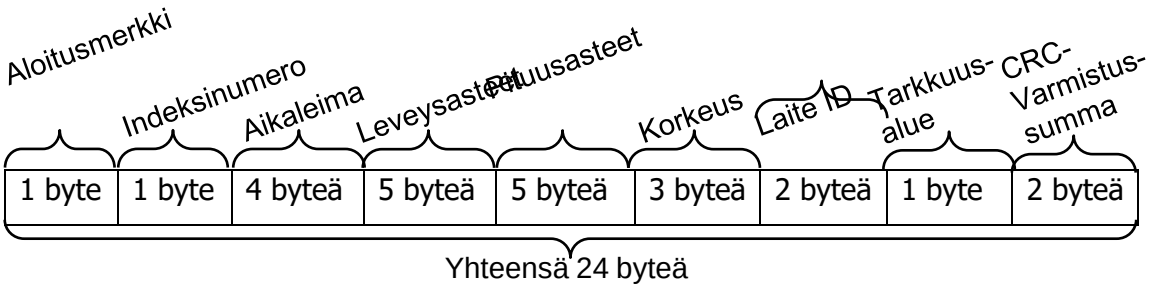
XBee GPS -flow

XBee GPS -flow'ta käytetään, kun käytössä on XBee-radioyhteydet. Kuvassa 8 on esitetty sen toiminnan kannalta tärkeimmät osat. Flow on nähtävillä kokonaisuudessaan liitteen 1 kuvassa 1.



Kuva 8. XBee GPS -flow yksinkertaistettuna.

Flow'n toiminta on seuraavanlainen. GPS-data otetaan vastaan Python-funktiolla XBee-radion kautta, joka vastaa koodissa sarjaporttia. Funktio on nimetty XBeeGPS.py'ksi ja kuvassa 8 sininen laatikko kuvastaa sitä. Data saapuu 24-tavun pituisina paketteina, jotka otetaan sisään tavu kerrallaan aina aloitusmerkistä alkaen. Kuvassa 9 on esitelty paketin rakenne. Saapuneen paketin eheys varmistetaan CRC (Cyclical Redundancy Check) -varmistussummalla. Lisäksi tarkastetaan, kuinka pitkä aika edelliseen pakettiin samalta lähettäjältä on. Mikäli aikaväli on yli kolme (3) sekuntia, virhettä merkkaavalle bitille annetaan arvoksi True. Jos CRC-laskenta oli virheetön, Python-funktio lähettää paketin eteenpäin Node-RED -koodin puolelle. Tiedon lähettäminen Python-funktiosta Node-REDiin onnistuu yksinkertaisesti print()-komennolla.



Kuva 9. GPS-lähettimen datapaketti XBee-verkossa.

Paketti jatkaa matkaa viestinä Node-REDissä DataCheckFunction-funktioon. Funktio on kirjoitettu JavaScript-ohjelmointikielellä. Node-RED-koodiin voi luoda myös nodeja, joihin

pystyy kirjoittaa JavaScriptillä oman funktionsa. Skriptissä paketista tarkastetaan seuraavat asiat:

- Paketin ikä on alle 3 sekuntia.
- GPS:än tarkkuusalue on paras tai toisiksi paras.
- Indeksi-numero on kasvanut 1–4 arvolla.
- Python shellin lisäämän virhebitin arvo on False, eli pakettien välinen aika on alle 3 sekuntia.

Mikäli tarkastukset eivät tuota virheitä, pakettia ei lähetetä enää pidemmälle. Siitä tallennetaan välimuistiin laitetunnus ja vastaanottamishetken aikaleima, ja sitten paketti unohdetaan.

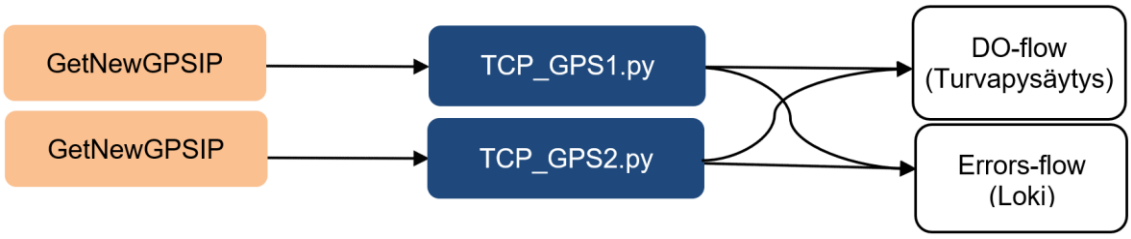
Kaikki paketit ovat indeksoitu perättäisillä numeroilla. Indeksin kasvussa sallitaan kuitenkin muutaman numeron välistä jääminen, koska paketteja jää jatkuvasti saapumatta yhteyshäiriöiden takia, eikä muutaman paketin pois jääminen ei vielä haittaa ohjausjärjestelmän toimintaa.

Virheiden löytyessä paketti lähetetään edelleen ErrorMSGs-funktioon, joka sekin on JavaScriptillä kirjoitettu. Funktiossa pakettiin lisätään virheen mukainen virhesanoma lokiin kirjoittamista varten, jonka jälkeen paketti lähetetään viestinä DO-flow'hun sekä Errors-flow'hun. Viestin lähettäminen DO-flow'hun tarkoittaa käytännössä turvapysäytyksen laukaisemista, ja Errors-flow'n tehtävä on tehdä tapahtumasta lokimerkintä. Näiden flow'ien toimintaan palataan myöhemmin.

Kuvaan 8 merkitty DeviceStillTransmittingCheck on edellä selitetystä koodista erillään toimiva JavaScript-funktio. Sen tehtävä on valvoa, ettei kumpikaan GPS-lähettimistä lakkaa lähettämästä GPS-dataa. Se tarkastaa sekunnin välein aiemman koodin osan välimuistiin tallentamat aikaleimat ja laitetunnukset. Yli viisi (5) sekuntia vanha aikaleiman löytyessä funktio lähettää virheviestin DO- ja Errors-flow'hun eli toisin sanoen laukaisee turvapysäytyksen ja kirjoittaa loki merkinnän.

TCP GPS -flow

TCP GPS -flow on käytössä, kun GPS-lähettimet käyttävät TCP-yhteyksiä. Flow'n toiminnan kannalta keskeisimmät nodet on kuvattu kuvassa 10. Flow sisältää kaksi toiminnaltaan samanlaista puolta, joista kumpikin huolehtii yhden TCP-yhteyden muodostuksesta ja ylläpidosta. Seuraava koodin selitys pätee siis molempiin puoliin.



Kuva 10. TCP GPS -flow yksinkertaistettuna.

Turvalaitteen käynnistyessä tai käyttäjän painaessa reset-nappia käyttöliittymästä flow'n koodi suoritetaan. Ensimmäisenä GetNewGPSIP-funktio lukee muistiin tallennetun IP-osoitteen, jonka sinne on tallennettu käyttöliittymän kautta. Funktio lähettää osoitteen TCP_GPS1.py tai TCP_GPS2.py-Python-funktioon, riippuen kummasta puolesta puhutaan. Käyttöliittymä esitellään seuraavassa luvussa.

Python-funktio ottaa IP-osoitteen vastaan argumenttina ja yrittää muodostaa TCP-yhteyden kyseiseen osoitteeseen. Jos yhteyden muodostus onnistuu, saapuvasta datapaketista tarkastetaan GPS-lähettimen tarkkuusalue. Mikäli yhteyttä ei voida muodostaa, se katkeaa tai tarkkuusalue on huono, Python-funktio lähettää virheviestin Node-RED-koodin puolelle. Virheviesti menee DO- ja Errors-flow'hun, joka aiheuttaa turvapysäyksen ja loki merkinnän. Yhteydelle on asetettu kolmen (3) sekunnin aikakatkaisu (timeout), joten yli kolmen sekunnin viive tai yhteyden katkeaminen aiheuttaa virheviestin. Python-funktiossa käytettiin Socket -nimistä moduulia TCP-yhteyden hallitsemiseen.

TCP-yhteyksiä käytettäessä säästyttiin kirjoittamasta itse monia yhteyteen liittyviä tarkastuksia toisin kuin XBee-yhteyksillä, koska TCP-yhteyksiä varten oli olemassa valmis moduuli. TCP-protokolla kuuluu TCP/IP-protokollaperheeseen, jota käytetään lähes kaikessa verkkoliikenteessä. TCP on niin sanottu yhteydellinen (connection oriented) pro-

tokolla, jossa kaikki lähetyt paketit kuitataan ja lähetetään tarvittaessa uudestaan. Paketeille määritellään myöskin maksimikoko, ne ovat indeksoituja, ne järjestetään vastaanottopäässä alkuperäiseen järjestykseen ja jokaisen paketin eheys tarkistetaan CRC-varmistussumman avulla. Käyttämällä koodissa TCP/IP-yhteyksille tarkoitettua moduulia kaikki nämä toiminnot sai huomaamatta toimimaan muutamalla rivillä koodia. (22, s.298.)

