Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Panu Sutela

Teollisen Internetin välittäjän kehittäminen pienoistietokoneelle

Metropolia Ammattikorkeakoulu

Sähkö- ja automaatiotekniikka

Automaatiotekniikan insinööri

Opinnäytetyö

4.2.2021

Tekijä Otsikko Sivumäärä Aika	Panu Sutela Teollisen Internetin välittäjän kehittäminen pienoistietokoneelle 21 sivua + 5 liitettä 4.2.2021
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	sähkö- ja automaatiotekniikka
Suuntautumisvaihtoehto	automaatiotekniikka
Ohjaajat	teknologiamanageri Kalle Ahola yliopettaja Erkki Räsänen

Projektin tarkoituksena oli kehittää pienoistietokoneella toimiva datavälttäjä, jolla voidaan lukea mielivaltaisia tietoja ja välittää ne haluttuun kohteeseen. Tietoa voidaan välittää erilaisilla teollisuuden protokollilla ja uusia välitysmetodeja on helppo lisätä.

Projekti kehitettiin yhteistyössä Pinja Groupin kanssa. Pinja Group antoi työhön aiheen ja laitteiston. Tästä huolimatta työtä ei suoranaisesti tehty Pinja Groupille. Projektin lopputavoite määriteltiin alussa vain pääpiirteisesti, ja sen suunta kehittyi vasta työn edetessä.

Projektin perustavien teknologioiden ansiosta koodin kehittäminen voitiin toteuttaa suoraan ohjelmalle tarkoitetulla mikrotietokoneella. Tämä mahdollisti nopean koodin testaamisen, sekä varmisti järjestelmän yhteensopivuuden.

Ohjaavana periaatteena projektissa oli muokattavuus ja modulaarisuus. Kuitenkin johtuen suunnittelun puutteista työn aikana tutkittiin useita kehitysmahdollisuuksia, jotka kuitenkin hylättiin projektin edetessä. Esimerkiksi yhteyttä Google Sheetsiin käytettiin järjestelmän testaamiseen työn varhaisessa vaiheessa. Tämän ominaisuuden kehitys jätettiin kuitenkin kesken sen potentiaalisen tarpeen puutteen takia.

Lopputuloksena syntyneelle ohjelmalle annettiin nimeksi RICO. Projektille on valmiiksi määritelty moduulit sekä OPC UA-, että MQTT-viestintään, ja ohjelmalle voidaan antaa uusia komentoja MQTT-yhteydellä ja komentotiedostolla. Projekti on toteutettu Raspberry Pilla käyttäen Raspberry Pi OS:aa kokonaisuuden hinnan sekä laajan käyttäjämäärän takia, ja projektin ohjelmointikielenä on Python 3.7 samoista syistä.

Valmistuneella ohjelmalla ei ole erityisiä käyttökohteita. Kuitenkin muokattavuutensa ansiosta RICO sopii useisiin hypoteettisiin käyttökohteisiin. Suunnittelupohjana käytettyjen esimerkkien mukaan sillä voi ohjata tuotannon tietoja etätuelle tai automaattisesti asettaa laitteiden parametreja.

Avainsanat	Raspberry Pi, Python, MQTT, OPC UA

Author	Panu Sutela
Title	Implementation of an Industrial Internet Repeater on a Microcomputer
Number of Pages	21 pages + 5 appendices
Date	4 Feb 2021
Degree	Bachelor of Engineering
Degree Programme	Electrical and Automation Engineering
Specialisation option	Automation Technology
Instructors	Erkki Räsänen, Senior Lecturer Kalle Ahola, Technology Manager

The aim of the project was to develop a simple data collector or broker for an off-the-shelf microcomputer, that can read and transmit arbitrary information to and from arbitrary locations. This information can be read and transmitted via any defined industrial communications protocol and new ones can be added with ease.

The project was developed in partnership with Pinja Group that provided the necessary hardware and subject for this work. Despite this, the project was not technically made for Pinja Group. The end goal for the project was defined very loosely and the direction of the project solidified during the development.

Thanks to the platform being used, developing the code was possible on the microcomputer itself. This enabled rapid testing and ensured the compatibility of the elements in the project.

One basic guiding principle in the project was modularity so new functionality could be added easily. Due to the unfocused development, the development process explored many functions that were not included in the final version for various reasons. For example, Google Sheets was used as a testing platform. Support for connecting to it was dropped due to lack of perceived potential utility.

The program that was born as result was named RICO. As it stands, the program supports communications using both OPC UA and MQTT protocols and it can be controlled using a MQTT connection or a controlling file on the executing device. The selected platform for the project was Raspberry Pi 4 model B running Raspberry Pi OS for its affordability and community support. For the programming language used, Python 3.7 was chosen for similar reasons.

Resulting program does not have any immediate use cases. However, due to its malleability, RICO is fit for many hypothetical uses. For example, it can transmit process data from a manufacturing plant to a remote technical support or automatically set up program variables for devices.

Keywords	Raspberry Pi, Python, MQTT, OPC UA

Sisällys

Liitteet

Liite 1. Main.py

Liite 2. Textfile_UI.py

Liite 3. OPCUA_Connector.py

Liite 4. MQTT_UI.py

Liite 5. MQTT_Connector.py

Lyhenteet ja käsitteet

Aihe	Englanniksi ja koodissa ”topic”. MQTT-kommunikaatiossa viittaa sen sisäiseen osoitteen tai polun tapaiseen tunnukseen.
API	*Aplication Progamming Interface*. Ohjelmointirajapinta.
Apuolio	Koodissa ”entity”. Tässä projektissa erityisesti RICOn sisäinen olioryhmä, joka voi säilöä mielivaltaista tietoa esim. käynnissä olevista yhteyksistä.
IDE	*Integrated Development Environment*. Sulautettu ohjelmointiympäristö.
Kohde	Koodissa ”node”. Tässä projektissa erityisesti RICOn sisäinen olio, joka voi vastaanottaa ja/tai lähettää dataa.
Moduuli	Koodissa ”module”. Tässä projektissa erityisesti RICOn erityisestä tiedostosta lataama paketti, joka sisältää uusia ominaisuuksia.
MQTT	*Message Queuing Telemetry Transport*. Yleinen protokolla tiedon välitykseen palveluiden ja IIoT-laitteiden välillä.
Muunnin	Koodissa ”transformer”. Tässä projektissa funktio, jolla tietoa muunnetaan muodosta toiseen.
OPC UA	*OPC Unified Architecture*. Teollisuudessa yleisesti käytetty Ethernetiin pohjautuva viestintäprotokolla.
RICO	Lopputuloksena syntyneen ohjelman nimi.
Silta	Koodissa ”bridge”. Tässä projektissa RICOn sisäinen olio, joka vastaa saapuvan tiedon käsittelystä ja sen suuntaamisesta oikeisiin lähettäviin kohteisiin.
UI	*User Interface*. Käyttöliittymä.
Välittäjä	Englanniksi ”broker”. Yleisenä terminä tietoa välittävä ohjelma. MQTT:n suhteen erityismerkitys

1 Johdanto

Tämän insinöörityön aiheena on kehittää Raspberry Pille Python-ohjelmisto, joka kerää dataa eri verkkolähteistä ja ohjata sitä eteenpäin muihin kohteisiin. Yleisesti tällaista ohjelmaa nimitetään välittäjäksi tai brokeriksi. Sekä keräys- että lähetyskohteet voidaan määritellä mielivaltaisesti ja uusien viestintäprotokollien sekä ominaisuuksien määrittäminen on tehty helpoksi.

Tuloksena syntynyt ohjelma on nimetty RICOksi. Se valittiin ihan vain lähinnä, koska projektilla piti olla jokin nimi, jotta siitä pystyy puhumaan ja kirjoittamaan järkevästi. RICO sopi tarkoitukseen sekä kuulosti miellyttävältä. Alun perin nimi on myös ollut ohjelman toimintakuvauksesta johdettu akronyymi, mutta koska sen muodostaneita sanoja ei koskaan kirjattu ylös, on sen merkitys lukijan mielikuvituksen täytettävissä.

Ohjelma on laadittu täysin englannin kielellä. Tämän takia, ja koska monilla aihetta koskevilla termeillä ei ole vakiintuneita suomenkielisiä termejä, tässä raportissa on jouduttu käyttämään paljon englanninkielistä sanastoa suomen seassa ja on tiedostettu, että tämä voi häiritä lukukokemusta.

Laitteistolle ei alussa määritelty yksiselitteisiä käyttökohteita tai tarkkoja määritelmiä. Lähtökohtaiseksi rajaukseksi projektille asetettiin lyhykäisyydessään mukautettava, Raspberry Pi -pohjainen, Pythonilla toteutettu Ethernet-yhdistin. Suuntaa antavina ehdotuksina on kuitenkin käytetty datan välittäminen OPC UA -väylältä MQTT-välittäjille ja ulkoisen laitteen parametrien automatisoitu asettaminen. Nämä eivät olleet pöydällä projektin alusta lähtien vaan työn suunta ja päämäärä rakentuivat työn edistyessä.

Projekti on suoritettu yhteistyössä Pinja Groupin kanssa. Pinja ehdotti työn aiheen, konsultoi projektin kehittämistä sekä tarjosi projektin vaatiman kehitysympäristön ja laitteiston.

Pinja on suomalainen teollisuuden ICT:hen ja automaatioon erikoistunut yritys, joka tarjoaa digitaalisia ratkaisuja työn sujuvoittamiseen ja huippusuorituksen tukemiseen vaativissa teollisuus- ja tuotantoympäristöissä. Näihin kuuluu muun muassa ylläpitoa, ohjelmistoratkaisuja ja suunnittelupalveluita. Pinjan asiakkaisiin kuuluvat muun muassa Fazer, ABB, VTT, Neste, Metso ja Opetushallitus. Se muodostui vuonna 2020 kun Protacon, ARROW, SWD, Descal, Netwell, Vision Systems ja Powen yhdistyivät. [1.]

2 Käytetyistä teknologioista

2.1 Python

Python on ohjelmointikieli, joka on hyvin suuri suosittu ympäri maailman erityisesti tutkimuksen ja harrastuspohjaisen ohjelmoinnin alueilla. Vuonna 2019 maailmassa oli noin 8.2 miljoonaa aktiivista Python ohjelmoijaa [2]. Ensimmäinen versio Pythonista julkaistiin helmikuussa 1991 tehden siitä tämän raportin kirjoitushetkellä 30 vuotta vanhan [3]. Vuosien varrella kieli on kehittynyt ja vuoden 2021 alussa sen uusin stabiili versio olikin 3.9.1. Pythonin versio 2.7 on ollut huomattavan suosittu 3.x-versioiden yhteensopivuusongelmien takia, mutta tuki sille lopetettiin vuoden 2020 alussa. 2019 Pythonin suosituin versio oli 3.7, jota käytti noin puolet kaikista Python-kehittäjistä [4].

Python perustuu avoimeen lähdekoodiin. Sen filosofiaan kuuluu pitää koodi selvänä ja helposti luettavana. Tämä näkyykin sen syntaksissa, joka vaatii tarkasti sisennettyjä rivejä ja käyttää selkokielisiä komentoja. Koodia ei myöskään tarvitse kääntää etukäteen, vaan tämä tapahtuu automaattisesti koodia suorittaessa.

2.2 Raspberry Pi

Raspberry Pi on suosittu mikrotietokonemerkki, jota kehittää Raspberry Pi Foundation. Pienen kokonsa, suuren lisälaitevalikoimansa ja alhaisen hintansa ansiosta ne ovat yleisiä muun muassa opetuksessa, harrastelijatason automaatioprojekteissa ja pelikonsoliemulaattoreina. Raspberry Pillä on myös potentiaalia teollisuuden käyttökohteisiin, kunhan laitteiston luotettavuus ja turvallisuus kehittyvät [5].

Raspberry Pistä on useita versioita eri tarkoituksiin. Uusin malli joulukuussa 2020 on Raspberry Pi 4 model B, jolla tämäkin opinnäytetyö on toteutettu. Tässä mallissa liitäntöinä on USB-C virransyötön lisäksi kaksi micro HDMI -liitäntää näytöille, neljä USB-A porttia, gigabittinen Ethernet-portti, Bluetooth sekä 2,4 ja 5 GHz:n langaton yhteys. Huomattavana päivityksenä aiempiin malleihin 4 model B sisältää myös neliydinprosessorin, joka mahdollistaa todellisen ohjelmien moniajon. [6]

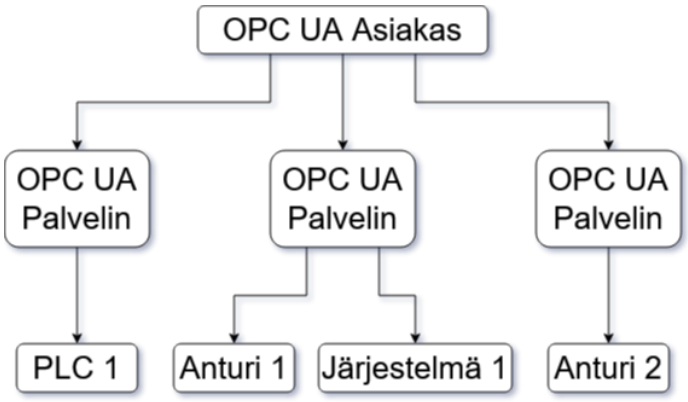
Raspberry Pi Foundation ylläpitää myös Raspberry OS -nimistä käyttöjärjestelmää, joka on tarkoitettu Raspberry Pi -laitteille. Käyttöjärjestelmä on Debian-pohjainen ja sen mukana tulee asennettuna suuri määrä työkaluja ohjelmistokehitykseen.