XBee-radioyhteydet sen sijaan toimivat yhteydettömästi, joten ne eivät itsessään sisällä mitään virheentarkastusta. Sen takia lähetettävään dataan itsessään on liitetty tietoa virheentarkastusta varten, kuten esimerkiksi indeksinumerot ja CRC-varmistussumma. Näiden tietojen pohjalta tehdyt tarkastukset joudutaan kuitenkin kirjoittamaan koodiin itse. Yhteydettömän lähetyksen etuna on kuitenkin yksinkertaisempi ja nopeampi tiedonsiirto.

6.4 Dashboard-flow ja käyttöliittymä

Dashboard-flow sisältää turvalaitteen käyttöliittymään liittyvän koodin. Node-RED-ohjelmointityökalu sisältää myös ominaisuuden käyttöliittymän luomiseen. Turvalaitteen käyttöliittymä on luotu tällä ominaisuudella. Myös Dashboard-flow'sta on kaksi eri versiota, koska käyttöliittymän sisältö on hieman erilainen riippuen siitä, käytetäänkö GPS-lähettimissä XBee- vai TCP-yhteyksiä.

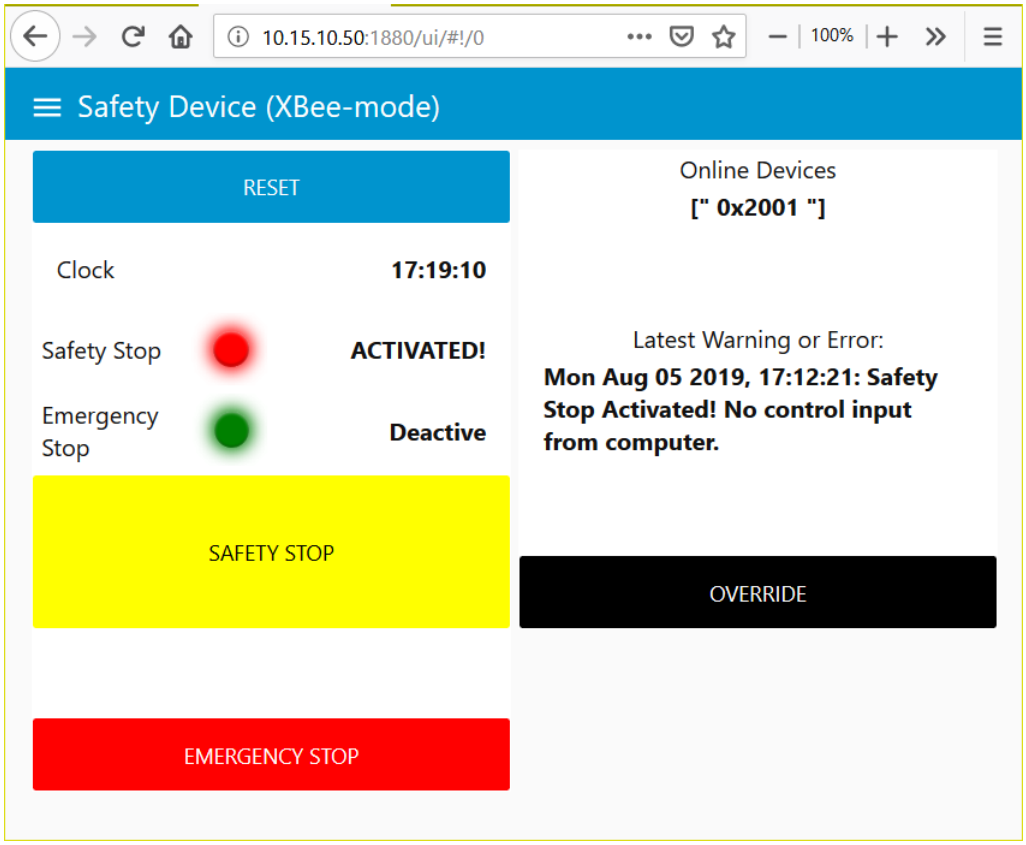
Node-RED-ohjelmointityökalulla luodut käyttöliittymät toimivat web-selainpohjaisesti. Käyttöliittymäkin luodaan Node-REDissä vetämällä koodiin nodeja. Uusi käyttöliittymäobjekti, kuten painike tai teksti-ikkuna, tehdään vetämällä koodiin sitä vastaava node. Objektin ulkoasua ja asettelua voi sitten muuttaa noden säädöistä.

Kaikki käyttöliittymän nodet kannatti sijoittaa yhteen flow'hun, jotta käyttöliittymän sisältö pysyi hallinnassa, eikä käyttöliittymän nodet hukkuneet muun koodin sekaan. Käyttöliittymän saavuttaa samassa lähiverkossa olevalla laitteella, syöttämällä web-selaimen osoitekenttään sen laitteen IP-osoite, jolla käyttöliittymä on. IP-osoitteen perään täytyy lisäksi kirjoittaa porttinumeroksi :1880/ui.

Seuraavissa kappaleissa käydään turvalaitteen käyttöliittymän kummankin version sisältö läpi. Dashboard-flow'n versioita ei käydä sen enempää läpi, koska ne pääasiassa

vain sisältävät passiivisesti käyttöliittymän nodet. Flow't ovat kuitenkin nähtävissä liitteen 1 kuvissa 3 ja 8.

Kuvassa 11 on nähtävissä turvalaitteen käyttöliittymä, kun käytössä on XBee-radioyhteydet. Käyttöliittymä on jaettu oikean ja vasemman puoleiseen palstaan. Vasemmanpuoleinen palsta on aina samanlainen, mutta oikeanpuoleisessa on eroja käyttöliittymän versioiden välillä.

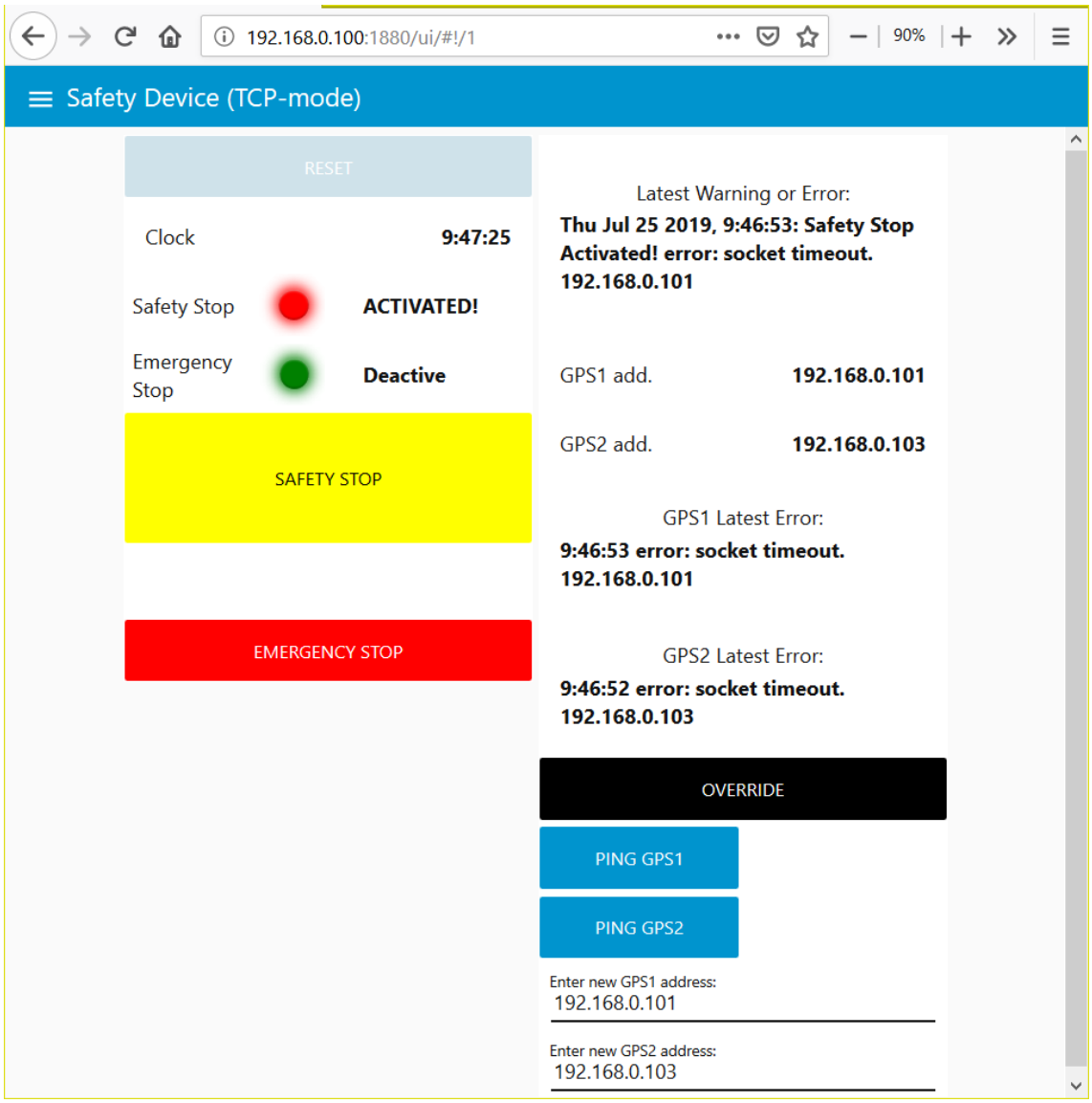


Kuva 11. Turvalaitteen käyttöliittymä XBee-verkossa toimivien GPS-lähettimien kanssa.

Vasemmanpuoleiseen palstaan on sijoitettu Safety stop- ja Emergency stop -painikkeet, joilla käyttäjä voi manuaalisesti laukaista turva- tai hätäpysäytyksen. Niiden yläpuolella on lamput, jotka kertovat käyttäjälle, mikäli turva- tai hätäpysäytys on lauennut. Pysäytyksen lauetessa lampun väri vaihtuu vihreästä punaiseksi. Värien merkitystä selventämässä on myös teksti, joka vaihtuu lampun värin mukana. Vasemman palstan ylimpänä on reset-painike, jolla kuitataan lauennut turva- ja hätäpysäytys.

Oikean puoleisesta palstasta löytyy override-painike, jota painamalla turvalaitteen automaattiset vianhavaitsemistoiminnot voi ohittaa. Tällöin turvalaite toimii ainoastaan manuaalisena pysäytyslaitteena. Samasta palstasta löytyy myös Latest Warning or Error -ikkuna, josta näkyy viimeinen lokiin kirjoitettu viesti. Tämän on käyttäjälle erittäin hyödyllistä, koska näin käyttäjä näkee avaamatta lokitiedostoa mistä syystä mahdollinen nosturinpysäytys on tapahtunut. Online Devices -ikkuna näyttää toiminnassa olevien GPS-lähettimien laite-ID:t. Kuvassa 11 toiminnassa on vain yksi laite.

Kuvasta 12 näkee, millainen käyttöliittymä on, kun GPS-lähettimet käyttävät TCP-yhteyksiä. Vasemmanpuoleinen palsta on edelleen samanlainen, mutta oikean puoleisessa palstassa Online Devices -ikkuna korvattu monilla muilla objekteilla.



Kuva 12. Turvalaitteen käyttöliittymä TCP-yhteyksillä toimivien GPS-lähettimien kanssa.

Tässä käymme läpi ylhäältä alas oikean puoleisen palstan uudet objektit. GPS1 add- ja GPS2 add. -rivit kertovat IP-osoitteet, joihin sillä hetkellä TCP-yhteydet GPS-datan vastaanottamiseksi on muodostettu tai yritetään muodostaa. GPS1 Latest Error- ja GPS2 Latest Error -ikkunoissa näkyy viimeisimmät virheviestit kumpaakin TCP-yhteyttä koskien. Ping GPS1- ja Ping GPS2 -napeilla pystyy tehdä ping-testin GPS add. -rivien osoitteille. Näiden toimintojen tarkoitus on kertoa käyttäjälle TCP-yhteyksien tila ja auttaa mahdollisen ongelman paikallistamisessa. GPS-paikantimet lopettavat datan lähettämisen myös silloin kun ne menettävät GPS-signaalin, joten lähetyksen katketessa vika voi

olla TCP-yhteydessä, GPS-signaaleissa tai itse paikantimessa. Oikean palstan alimmaisina olevien Enter new GPS1 address- ja Enter new GPS2 address -rivien kautta voi syöttää uudet IP-osoitteet GPS-datan vastaanottamiseksi. Yhteys uusiin IP-osoitteisiin muodostetaan heti kun reset-painiketta painetaan.

6.5 DO-flow

Releiden ohjaus, jolla turva- tai hätäpysäytykset tehdään, keskitettiin koodissa DO-flow'hun. Releitä ohjataan Revolution Pin digitaalisilla ulostuloilla, joiden tiloja hallitaan Node-RED:issä nodella, joka on olemassa juuri sitä varten. Nimi DO viittaa sanoihin Digital Output. Kun jossain flow'ssa halutaan tehdä pysäytys, pysäytysviesti linkataan DO-flow'hun. Näin vältyttiin turhalta koodin uudelleen kirjoittamiselta jokaisessa flow'ssa ja koodi pysyi siistimpänä. Aina kun DO-flow'ssa vaihdetaan ulostulon tilaa, siitä lähetetään viesti Dashboard-flow'hun, jonka perusteella käyttöliittymän lamppujen värit vaihdetaan.

Myös override-ominaisuus eli automaattisen vianhavaitsemisen ohitus toteutettiin DO-flow'ssa. Flowssa oleva node tarkkailee override-napin painamista ja estää automaattisten vianhavaitsemistoimintojen lähettämät pysäytysviestien pääsemisen lävitsensä, jos nappia on painettu. Reset-napin painaminen resetoi toiminnon.

6.6 Errors-flow

Errors-flow'n tehtävänä on kirjoittaa virheviestit lokitiedostoon. Lokiin merkitään kaikki automaattisesti laukaistut nosturin pysäytykset, käsin laukaistut nosturin pysäytykset, kuittaukset (reset), turvalaitteen ohitukset (override) ja turvalaitteen käynnistymiset. Kaikki lokiin kirjatut tapahtumat eivät siis ole pelkkiä virheitä. Lokiin kirjoitetaan päivämäärä, aika, toiminto ja syy toiminnolle. Esimerkkikoodissa 1 on esimerkki virheviestistä.

```
Mon Aug 05 2019, 16:57:55: Safety Stop Activated! error: Safety Stop But-
ton pressed
```

Esimerkkikoodi 1. Esimerkki lokimerkinnästä.

Lokitiedosto on tekstimuotoinen ja se sijaitsee Revolution Pin työpöydällä. Jokainen viesti kirjoitetaan omalle riville. Virheviestit lähetetään Errors-flow'hun toisista flow'eista. Osa viesteistä sisältää jo tullessaan jonkinlaisen sanoman, mutta kaikkiin viesteihin kuitenkin lisätään Errors-flow'ssa lisää tietoa. Esimerkiksi päivämäärä ja kellonaika lisätään aina. Kirjoitettu virheviesti lähetetään myös Dashboard-flow'hun, jonka kautta se tulee näkyviin käyttäjälle.

Errors-flow sisältää myös koodin, joka tarkastaa viiden sekunnin välein Revolution Pin prosessorin lämmön. Jos lämpötila ylittää ylikuumenemisrajan 80 °C, lokiin kirjoitetaan merkintä ylikuumentumisesta. Lämpötilan palautumisesta normaaliksi tehdään myös merkintä.

7 Testaus ja työn tulokset

7.1 Testaus ja asentaminen

Opinnäytetyössä valmistunut laite rakennettiin käyttövalmiiksi toimistolla käytännössä kokonaan. Ainostaan releiden kontaktoripuolen johdotus ja kotelon läpiviennit tehtiin asennuspaikalla nosturin luona ammattinosturiasentajan toimesta. Suurin osa rakentamiseen liittyneestä testaamisesta oli koodin toimivuuden testaamista. Muu testaaminen sisälsi fyysisten komponenttien oikean asennuksen ja eheyden varmistamisen ja kotelointiin liittyneet lämpötilatestit. Lisäksi laitteen asennuksen yhteydessä tehtiin lopputesti, jolla varmistettiin asennuksen onnistuminen ja osoitettiin laitteen toimivan myös todellisessa käyttötilanteessa.

Lopputestissä turvapysäytyksen laukaisua kokeiltiin samalla kun nosturia ohjattiin eri suuntiin, lisäksi hätä-seis ja turvalaitteen ohitus-toimintoa kokeiltiin, ja kaikkien todettiin toimivan halutulla tavalla. Myös GPS-lähettimien ja ohjaustietokoneen päällä olo ja pois päältä olo havaittiin, kuten oli haluttu.