2.3 OPC UA

OPC UA on uudistettu versio vanhemmasta, vuonna julkaistusta, OPC-viestintäprotokollasta (Open Platform Communications). Se on vuoden 2006 julkaisunsa jälkeen kasvanut niin käyttäjämäärässään kuin ominaisuuksissaan ja on nykyään teollisuudessa erittäin laajassa käytössä. Vuonna 2019 kehittävässä OPC Foundation -järjestössä oli yritysjäseniä yli 700 kaikilta automatisoidun tuotannon aloilta. [7.]

OPC UA on alustasta riippumaton ja tukee suurta määrää erityyppisiä toimintoja natiivisti. Lähetetty tieto kulkee täysin salattuna lähteen ja päämäärän välillä, ja sen tietopaketeissa on useita ominaisuuksia, joilla sen aitous voidaan todentaa. Protokolla sisältää toiminnallisuudet muun muassa tuntemattomien palvelimien etsintään verkoista, kyvyn tilata ilmoituksia arvojen muutoksista tai muista huomattavista tapahtumista ja mahdollisuuden sekä lukea että kirjoittaa palvelimen arvoja. Tämän lisäksi, jatkuvasta kehityksestä huolimatta, se on hyvin yhteensopiva aiempien versioidensa kanssa. OPC Classic on pitkälti yhteensopiva OPC UA:n kanssa OPC Foundationin kehittämien COM/Proxy wrappereiden avulla [8].

Poiketen perinteisestä palvelinkeskeisestä mallista, OPC UA:ssa palvelimet sijaitsevat pääosin dataa lähettävissä laitteissa kuten antureissa tai PLC:ssä ja tietoa käsittelevä laitteisto jolle arvot lähetetään, toimii asiakkaana. Kuten kaavio 1 esittää, ajoittain tämä johtaa verkkorakenteisiin, joissa monta pientä ja yksinkertaista palvelinta on yhteydessä yhteen tai muutamaan monimutkaisempaan asiakkaaseen.



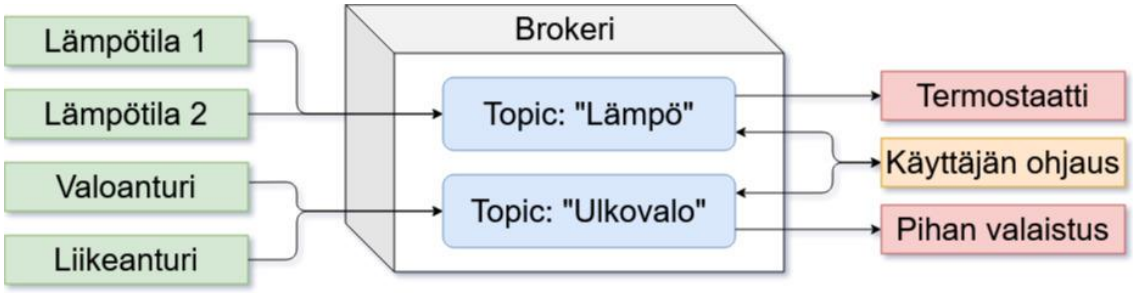
Kaavio 1. Kuvaus OPC UA:n tominnasta.

Suuren kehittäjäkuntansa, sen perusrakenteen vanhojen juurien ja laajan ominaisuuslistansa takia OPC UA on varsin monimutkainen ja vaikeakäsitteinen kokonaisuus. Esimerkiksi geneerinenkin node sisältää muuttujat ID:lle, noden tyypille, sisäiselle nimelle, ulkoiselle nimelle ja kuvaukselle. Tämän tyyppisillä ominaisuuksilla pyritään ilmeisesti parantamaan järjestelmän ymmärrettävyyttä, mutta ne myös lisäävät sen monimutkaisuutta. Vaikka OPC UA onkin vielä yleisessä käytössä ja sen toiminnoille on selvästi käyttöä, muut selkeämmät viestintäprotokollat, kuten MQTT, ovat alkaneet nousemaan sen rinnalle tietyissä sovelluksissa.

2.4 MQTT

MQTT on 1999 kehitetty viestintäprotokolla erikokoisiin automaatioratkaisuihin, alun perin sitä käytettiin öljyputken sensoreiden kommunikointiin. Sen etuina ovat selkeys, pieni datajalanjälki ja keveät asiakassovellukset. Se on kuitenkin riippuvainen keskeisistä välittäjistä, eikä siinä ei ole juurikaan sisäänrakennettuja ominaisuuksia salaukseen tai muuhun yksinkertaisesta tiedonvälityksestä poikkeavaan.

MQTT-verkon keskustassa on välittäjä, jonka kautta kaikki siinä liikkuva informaatio kulkee. Välittäjällä tieto on organisoitu aiheisiin, merkkijonopohjaisiin tunnuksiin, joista muodostuu perinteistä kansiorakennetta muistuttava hierarkia. Asiakkaat eivät suoraan lue aiheita, vaan tilaavat niitä. Kun jokin asiakas lähettää välittäjän aiheelle viestin, välittäjä lähettää sen kaikille sitä tilaaville asiakkaille. Tämän rakenteen ansiosta asiakkaina toimivat laitteet voivat olla hyvinkin pieniä niin fyysiseltä kooltaan kuin suorituskyvyltäänkin ja soveltuvat erinomaisesti IoT-sovelluksiin. Tästä esimerkki kaaviossa 2.



Kaavio 2. Esimerkki yksinkertaisesta MQTT-kotiautomaatiosovelluksesta

Minimalistisesta rakenteestaan huolimatta MQTT tarjoaa viestinvälitykseen muutaman lisätoiminnon. Viesteille voi määritellä, kuinka viestin perille saapuminen tulee varmistaa ja lähetetäänkö aiheen tuorein viesti automaattisesti uusille tilauksille. Asiakkaat voivat myös määrittää testamentin. Tämä on viesti, joka lisätään automaattisesti aiheeseen, mikäli asiakkaan yhteys välittäjälle katkeaa.

3 Työkalut

3.1 Ohjelmointityökalut

Projektin ohjelmointikieleksi valittiin Python sen suuren tukiverkoston ja käyttäjäystävällisyyden takia. Ohjelma on toteutettu Pythonin 3.7-versiolla lähinnä käytettävien kirjastojen yhteensopivuuden takaamiseksi.

Valtaosa ohjelman koodista on luotu Thonnyllä. Se on hyvin kevyt Python IDE, joka tulee valmiiksi asennettuna Raspberry Pi OS:ssa. Edistyneempiä editoreja on harkittu mutta Thonny toteuttaa kaiken tarpeellisen; sillä voi muokata useita tiedostoja yhtä aikaa, siinä on terminaali pythonkoodin suorittamiseen, ja se tarjoaa yksinkertaisen syntaksin tarkistuksen.

3.2 Fyysinen laitteisto

Raspberry Pi 4 model B (kuva 1) on pienoistietokone, jossa on Ethernet-liitäntä ja sarjaportteja. Se valittiin projektin pohjaksi sen suuren käyttäjäkunnan ja halvan hinnan takia. Se on myös tarpeeksi tehokas soveltuakseen itsessään kehitysympäristöksi.

Kuva 1. Raspberry Pi 4 model B

Raspin virtalähteenä käytetään Raspberry Pin virallista virtalähdettä, ja sen tallennustilana toimii Kingstonin 16 Gb micro-sd kortti. Ohjelmointia varten Raspiin on myös liitetty geneerinen näyttö, näppäimistö ja hiiri.

3.3 Ulkoiset resurssit

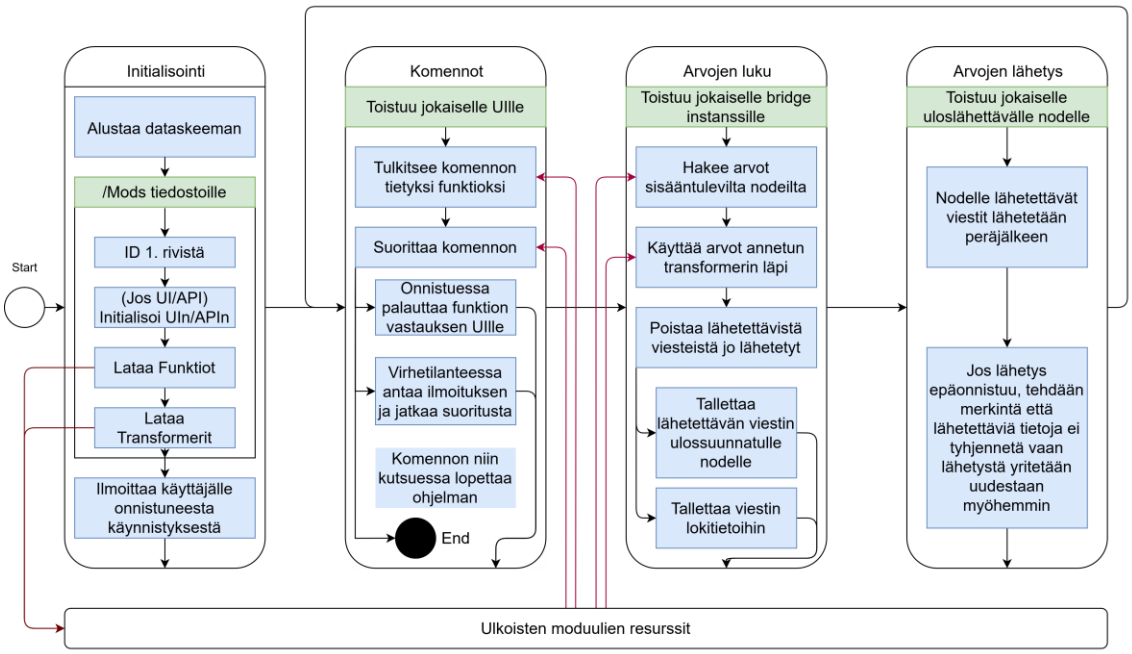
Kehitysympäristössä Raspberry Pi on ollut kiinni kotitalouden verkossa reitittimen kautta, ja kun tarvetta on ollut, ulkoisia tietolähteitä on emuloitu samassa verkossa olevalla tavallisella pöytäkoneella.

Projektin kehityksessä on luonnollisesti käytetty paljon Googlen ja DuckDuckGon avulla internetistä etsittyä tietoa ja neuvoa. Tiettyjä verkkoresursseja on myös käytetty kehityksen aikaiseen datan syöttöön ja vastaanottamiseen, nimellisesti Google Sheetsiä [9] ja HiveMQ:n julkista MQTT-välittäjää [10].

4 Pääohjelman rakenne

Projektin ohjaavana filosofiana on modulaarisuus. Siksi kaikki viestintäprotokollat ja rajapinnat on määritelty omissa moduuleissaan, joita RICO voi hyödyntää. Tämä takaa eri teknologioiden yhteensopivuuden ja ohjelman räätälöitävyyden. Moduulit pääsevät käsiksi lähes kaikkeen ohjelman ajonaikaiseen tietoon, joten niillä voidaan toteuttaa hyvinkin radikaaleja toimintoja.

Kuten kaavio 3 kuvaa, RICO tarkastaa aluksi kaikki tiedostot ”/Mods”-kansiossa ja lataa niissä määritellyt ominaisuudet osaksi pääohjelmaa. Näihin ominaisuuksiin voi kuulua uusia funktioita, viestintäprotokollia ja muuntimia. Sopivat tiedostot tunnistetaan niiden ensimmäisen rivin perusteella, joka kertoo, ladataanko se UI:na, tavallisena moduulina vai jätetäänkö tiedosto lataamatta kokonaan. Erona UI- ja perusmoduulien kanssa on, että UI-moduulissa on erillinen Python-luokka, jonka kautta RICO voi vastaanottaa uusia komentoja. Vaikka UI yleisesti viittaakin nimenomaisesti ihmiskäyttäjään, myös erilaiset APIt luokitellaan tässä UI:ksi, sillä raja näiden välillä on tässä tapauksessa hieman häilyvä.



Kaavio 3. Kaavio ohjelman toiminnasta.

Initialisoinnin jälkeen ohjelma siirtyy kaavion 3 esittämään pääkiertoon, jossa se toimii seuraavasti:

1. Lukee ja suorittaa kaikilta määritellyiltä rajapinnoilta saadut komennot, kuten käskyt uusien yhteyksien muodostamisesta tai pyynnöt tilatiedoista. Mikäli suoritettu funktio palauttaa jonkin arvon tai epäonnistuu suorituksessaan, tieto siitä palautetaan käyttäjälle.

2. Kerää kaikki välitettävät tiedot määritetyistä kohteista. Tiedot voidaan kerätä joko tietyin aikavälein tai arvon muuttuessa.

3. Muokkaa kerätyt tiedot haluttuun muotoon ja lähettää ne eteenpäin määriteltyihin kohteisiin.

4.1 Ohjelman dataskeema

Lähestulkoon kaikki ohjelman tarvitsema data sijaitsee Data-nimisen sanakirjan sisällä ajon aikana, kuten esimerkkikoodi 1 esittää. Tämä tekee tiedon jakamisesta helppoa. Python ei suoranaisesti siirrä tai kopioi sanakirjoja ennen kuin on pakko, vaan ohjelman sisällä liikkuvat pääosin vain pointterit. Tämän takia toteutus minimoi myös järjestelmäresurssien käytön.