Lopputestissä havaittiin, että nosturin ohjauksen virtasignaalin johtimen katkaisu kytki jarrut päälle, mikä aiheuttaisi samanlaisen äkkinäisen pysäytyksen kuin hätäpysäytyksessä. Ongelma pystyttiin kuitenkin ratkaisemaan helposti siirtämällä yksi releistä turva-

seis-piiristä hätä-seis-piiriin. Virtasignaali on nosturien ohjausjärjestelmissä käytetty signaali, joka määrittää onko nosturi päällä vai sammutettu. Sammutettuna nosturi ei liiku, vaikka sen ohjaimiin koskettaisiinkin.

Koodia testattiin sitä mukaa, kun siihen tehtiin uusia toimintoja. Testauksessa käytettiin oikeita GPS-lähettimiä, joita kytkettiin pois päältä tai joiden signaalia häirittiin peittämällä antenni. Automaatiojärjestelmän ohjaustietokoneelta sarjakaapelia pitkin tulevia ohjauskäskyjä simuloitiin kytkemällä sarjakaapelin toinenkin pää Revolution Pi -tietokoneyksikköön ja lähettämällä tekaistuja ohjauskäskyjä tämän takaisinkytkennän kautta. Testauksessa käytettiin suurelta osin myös koodin sisäistä simulointia. Revolution Pin I/O-moduulin sisääntuloporttien käyttöä testattiin yhdistämällä niitä ulostuloportteihin.

7.2 Pohdintaa projektista

Opinnäytetyön tuloksena saatiin laite, jolla Trenox Oy:n automaattiohjausjärjestelmä pystytään kytkemään irti nosturista ja näin pysäyttämään nosturi rauhallisesti ja ilman nosturiin kohdistuvaa rasitusta. Opinnäytetyöprojektin tavoite oli kehittää erillinen laite valvomaan automaattiohjausjärjestelmän osia vikaantumisen varalta ja pysäyttämään nosturi pehmeästi, mikäli vika havaitaan. Laitetta haluttiin toimivan myös manuaalisena pysäytysjärjestelmänä. Projekti oli luonteeltaan tuotekehitysprojekti. Laitetta varten kirjoitettiin myös englanninkielinen dokumentaatio Trenox Oy:n käyttöön.

Laitteesta tahdottiin nimenomaan erillinen, eikä esimerkiksi automaatiojärjestelmän ohjaustietokoneeseen integroitu, jotta sen toimimisesta voitaisiin olla varmoja riippumatta pääjärjestelmästä. Hyöty korostuu etenkin nyt kun järjestelmä vielä kokee muutoksia kehitystyön aikana. Ajatuksena myös oli luoda perusta erilliselle turvajärjestelmälle, jota voitaisiin kehittää lisää muun kehitystyön päästessä eteenpäin.

Vikojen havaitsemisessa GPS-lähettimien kanssa onnistuttiin hyvin. Lähettimien päällä olo havaitaan ja lisäksi lähetyksen laatu tarkastetaan. Vanhojen TCP-GPS-lähettimien ratkaisut tulivat TCP-protokollan mukana, mutta XBee-GPS-lähettimillä tarkastustoiminnot täytyi koodata kokonaan itse. XBee-GPS-lähettimien tunnistaminen ja tarkkailu saatiin lisäksi toimimaan täysin automaattisesti.

Link Box -ohjauslaitteen tilan tarkkailuun sen sijaan ei löydetty ratkaisua. Link Boxin seuraavan version analogisignaaleja olisi pystytty havaitsemaan, mutta tätäkään ei päästy käytännössä kokeilemaan seuraavan version valmistumisen viipyessä. Link Box -ohjauslaitteen havainnointi tiedettiin jo projektin alkaessa hyvin vaikeaksi haasteeksi. Sen selvittämiseksi vaadittaisiin paljon laajempaa suunnittelutyötä ja syvempää elektroniikan tuntemusta. Paras tapa toteuttaa havainnointi olisi alusta alkaen kehittää oma piiri sitä varten. Piiriä varten kannattaisi mahdollisesti tehdä muokkauksia Link Boxiin tai jopa integroida se Link Boxiin. Jää kuitenkin arvioitavaksi, kannattaako tähän työhön ryhtyä, sillä kyseessä ei kuitenkaan ole järjestelmän kannalta välttämätön toiminto ja työn määrä olisi hyvin suuri.

Ohjaustietokoneen ohjauskomennot saatiin onnistuneesti välitettyä turvalaitteelle, ja niiden pohjalta pystytään havaitsemaan, onko tietokone päällä ja lähettääkö se oikeanmuotoisia ohjauskäskyjä. Jos Link Boxin ohjaussignaalit olisi saatu tuotua turvalaitteelle, olisi ohjauskomentojen tarkkailusta saatu enemmän hyötyä irti, kun niiden sisältöä olisi voitu verrata Link Boxin toteuttamaan toimintaan.

Manuaalisiin toimintoihin oltiin tyytyväisiä. Nosturin pystyy pysäyttämään etänä niin turvapysäytyksellä kuin hätäpysäytyksellä, ja turvalaitteen ohitustoiminnolla automaattiset pysäytystoiminnot voidaan ohittaa, jolloin laite toimii pelkkänä manuaalisena pysäytyslaitteena. Käyttöliittymä on helposti saavutettavissa käytännössä millä tahansa laitteella nosturin WLAN-verkon alueella, ja se tarjoaa käyttäjälle turvalaitteen lisäksi reaaliaikaisesti tietoa GPS-lähettimien ja ohjaustietokoneen tilasta.

Turvalaite tarjoaa myös alustan turvallisuustoimintojen kehityksen jatkolle. Laite on vapaasti ohjelmoitavissa, ja siinä on melko paljon laskentatehoa. Jotta turvalaitteesta saataisiin enemmän irti sen potentiaalista erillisenä automaatiojärjestelmää valvovana laitteen, täytyisi sille kuitenkin saada tuotua paljon enemmän automaatiojärjestelmän toiminnasta kertovaa tietoa.

Vielä parempi olisi saada tietoa järjestelmän ulkopuolelta nosturin liikkeestä ja ympäristöstä, jolloin laite voisi toimia todellisena nosturin toimintaa valvovana turvajärjestelmänä. Tässä kuitenkin kohdataan automaatiojärjestelmän kehitystyön rajat, sillä järjestelmälle ei vielä ole mitään laitteistoa ympäristön havaitsemiseksi. Automaatio siis ajaa

tällä hetkellä vielä sokkona ympäristössään. Ennen tällaisten pidemmälle vietyjen turvatoimintojen kehittelyä onkin järkevintä odottaa pääjärjestelmän valmiiksituloa, jotta turvajärjestelyjen tarvetta ja toteutus mahdollisuuksia voi arvioida paremmin.

8 Yhteenveto

Tiivistetysti opinnäytetyön tuloksena saatu laite estää automaattista ohjausjärjestelmää liikuttamasta nosturia, mikäli jokin järjestelmä laite ei ole toiminnassa ja sillä pystyy pysäyttämään nosturin rauhallisesti myös manuaalisella etälaukaisulla. Laite oli alkuvaatimusten mukainen, sitä testattiin onnistuneesti ja tilaaja oli siihen tyytyväinen. Laitetta tullaan käyttämään automaatiotesteissä vikatilainteiden hallintaan niin suljetuilla testialueilla kuin oikeilla työmaillakin.

Automaattisten vianhavaitsemistoimintojen kannalta turvalaitteen erillisyys automaatiojärjestelmästä ei olisi välttämätöntä, koska toiminnot pohjautuvat automaatiojärjestelmän sisäiseen tietoon, eli vianhavaitsemistoiminnot ovat alttiita samoille häiriöille kuin automaatiojärjestelmä. Manuaalisen pysäytystoiminnon toimintavarmuuden kannalta laitteen kannattaa kuitenkin ehdottomasti olla erillinen. Erillinen laskentatehoinen laite on myös hyvä varaus tulevien turvajärjestelmien kehitykselle. Jos laitteesta tahdotaan kehittää tulevaisuudessa automaatiojärjestelmän vaarallisen toiminnan estävä laite, sille tulisi pystyä tuoda myös järjestelmän ulkopuolelta tietoa.