Tämän lisäksi siltojen käsittelemä, automaattisesti kohteille lähetettävä tieto säilytetään erillisessä sanakirjassa nimeltä Datalists. Tämä on toteutettu teknisistä syistä ja jotta käyttäjä ei pääse sotkemaan lähetettävää dataa.

```
Data = {
        "UIs":        {},
        "Mods":       {},
        "Entities":   {},
        "Nodes":      {},
        "Bridges":    {},
        "Last_values":{},
        "Sys_status":{
                      "Sending":{},
                      "Commands":[""]*Commands_status_track_limit
                      },
        "Functions": {
                      "Main":{
                              "memory":     show_memory,
                              "new_bridge": new_bridge,
                              "del_bridge": del_bridge,
                              "del_node":   del_node,
                              "pass_value": pass_value,
                              "terminate":  terminate,
                              "values":     show_values,
                              "pass":       pass_function,
                              "dump":       dump_to_file,
                              "status":     sys_status,
                              "bye":        disable_UI
                              }
                      },
        "Transformers": {
                      "Main":{
                              "pass":    pass_transformer,
                              "to_str":  to_str_transformer,
                              "example": example_transformer
                              }
                      },
        "Configuration":{
                      "Brake" : Execution_brake_time
                      }
        }
```

Esimerkkikoodi 1. Kuvaus RICOn dataskeemasta

Dataskeeman avaimilla on seuraavat tarkoitukset:

- Data[”UIs”] sisältää kaikki käyttöliittymät ja APIt nimi/instanssi pareittain. Joka suoritussyklin aluksi sanakirjan instansseilta pollataan uudet suoritettaviksi annetut komennot. Näille olioille ominaisista metodeista get_orders() hakee viimeisimmät komennot ja response() antaa käyttäjälle palautetta ohjelmasta.
- Data[”Mods”] listaa kaikki normaalit moduulit ja niiden oliot.
- Data[”Entities”] sisältää kaikki apuoliot. Näihin voidaan säilöä tietoliikenneyhteyksien oliot, kirjautumistiedot tai mitä vain muuta tarpeellista tietoa.
- Data[”Nodes”] säilyttää luokan kohde instanssit nimineen, joilla määritellään datan lähetys ja keräyspaikat. Huomattavia luokan metodeja ovat get_data() joka hakee kohteen sen hetkiset arvot ja put_data() joka lähettää kohteelle dataa.
- Data[”Bridges”] sisältää kaikki luokan bridge instanssit. Nämä oliot säilövät yhdessä paketissa joukot kohteita, joilta tietoa haetaan ja joille sitä välitetään, ja muuntimen, joka muokkaa tiedon haluttuun muotoon. Tämä luokka sisältää myös koodin kaiken edellä mainitun suorittamiseen.
- Data[”Last_values”] sisältää viimeisimmät arvot kaikista luetuista kohteista.
- Data[”Sys_status”] sisältää viestejä järjestelmän sen hetkisestä tilasta. Vakiona siinä on tiedot kymmenestä viimeisimmästä suoritetusta komennosta ja lähetettyjen tietopakettien tila.
- Data[”Functions”] sanakirja sisältää kaikki käyttäjän käytettävissä olevat funktiot kutsumanimineen. Funktiot on lajiteltu moduulin mukaan, josta ne on saatu. Pääohjelman funktiot ovat ”Main”-avaimen alla. Funktioista lisää kohdassa 4.2.1.
- Data[”Transformers”] on pitkälti samanlainen kuin Data[”Functions”], paitsi että se säilöö muuntimia. Muuntimista lisää kohdassa 4.2.3.
- Data[”Configuration”] sisältää tietoja liittyen ohjelman yleiseen suorittamiseen. Tällä hetkellä ohjelma ei juurikaan käytä tätä sanakirjaa, mutta jatkokehityksen kannalta tämä voi osoittautua hyvinkin tärkeäksi.

4.2 Toiminnallisia elementtejä

4.2.1 Funktiot

Funktiot ovat käyttäjän käytettävissä olevia komentoja, joilla voidaan toteuttaa muutoksia RICOn toiminnassa tai saada siitä tietoja. Pääohjelma sisältää valmiita funktiota, joihin moduulit sekä UI:t voivat lisätä omiaan. Kun funktiota halutaan käyttää, jonkin UI:n tulee antaa käsky sanakirjana muodossa {”module”:”<moduuli>”, ”function”:”<funktio>”, ”args”: {<argumentit>}}.

```
{"module":"OPCUA", "function":"new_node", "args":{"name":"OPC_Node",
"connection":"OPCUA_Conn", "nodeid":"ns=2;i=3"}}
```

Esimerkkikoodi 2. Esimerki funktion kutsusta.

Esimerkkikoodissa 2 on kuvattuna komento, joka lisää OPC UA -moduulilla uuden kohteen nimellä OPC_Node ja jonka ID on ”ns=2;i=3". Johtuen ohjelmaa ohjaavien tekniikoiden luonteesta komennot annetaan useimmiten JSON-merkkijonona, josta varsinainen sanakirja luodaan. RICO on toteutettu niin, että mikäli ”module” puuttuu tai on jätetty tyhjäksi, ohjelma hakee funktion sen pelkän nimen perusteella. Tämä voi kuitenkin johtaa väärän funktion käyttöön, mikäli useampi moduuli lisää saman nimisen funktion.

Ohjelman sisällä suoritettavalle funktiolle annetaan kolme tietoa: komennon kutsuneen UI:n referenssin sekä sanakirjat Data ja args. UI on tarpeellinen, koska funktio voi välittää käyttäjälle dataa kutsumalla UI.response(”viesti”) tyylistä komentoa. Data sanakirjan avulla funktio voi suorittaa lähes mitä hyvänsä muutoksia ohjelman toimintaan. Lopulta args sisältää kaikki funktiolle annettavat lisätiedot sen tehtävästä, kuten käsiteltävien kohteiden nimet, osoitteet ym. Kaikki funktiot eivät välttämättä käytä mitään argumentteja ja näissä tapauksissa argsin voikin jättää kokonaan pois.

Mikäli funktio palautta jonkin arvon, se annetaan automaattisesti käyttäjälle viestinä funktiota kutsunutta UI:ta käyttäen.

RICOssa on sisäänrakennettuna seuraavat vakiofunktiot:

- "memory" näyttää käyttäjälle listan ohjelman sisäisestä tilasta. Ei argumentteja.

- "new_bridge" luo uuden siltainstanssin tai kirjoittaa yli vanhan. Argumentteina sille annetaan sillan nimi, totuusarvo ”only_new_data”, liukuluku ”delay”, lista ”nodes_in”, lista ”nodes_out” sekä listan tai merkkijonon ”transformer”. ”only_new_data” määrittää, vaaditaanko arvossa muutos ennen kuin se välitetään ja ”delay” kerto sekunteina kuinka usein tiedot tarkistetaan. ”nodes_in” ja ”nodes_out” määrittävät käytettävät kohteet, ja ”transformer” asettaa halutun muuntimen.

- "del_bridge" poistaa halutun sillan. Argumenttina sillan nimi, ”name”

- "del_node" poistaa tietyn kohdeinstanssin, joka määritellään argumentilla ”name”

- "pass_value" välittää yksittäisen tietopaketin valmiina olevalle kohteelle. Argumentteina kohteen nimi ”name”, sille annettava arvo ”value”, ja muunnin ”transformer”. Mikäli muutokselle ei ole tarvetta, muuntimen voi jättää pois ja [”Main”, ”pass”] muunninta käytetään automaattisesti.

- "terminate" sulkee RICOn. Ei argumentteja.

- "values" esittää käyttäjälle viimeisimmät kohteilta saadut tiedot. Ei argumentteja.

- "pass" ei tee mitään, kirjaimellisesti. Sille voi olla tarvetta testaamisessa tai edistyneemmissä skripteissä. Ei argumentteja.

- "dump" luo tiedoston, josta näkyy kaikki ohjelman sen hetkiset tiedot ihmisluettavaksi muotoiltuina, mukaan lukien kaikki lähettämistä odottava data. Sopii tilanteisiin, joissa ohjelma pitäisi sulkea, mutta tietojen katoaminen halutaan välttää.

- "status" näyttää kirjaston Data[”Sys_status”] sisällön eli tilatiedot tietyistä ohjelman alijärjestelmistä.

- "bye" tekee sitä kutsuvasta UI:sta epäaktiivisen. Lähinnä Textfile_UI:n käyttöön, kun sen Only_init -moodi on päällä.

4.2.2 Sillat

RICOssa silta on ohjelman sisäinen olio, joka hoitaa tiedon automatisoitua hakemista ja käsittelyä.

Ohjelman pääkierron aikana RICO käy läpi kaikki määritellyt siltainstanssit ja suorittaa niiden process()-metodin. Kun metodi käynnistyy, se tutkii edellisestä suorituksesta kulunutta aikaa sekä silloin saatujen arvojen eroavaisuuksia viimeisimmistä, ja niitä vertaamalla sille annettuihin parametreihin päättää, suoritetaanko kyseinen silta. Sillalle voidaan myös sitä luodessa määritellä parametreihin excluded_nodes lista. Siihen kirjatut kohteet jätetään eroavaisuusvertailun ulkopuolelle.

Mikäli silta suoritetaan, se ottaa uusimmat tiedot kohteilta, pakkaa ne kirjastoksi ja syöttää tämän kirjaston sille annetulle muuntimelle. Muunnin tekee tehtävänsä ja antaa lopullisen, lähetyskelpoisen tiedon sillalle.

Lopulta silta jakaa lähetyskelpoisen tiedon listoihin, joista se lähetetään myöhemmin ohjelmakierrossa halutuille kohteille. Tällä tavalla voidaan hyödyntää etuja, jotka mahdollisesti syntyvät tietyillä protokollilla, kun tiedot lähetetään yhtenä isona pakettina.

Silloille voidaan periaatteessa määrittää myös muita joka suoritussyklissä suoritettavia toimia. Jossain moduulissa pitää vain määritellä uudenlainen siltaluokka, joka suorittaa tehtävänsä process() -metodilla, sekä funktio halutun sillan initialisointiin.

4.2.3 Muuntimet

Muuntimet ovat funktioita, joilla määritellään, kuinka välitettävää dataa tulee muuttaa ennen lähetystä. Nämä funktiot saavat muutettavat tiedot koodiesimerkin 3 kuvaamassa muodossa.

```
{”<node>”:{””:<arvo>, ”extra_key”:<arvo 2>}, ”<node2>”:{””:<arvo 3>, ”extra_key”:<arvo 4>}}
```

Koodiesimerkki 3. Esimerkki muuntimen saamasta tietokirjastosta.

Tässä <node> ja <node2> ovat vastaanottavien kohteiden nimet ja ”extra_key” kohdat vastaavat kohteilta tulevien lisätietojen avaimia. Yhdeltä kohteelta voi halutun arvon lisäksi tulla useita muitakin tietoja kuten aikatietoja, sertifikaatteja tai metadataa. Yleisenä

käytäntönä ehdotan kuitenkin, että kohteesta annetaan avaimelle "", siis tyhjä merkkijono, arvoksi sen varsinainen arvo. Esimerkiksi jos kohteen tarkoituksena on raportoida lämpötilamittauksesta, "" arvoksi tulisi sen hetken mittalämpötila. Sama tieto voidaan tietenkin esittää tämän lisäksi normaalisti nimetyllä avaimella. Tällä tuetaan järjestelmän modulaarisuutta, kun jokaiselle muuntimelle ei tarvitse määritellä miksi mikäkin kohde kutsuu tätä arvoa. Sekä OPC UA- ja MQTT-moduulit tukevat tätä toimintoa.

Tapoja kuinka tietoa tulee lopulta muokata ei ole järkevää rajoittaa, etenkään ennen kuin ohjelman käyttösovellukset ovat tiedossa. Siksi erikseen laadittu ohjelman sisäinen muokkausjärjestelmä olisi ollut turha. Muuntimet tuleekin laatia etukäteen käyttötarkoituksen mukaan tavallisena python-koodina. Ne voi lisätä omaan moduuliinsa tai niitä todennäköisesti käyttävän moduulin mukaan. RICOssa tulee vakiona vain muuntimet ”pass” ja ”to_str” (sekä ”example”). Näistä voi olla hyötyä testauksessa, mutta niiden yksinkertaisuus rajoittaa niiden sovelluksia.

4.2.4 UI:t

UI-moduulit ovat erikoistuneita moduuleja, jotka lisäävät UI-luokan. Nimestään huolimatta nämä moduulit hoitavat myös API-liikenteen. Näille luokille luodaan jäsenet RICOn käynnistyessä ja ohjelma saa niiltä get_orders()-metodilla suoritettavat komennot jokaisen suoritussyklin aluksi. Näillä olioilla myös välitetään palautetta käyttäjälle. Näiden toimintojen lisäksi niillä on täysin sama toiminnallisuus kuin tavallisilla moduuleilla.

4.3 Toteutettuja moduuleja

4.3.1 OPCUA_Connector

OPC UA on teollisuudessa laajasti käytetty viestintäprotokolla. Koska se on erityisen yleinen anturilaitteissa, se sopii tämän projektin tietolähteeksi hyvin. Testauksessa OPC UA -palvelimena toimi henkilökohtaisella pöytäkoneella pyörivä yksinkertainen palvelin, joka satunnaisin väliajoin päivittää sisäisiä satunnaisia lukujaan sekä tiedon siitä, kauanko kyseinen ohjelma on ollut käynnissä.

Tämä moduuli on toteutettu käyttäen FreeOpcUa-kirjastoa [11]. Kyseinen kirjasto toimii pääpiirteissään, mutta sen sekava dokumentaatio ja ajoittaiset erikoisuudet aiheuttivat tähän moduuliin useita ongelmia ja sekaannuksia. Esimerkiksi get_value() ei hakenut

testipalvelimeilta haluttua arvoa, vaan ohjelmassa piti käyttää get_attribute()-funktiota. Myös tietueiden nodeId-elementtien hakemisessa oli epäselvyyksiä. Vaikka suurin osa ratkenneista vaikeuksista lopulta osoittautuikin koodaajan virheiksi, niiden jatkuva ilmaantuminen hidasti työskentelyä huomattavasti. Näistä syistä tämä projekti ei myöskään käytä OPC UA:n kehittyneempiä ominaisuuksia, kuten tilatietojen muutosilmoituksien tilausta. Riittävä toiminnallisuus kyettiin toteuttamaan ilmankin niitä.