Suunnitteluvaiheessa projektissa harkittiin käytettävän ohjelmoitavaa logiikkaa ja logiikkaohjelmointikieliä. Suunnittelun aikana kuitenkin todettiin, ettei logiikkojen ominaisuudet sovellu projektin toteutukseen. Projekti muistuttikin lopulta enemmän IoT-kehitystyötä kuin perinteistä automaatiota. Opinnäytetyössä käytettiin Node-RED, Python ja JavaScript-ohjelmointia. Sen aikana perinteisen ohjelmoinnin taidot kehittyivät Node-RED-in ohella. Lisäksi työn aikana oppi muun muassa XBee-radio, sarjaportti ja USB-kommunikaatiosta, sekä erityisesti tutustui torninostureiden toimintaan.

Lähteet

1 Operating a tower crane is a job full of challenges. 2008. Verkkoaineisto. The News-Gazette. <https://www.news-gazette.com/news/operating-a-tower-crane-is-a-job-full-of-challenges/article_7bad6b7b-a1f5-547a-a32b-6c65074e58b5.html> Luettu: 13.09.2019.

2 Bolton, W. 2009. Programmable Logic Controllers. Oxford, UK: Elsevier.

3 GPIO. 2019. Verkkoaineisto. Raspberry Pi Foundation. <https://www.raspberrypi.org/documentation/usage/gpio/>. Luettu 20.09.2019.

4 Huayao 8-Channel Relay Module. Verkkoaineisto. Shop smart express. <https://www.shopsmartexpress.com/item/B07DNB2NGD/huayao-2pcs-8-channel-dc-5v-relay-module-with-optocoupler-for-arduino-uno-r3-mega-2560-1280-dsp-arm-pic-avr-stm32-raspberry-pi>. Luettu 1.10.2019.

5 JBtek 8 Channel Relay Module. Verkkoaineisto. Amazon. <https://www.amazon.com/JBtek-Channel-Module-Arduino-Raspberry/dp/B00KTELP3I/ref=sr_1_4?keywords=Relay+Module+for+Raspberry+Pi&qid=1568978410&s=electronics&sr=1-4>. Luettu 1.10.2019.

6 Whats wrong with the Raspberry Pi. 2019. Verkkoaineisto. Own your bits. <https://ownyourbits.com/2019/02/02/whats-wrong-with-the-raspberry-pi/>. Luettu 20.9.2019.

7 Raspberry Pi. Verkkoaineisto. Partcon. <https://www.partco.fi/fi/2201-raspberry-pi>. Luettu 1.7.2019.

8 Running Raspberry Pi without an SD card. 2018. Verkkoaineisto. <https://hackaday.com/2018/10/08/hack-my-house-running-raspberry-pi-without-an-sd-card/>. Luettu 1.8.2019.

9 Corrupt file systems every 2–3 month. 2016. Foorumikeskustelu. <https://community.openhab.org/t/corrupt-filesystems-every-2-3-month/13057>. Luettu 1.8.2019.

10 Digi Connect Core. Verkkoaineisto. Digi. <https://www.digi.com/products/embedded-systems/system-on-modules>. Luettu 1.10.2019.

11 Digi Connect Core 6ul sbc Express. Verkkoaineisto. Mouser. <https://www.mouser.fi/new/digi-international/digi-connectcore-6ul-sbc-express/>. Luettu 1.7.2019.

12 Open Source IPC. Verkkoaineisto. Kunbus. <https://revolution.kunbus.com/revpi-core/>. Luettu 1.10.2019.

13 Industrial IoT. Verkkoaineisto. Siemens. <https://w3.siemens.com/mcms/pc-based-automation/en/industrial-iot/Pages/Default.aspx>. Luettu 27.6.2019

14 Siemens IoT2040 Intelligent Gateway. Verkkoaineisto. RS-Online. <https://uk.rs-online.com/web/p/iot-development-kits/1244038/>. Luettu 7.6.2019.

15 Embedded PC CX. Verkkoaineisto. Beckhoff. <https://www.beckhoff.com/english.asp?embedded_pc/cx.htm?id=15987759973374>. Luettu 1.10.2019.

16 Beckhoff CX5130 Embedded-PC. Verkkoaineisto. TP Automation. <http://www.tpautomation.de/Automation-systems/Beckhoff-Embedded-PCs/Series-CX5100/CX5120:::1_3835_3887_3890.html?language=en>. Luettu 1.10.2019.

17 HTM 40 M metalli holkkitiiviste. Verkkoaineisto. Finnparttia. <https://www.finnparttia.fi/epages/finnparttia.sf/fi_FI/?ObjectPath=/Shops/2014102905/Products/HTM40M>. Luettu 28.9.2019.

18 CPU-temperature. 2017. Verkkoaineisto. Kunbus. <https://revolution.kunbus.de/forum/viewtopic.php?t=433>. Luettu 1.10.2019.

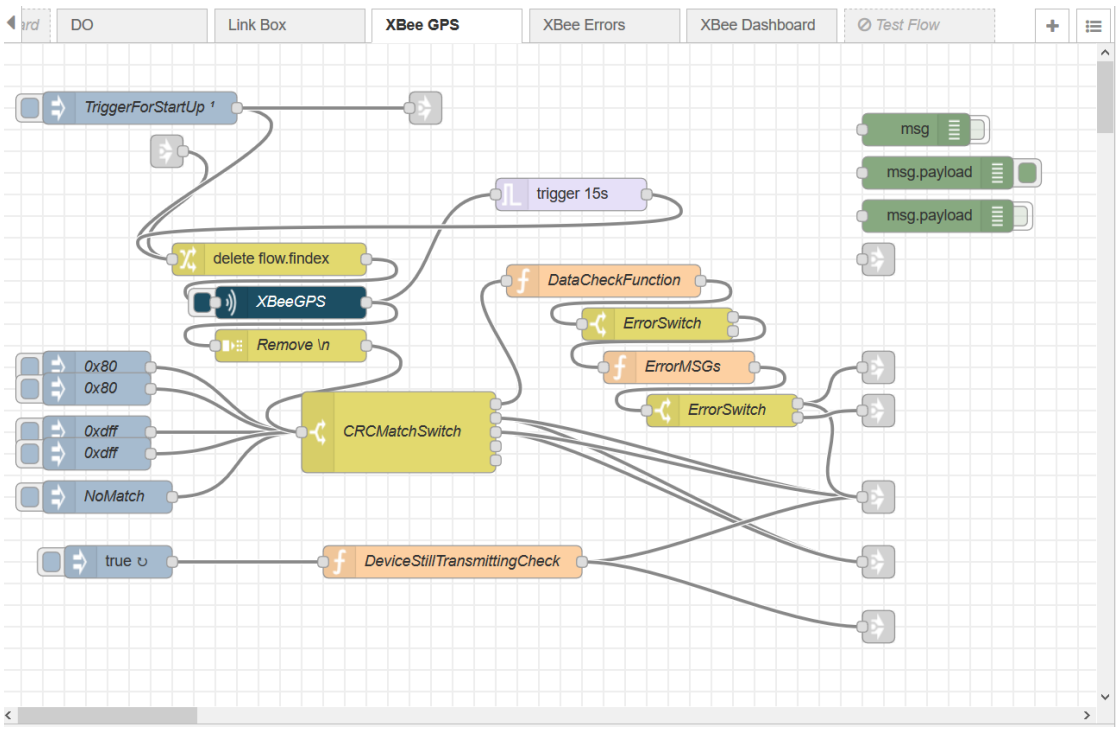
19 Introductory PLC Programming. 2019. Verkkoaineisto. Wikibooks. <https://en.wikibooks.org/wiki/Introductory_PLC_Programming>. Luettu 1.10.2019.