OPCUA_Connector-moduuli lisää RICOon kyvyn muodostaa yhteyden OPC UA -palvelimiin. Yhteyden välityksellä tietueilta voidaan lukea arvoja tai muita ominaisuuksia, ja niille voidaan kirjoittaa. Koska tiedonhaku nykyisellään suoritetaan sekventiaalisesti pääohjelmassa ja joka kerta kun get_data()-metodia kutsuaan, hidas yhteys palvelimeen ja suuri määrä bridgejä voi hidastaa koko järjestelmää. Tämän takia subscriptionien käyttöönotto tai asynkronisen ratkaisun laatiminen voisivat olla hyödyllisiä jatkokehityskohteita.

RICOon OPC UA -moduuli lisää UPCUA_Connection-olion, joka nimensä mukaisesti sisältää tiedot varsinaisesta yhteydestä. Moduulin kohde taas pitää huolta yksittäisistä OPC UA -kohteista palvelimella. (Vaikka näiden kohteiden nimi OPC UA:ssa on myös ”node”, ne ovat täysin erillinen konsepti RICOn ”nodeista” eli kohteista.)

4.3.2 Google sheets

Google Sheets -moduuli kehitettiin lähinnä ohjelman testaukseen. Se lähetti annetun datan Google Sheetsiin haluttuun taulukkoon. Koska moduulin potentiaalinen tarve todettiin hyvin alhaiseksi, sen kehitys jätettiin kesken eikä se ole yhteensopiva RICOn nykyisten versioiden kanssa. Sillä oli kuitenkin suuri merkitys projektin kehityksessä ja testialustana se toimi varsin hyvin, joten se ansaitsee maininnan tässä.

4.3.3 MQTT_Connector

MQTT on keskitetty IoT-protokolla, jossa kaikki päätelaitteet keskustelevat keskeisen välittäjän kautta. Se on kevyt, skaalautuva ja luotettava. Koska viestit välitetään aiheilla, jotka ovat tekstipohjaisia, on viestintä usein myös verrattain helppolukuista.

Ohjelma pystyy kuuntelemaan MQTT-välittäjän aiheita ja lähettämään niihin dataa. Lähtevien viestien muitakin ominaisuuksia kuten QoS- ja retain-ominaisuuksia voi muokata. Tämä moduuli lisää luokat yhteyden sisältävälle oliolle ja tietyn aiheen kanssa keskustelevalle kohteelle.

Moduulin MQTT-toiminnot on toteutettu paho-MQTT-kirjastolla [12]. Tämä moduuli ei nykyisellään tue välittäjän luontia, sillä käytetty kirjasto ei sitä tue. Välittäjän luonti ei vaikuttanut tarpeeksi tärkeältä sen toteuttamiseen, mutta tämän kyvyn lisääminen vaikuttaa varsin luontevalta jatkokehityskohteelta.

MQTTn avulla voidaan myös antaa RICOlle uusia komentoja, mutta koska sen toteuttaa erillinen moduuli, sen kuvaus on kohdassa 4.3.4.

4.3.4 MQTT_UI

MQTT_UI-moduuli muodostaa heti RICOn käynnistyksessä yhteyden esiasetetulle välittäjälle ja alkaa kuuntelemaan komentoja aiheesta ”RICO/in”. Vastaukset ohjelmalta syötetään ”RICO/out”-aiheeseen. Nämä eivät voi olla sama aihe, koska muuten ohjelma joutuisi helposti kierteeseen, jossa sen omat vastaukset tulkittaisiin uusiksi komennoiksi.

Ohjelma ei itsessään tarjoa erityisiä ominaisuuksia tietoturvallisuuden eteen, koska niitä ei pidetty kehityksen kannalta erityisen tärkeinä. Tämä voisi olla hyvä jatkokehityskohteena. Kuitenkin, koska käytettävissä olevat turvatoimet riippuvat täysin käytetyn välittäjän ominaisuuksista, ei toteutus voi olla täysin yksiselitteinen vaan vaatisi realistisesti RICOn pääohjelman sisäisiä tietoturvaratkaisuja.

4.3.5 Textfile_UI

RICO ei sisällä ominaisuutta lukea komentoja komentoriviltä. Sen korvikkeena Textfile_UI-moduuli lisää kyvyn antaa komentoja input.txt-tiedoston kautta. RICOn toimiessa ohjelma tarkkailee kyseistä tiedostoa jatkuvasti, ja mikäli sille tehdään muutos, sen sisältö luetaan komennoiksi, jotka suoritetaan. Rivit, jotka alkavat #-merkillä jätetään huomiotta. Kun UI:n tulee antaa käyttäjälle informaatiota, se printataan standarditerminaaliin.

Moduuli voidaan asettaa config-moodiin, jossa se lukittuu ensimmäisen suorituskerran jälkeen eikä se tulosta terminaaliin mitään. Se sisältää myös kohteet printout_node ja stdout_node, joilla vastaanotettu data voidaan kirjoittaa terminaaliin tai tiedostoon.

5 Moduulien luominen

Uutta RICO-moduulia kirjoittaessa ensimmäisenä on päätettävä, pitääkö kyseisellä moduulilla kyetä lähettämään ohjelmalle komentoja. Mikäli tähän on tarve, tulee tiedoston alkaa rivillä

```
#RICO <nimi> UI
```

jossa <nimi> on moduulin lyhyt nimi. Tätä nimeä käytetään moduulin tunnisteena, kun ohjelmalle annetaan komentoja. Tämän lisäksi sen koodissa tulee olla luokka nimeltä UI, jonka alustaja ei vaadi argumentteja, ja luokalla tulee olla metodit get_orders() ja response(). get_orders() palauttaa kutsujalle listan sanakirjoista, joissa kuvataan ohjelman suoritettavat komennot, kuten kohdassa 4.2.1 esitetään.

Mikäli moduulia taas ei ole tarkoitettu komentojen vastaanottamiseen, tulee se aloittaa rivillä

```
#RICO <nimi> Connector
```

Jos moduuli lisää kohdetyypin, tulee sitä vastaavalla luokalla olla get_data()- ja put_data()-metodit. Molempia ei ole välttämätöntä lisätä, mikäli kohde ei tee ulkoisia kirjoitus- tai lukuoperaatioita. Silti olisi suositeltavaa lisätä ainakin tyhjää palauttavat funktiot niiden paikalle mahdollisten yhteensopivuusongelmien välttämiseksi.

Kumpikin moduulityyppi voi määrittää mielivaltaisia funktioita ja muuntimia, kuten luvun 3.3 alakohdissa määritellään. Jotta RICO tunnistaa nämä, on moduulissa oltava myös ”Functions”- ja ”Transformers”-sanakirjat, jotka yhdistävät funktiot niiden kutsumanimiin.