20 Codesys. Verkkoaineisto. Codesys. <https://www.codesys.com/> Luettu 28.8.2019.

21 Node-RED. Verkkoaineisto. Node-RED. <https://nodered.org/>. Luettu 28.8.2019.

22 Hakala, Mika ja Vainio, Mika. 2005. Tietoverkon rakentaminen. Porvoo: Docendo.

Laitteen koodit



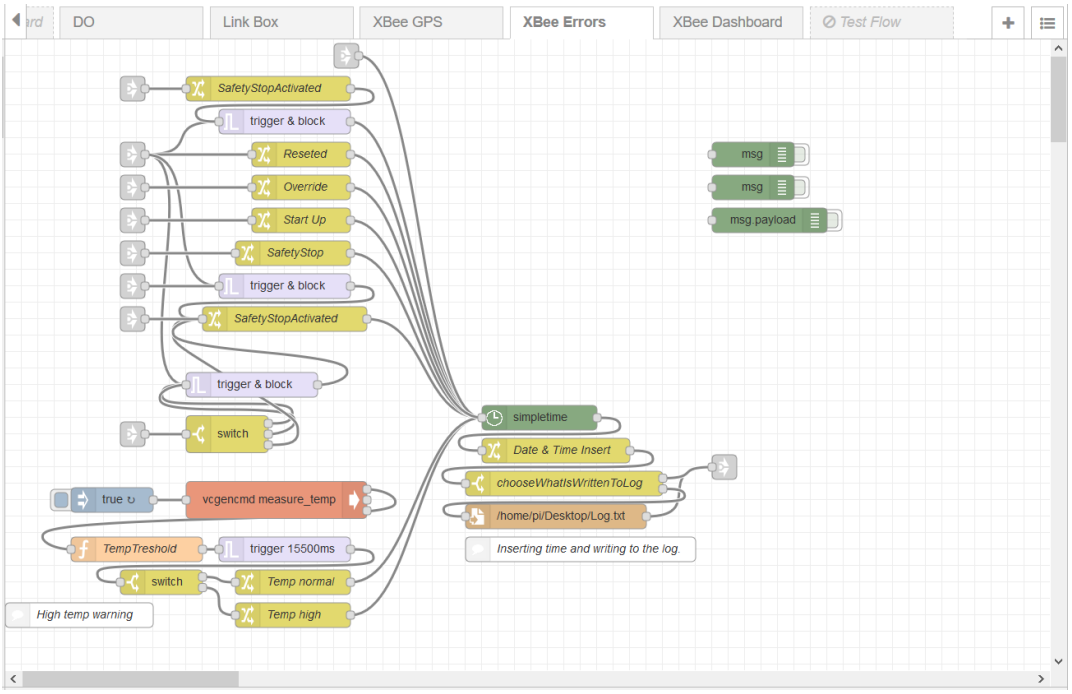
Kuva 1. XBee GPS -flow

```
def main():
      pack_count = 0

      #fname = open(fname, "r")
      names = glob("/dev/ttyUSB*")
      if len(names) >= 1:
            serOfTheRadioModule = 'AK05ZIUN' #CHANGE THIS ID ACCORDING THE RADIO
MODULE SER ID!!!
            ports = subprocess.check_output(["python3", "-m", "se-
rial.tools.list_ports", "-v"]) #Getting list of serial devs.
            ports = ports.decode() #Decoding the list from binary to UTF-8
            serIndex = ports.find(serOfTheRadioModule) #Finding the index of the
SER
            if serIndex == -1:
                  print("Radio module, ", serOfTheRadioModule, "not connected!
Radio is not connected or the ser num is incorrect.")
                  #print("Found ports: ", ports)
                  exit()
            else:
                  ports = ports[0:serIndex] #Slicing the tail of ports from
serIndex
                  serIndex = ports.rindex('/dev/ttyUSB') #Finding the index of
dev path
                  ports = ports[serIndex:] #Slicing the head of ports from
serIndex
                  ports = ports.splitlines() #Listifying the sliced ports. Right
dev path is the first object
```

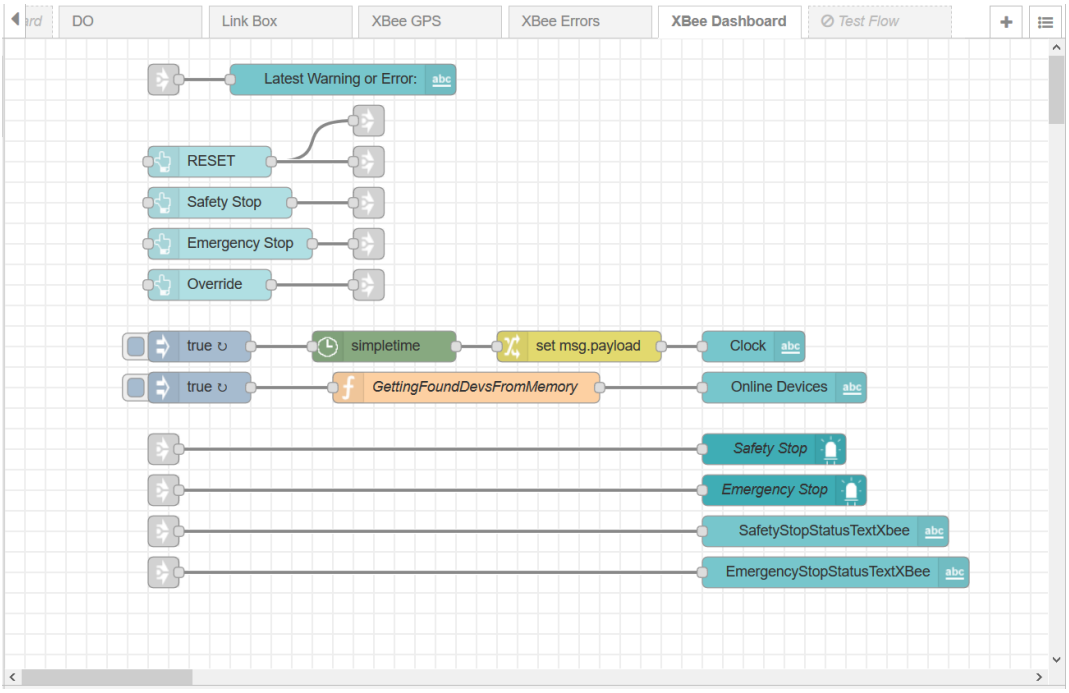
```
                names[0] = ports[0]
                names[0] = names[0].rstrip()

        if len(names) == 0:
             print("no usb devices /ttyUSB found, radio is not connected")
        ser = serial.Serial(names[0], 115200, timeout=5)
        #ser = serial.Serial("/dev/ttyUSB0") #Forcibly set a device path
        sleep(0.01) #give some time to start
        #skip first two lines as they are probably broken sentences
        ch = 0
        buf = []

        delays = []

        for i in range(0,24):
             #buf.append(ord(ser.read(1)))
             try:
                  buf.append(ord(ser.read(1)))
             except Exception as e:
                  print("stream ended")
        ix = 0


        last_time = {}
        time_gap = ()

        while True:
             if buf[0] == 0x3F:
                  #print("start char found", buf[0])
                  binary = np.array(buf[0:22], dtype=np.uint8).tobytes()
                  if crc16.crc16xmodem(binary) == buf[22]*256+buf[23]:
                       dt = datetime.utcnow()
                       arrival_time = dt.hour*3600+dt.minute*60+dt.second+dt.mi-
crosecond/1000000.0
                       gps_time_correction = 18
                       data = []
                       data.append(buf[1])#ix
                       time_millis = decode_custom_integer(np.array(buf[2:6],
dtype=np.uint8), 4)/1000.0
                       dev_id = buf[19]*256+buf[20]
                       fix_status = buf[21]
                       index = (buf[1])
                       gap_error = False

                       if dev_id in last_time: #Checking time gap
                            time_gap = arrival_time - last_time[dev_id]
                            if time_gap >= 3.0: #ALLOWED TIME GAP BETWEEN PACK-
ETS
                                 #print("TOO BIG GAP")
                                 gap_error = True
                            elif time_gap < 0 and (arrival_time+24*3600 -
last_time[dev_id]) >= 3:
                                 #print("TOO BIG GAP")
                                 gap_error = True
                       else:
                            time_gap = 0

                       last_time[dev_id] = arrival_time
                       #print(last_time, dev_id)

                       #print(buf)
```

```
                    latitude = decode_custom_integer(np.array(buf[6:11],
dtype=np.uint8), 5)/1000000000.0
                    #print(latitude)
                    longitude = decode_custom_integer(np.array(buf[11:16],
dtype=np.uint8), 5)/1000000000.0
                    #print(longitude)
                    height = decode_custom_integer(np.array(buf[16:19],
dtype=np.uint8), 3)/1000.0
                    #print(height)
                    #print(checksum)
                    print("CRC match", ":", time_millis, ":", hex(dev_id),
":", fix_status, ":", index, ":", arrival_time-(time_millis-gps_time_correc-
tion), ":", gap_error, ":", time_gap, ":", latitude, ":", longitude, ":",
height)
                    buf = []
                    for i in range(0,24):
                         buf.append(ord(ser.read(1)))

               else:
                    buf = buf[1::]
                    buf.append(ord(ser.read(1)))
          else:
               print("reading next char")
               buf = buf[1::]
               buf.append(ord(ser.read(1)))
          ix += 1

     delays = np.array(delays)
     print("mean", np.mean(delays), "min", np.min(delays), "max", np.max(de-
lays))



     sentence = ser.read(24)
     ix = 0
     a1 = datetime.now()
     while True:
          sleep(0.05)
          sentence = ser.read(24)#message ends in newline
          print(sentence)
main()
```

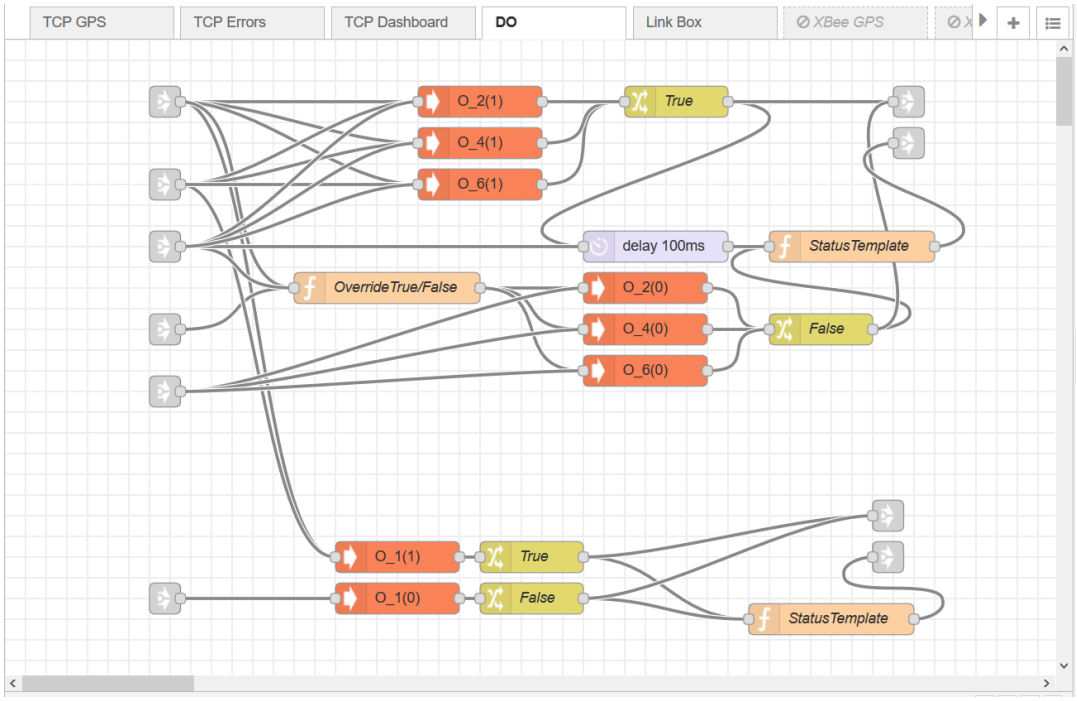
Esimerkkikoodi 1. XBeeGPS.py pääfunktio.



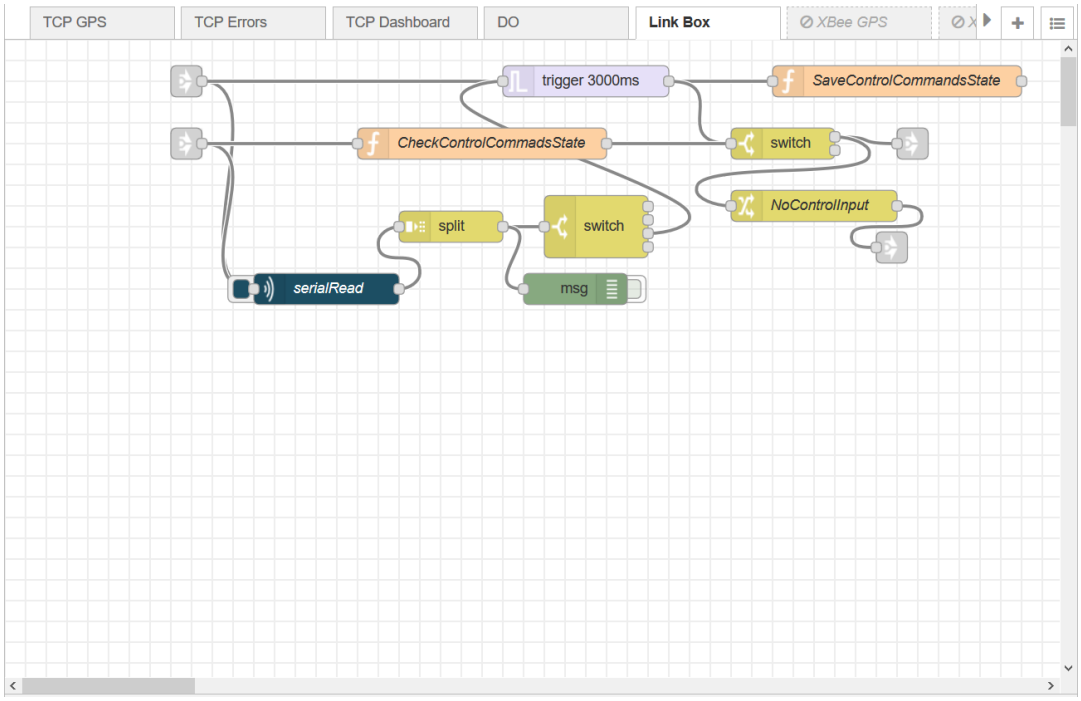
Kuva 2. XBee Errors -flow.



Kuva 3. XBee Dashboard -flow



Kuva 4. DO-flow

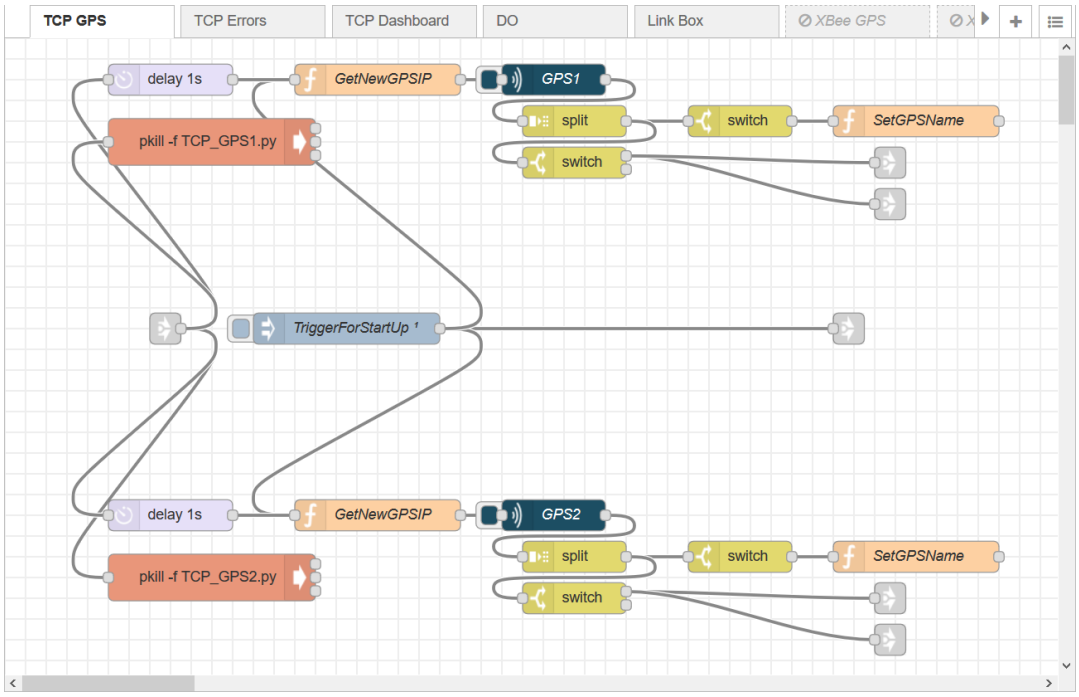


Kuva 5. Link Box -flow

```
import serial
import subprocess
from glob import glob
"""
This code is for reading control commands for old link box and comparing them
to input pin values.

"""

names = glob("/dev/ttyUSB*")
if len(names) >= 1:
     device = 'A907CBPG' #CHANGE THIS ACCORDING THE DEVICE!!! You can use Ser
number, hw desc or VID:PID. Run: python3 -m serial.tools.list_ports -v in ter-
minal find devices' information
     ports = subprocess.check_output(["python3", "-m", "se-
rial.tools.list_ports", "-v"]) #Getting list of serial devs.
     ports = ports.decode() #Decoding the list from binary to UTF-8
     serIndex = ports.find(device) #Finding the index of the SER (or other id
provided)
if serIndex == -1:
     print("Device, ", device, "not connected! Device is not connected or the
ser num is incorrect.")
     #print("Found ports: ", ports)
     exit()
else:
     ports = ports[0:serIndex] #Slicing the tail of ports from serIndex
     serIndex = ports.rindex('/dev/ttyUSB') #Finding the index of dev path
     ports = ports[serIndex:] #Slicing the head of ports from serIndex
     ports = ports.splitlines() #Listifying the sliced ports. Right dev path
is the first object
     names[0] = ports[0]
     names[0] = names[0].rstrip()

if len(names) == 0:
     print("no usb devices /ttyUSB found, device is not connected")

ser = serial.Serial(names[0], 19200)
#ser = serial.Serial('/dev/ttyUSB0', 19200) #Forcibly set a device path

while True:
     read = ser.read(1) #Reading 1 char
     read = read.decode(encoding='ASCII', errors='strict')
     #print(read)
     if(read == 'C'): #Checking if the char was start char
          #print('Start char!')
          packet = read #Saving the first char of the packet

          for i in range(0,20): #Saving the remainig chars of the packet
(20pcs)
               try:
                    read = ser.read()
                    read = read.decode(encoding='ASCII', errors='strict')
                    packet = packet + read
               except Exception as e:
                    print("stream ended")

          if(packet[20] == 'E'): #Checking if the end char is found at the end
of the packet
               stripPack = packet[1:20]
               stripPack = stripPack.replace(',', '')
               #print(stripPack)
               dValues = '' #Digital Input Values
```

```
                aValues = [] #Analog Output Values
                #print('End char.')

                disable = True #UNCOMMENT SECTION TO BYPASS CONTROL COMMAND<-
>INPUTS COMPARISON
                if(disable == True):
                     print('OK')
                     continue

                for i in range(1,15): #Reading values from the DI-pins
                     pin = 'DI'+str(i)
                     read = subprocess.check_output(["piTest", "-1", "-q", "-
r", pin])
                     read = read.decode(encoding='ASCII', errors='strict')
                     dValues = dValues + read
                dValues = dValues.replace("\n","")
                #print(dValues)
                for i in range(1,7): #Reading values from the AI-pins
                     pin = 'AI'+str(i)
                     read = subprocess.check_output(["piTest", "-1", "-q", "-
r", pin])
                     read = read.decode(encoding='ASCII', errors='strict')
                     read = read.replace('\n', '')
                     aValues.append(int(read))
                #print(aValues)

                if(stripPack == '000000100'):
                     print('Emergency Stop')

                if(stripPack == '000000000'):
                     if(dValues == '00000000000000' and aValues[0] >= 0 and
aValues[0] <= 65535):
                          print('OK')
                     else:
                          print('error')

           else:
                print('error: End char not found.')

      else:
           print('Reading next char')
```

Esimerkkikoodi 2. serialRead.py

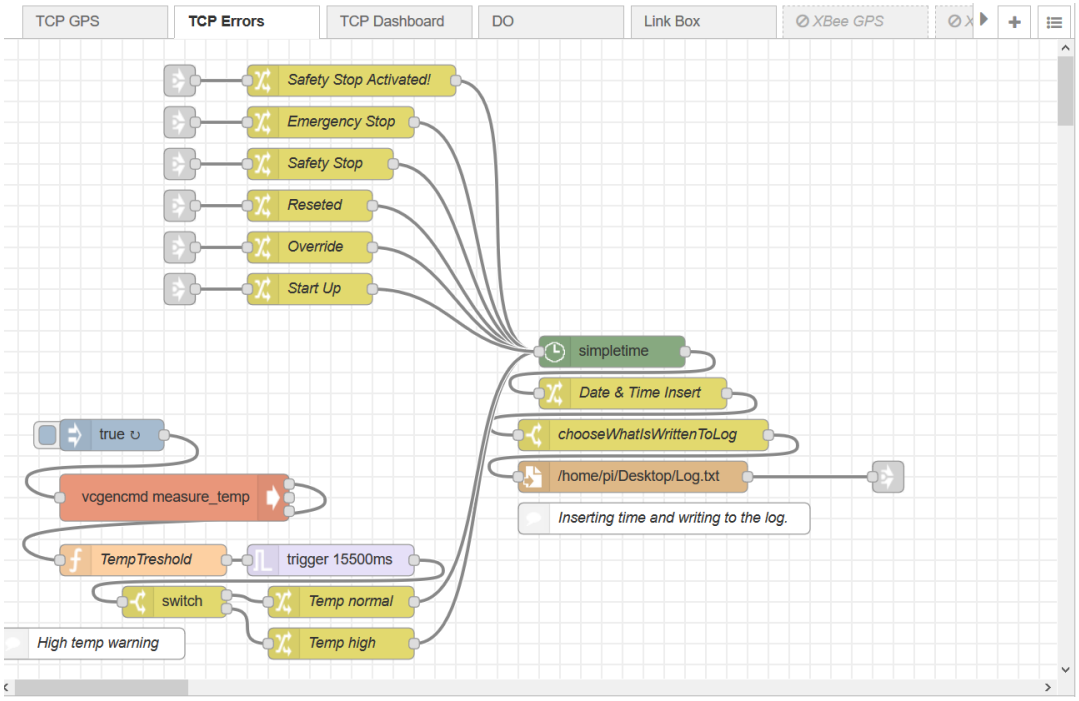


Kuva 6. TCP GPS -flow

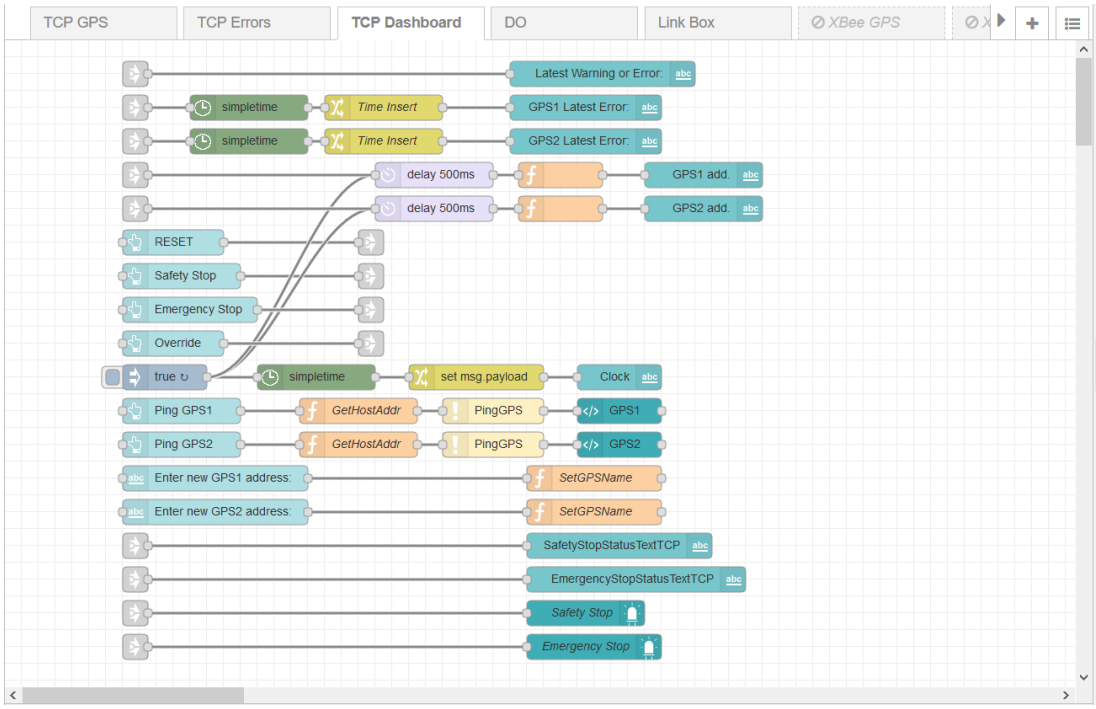
```
import socket
import time

#HOST = '10.15.10.234'  # ENTER the GPS's IP-ADDRESS HERE!
HOST = input()
PORT = 9001         # Port to listen on
singleTime = ''

print('IP: ' + HOST + '\n')
socket.setdefaulttimeout(3)

try:
    while True:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.connect((HOST, PORT))
            data = s.recv(142) #String length from Reach M+ is 142
            recvTime = time.time()
            data = data.decode() #Decoding the list from binary to UTF-8
            data = data.split() #Listifying the data string
            #Packet format: date, time, lati, long, heig, SolStat, sats, -, -,
-, -, -, -, -, -
            print(data)

            if data[5] == '5':
                if len(str(singleTime)) == 0:
                    singleTime = recvTime
                elif recvTime - singleTime > 5:
                    print('error: GPS(' + HOST + ') been in single mode more
than 5s.')
                    break
            else:
                singleTime = recvTime
```

```
except Exception as e:
    print('error: socket timeout. ' + HOST)
```

Esimerkkikoodi 3. TCP_GPS1.py



Kuva 7. TCP Errors -flow



Kuva 8. TCP Dashbaord -flow