6 Käytännön huomioita

6.1 Virheiden käsittely

Johtuen ohjelman monipuolisuudesta ja avoimuudesta ohjelman suorituksessa on odotettavissa paljon virheitä. Tämän takia yleisenä käytäntönä virheiden hallinta on siirretty mahdollisimman korkealle suoritustasolle. Tämä suojaa ohjelmaa kaatumiselta, mutta voi tehdä virhetilanteiden juurisyiden seuraamisesta vaikeaa. Suurin osa virheistä voidaan ohittaa ja ilmoittaa käyttäjälle. Toisinaan virhe voi kuitenkin johtaa vakaviin seurauksiin ja on hyvä tunnustaa, että näissä tilanteissa ohjelman kaatuminen voi olla täysin hyväksyttävää, ellei toivottavaakin, jotta tietokato minimoidaan ja painavat ongelmat nousevat esiin.

6.2 Tietoturva

RICO ei tarjoa erityisiä tietoturvaominaisuuksia viestintäprotokollien sisäisen suojauksen ulkopuolelta. Toisaalta ohjelma on varsin läpinäkyvä, koska siitä näkee aina, minne tietoja lähetetään. Käyttäjätunnusjärjestelmä voisi olla jatkokehityskohde, mutta nykyisellään sille ei ole nähtävää tarvetta. Hätätilanteessa ohjelma on myös helppo sulkea terminate-komennolla.

6.3 Suorituskyky

Kun RICOa testattiin kotioloissa Raspberry Pillä yksinkertaisella testiskriptillä, sen yksi suoritussykli kesti alle kymmenen millisekuntia. Suurin osa tästä ajasta kului OPC UA:n lukemiseen. Eräässä testissä välitettiin hiveMQTT:n testipalvelimen kaikki tietoliikenne kiireisimpään kellonaikaan Google Sheetsiin, eikä siinä noussut esille mitään hidastusta.

Toisessa testissä RICO pystyi jälkeen jäämättä lukemaan kotikoneen OPC UA -palvelimelta keskimäärin 240 liukulukua sekunnissa ja laskemaan niistä lukujen esiintymismäärät MQTT-välittäjälle.

Kehittämisen yhteydessä ei RICOssa havaittu erityisiä ongelmia liittyen sen pitkäaikaiseen päällä olemiseen, joskin Raspberry Pi itsessään hidastui toisinaan havaittavasti oltuaan päällä viikkoja ilman uudelleenkäynnistystä, vaikka RICOa ei tuona aikana jätettykään ajamaan.

Kuten on mainittu, testiympäristönä on käytetty kotiverkkoa simuloiduilla tietolähteillä. Toiveissa oli, että järjestelmää pääsisi testaamaan muissakin ympäristöissä, mutta työn kehittämisen aikainen Covid-19-pandemian asettamat rajoitukset ja erityisen käyttökohteen puute ovat tehneet tästä vaikeaa.

Ei ole olemassa erityistä syytä, miksi RICO ei toimisi muillakin Debian-pohjaisilla tietokoneilla. Sitä ei kuitenkaan ole suunniteltu tähän, koska sille ei vaikuttanut olevan tarvetta, ja näin ollen sen yhteensopivuuteen muissa laiteympäristöissä ei projektissa ole panostettu.

7 Jatkokehitysmahdollisuudet

Projektista löytyy monta parantelua kaipaavaa kohtaa. Ensinnäkin RICOn kommunikaatiomoduuleista puuttuu ominaisuuksia, joiden toteuttaminen parantaisi käytettävyyttä huomattavasti, kuten MQTT-välittäjän luonti. Lisäksi dokumentaatiota voi aina parantaa ja kokonaisuudessa on aina pieniä säätöjä, kuten asettaa RICO käynnistymään tietokoneen käynnistyksen yhteydessä.

Moduuleilla voisi aina lisätä tukia uusille viestintätavoille, kuten sähköposti-ilmoituksille tai muille teollisen Ethernetin protokollille. Moduuleilla voisi lisätä esimerkiksi myös kehittyneempiä algoritmeja haluttujen tietolähteiden löytämiseksi ja tiedon kehittyneempään käsittelyyn. Kehittyneempi graafinen käyttöliittymäkin on ollut suunnitelmissa.

Eräs mahdollisesti hyödyllinen ominaisuus, joka tuli mieleen vasta projektin loppumetreillä, olisi ollut kyky määritellä uusia muuntimia ajon aikana vakiokomennoilla. Sen toteuttaminen tuskin tulisi olemaan täysin vaivatonta, mutta teoriassa sen pitäisi olla täysin mahdollista ja etäoperaatioissa kenties hyvinkin tärkeää.

RICOa voisi myös valmistella muihin ympäristöihin ja varmistaa, että laitteisto oikeasti toimii teollisessa ympäristössä ja eri laitteistoilla. Tähän liittyen, RICOssa ei ole erityisesti huomioitu toimintaa eristettyjen verkkojen välillä. Mikäli tietoa pitäisi esimerkiksi lähettää laitoksen verkosta internetiin, yhteyksien muodostamisessa saattaa nousta ongelmia.

Lopulta kehitettävänä olisi vielä konkreettinen käyttökohde. Koska järjestelmä on varsin joustava, kohteista ei periaatteessa pitäisi olla pulaa, mutta monessa tapauksessa ongelmiin löytyy jo tarkoituksen mukainen ratkaisu.

8 Yhteenveto

Lopputyön tuloksena syntynyt ohjelmisto on kykenevä välittämään dataa eri tietoväylien välillä, kevyt suorittaa sekä helposti muokattava. Osittain projektissa onnistuttiin kuitenkin vain, koska työlle asetetut rajoitukset olivat niin löyhät ja suurin osa ajasta kuluikin suunnan hakemiseen.

RICOssa on monta kohtaa, jotka jälkikäteen ajateltuna olisi kannattanut tehdä toisin, mutta joiden muuttaminen tässä vaiheessa olisi vaikeaa. Näihin kuuluvat UI-moduulien ja tavallisten moduulien erillisyys sekä Python-versioon 3.7 sitoutuminen. On myös hyvin todennäköistä, että projektista jäi puuttumaan monta osaa, jotka jälkikäteen huomattuna vaikuttavat itsestään selviltä.

Vaikka sille ei suoraa käyttökohdetta olekaan, RICO on tarpeeksi muovautuva, jotta sitä pystyy käyttämään hyvinkin laajalti, kunhan kehittää sopivat lisäosat. Kuten alkuperäisissä suunnitelmissa ehdotettiin, ohjelmistoa voidaan käyttää esimerkiksi laitteiston etävalvontaan tai laitteiden asetusten automatisoituun asettamiseen. Henkilökohtaisina jatkokehityskohteina harkinnassa on ollut sovittaa ohjelma eräänlaiseksi puhelinvastaajaksi tai 3d-printterin valvontaan.

Lähteet

1 Pinjan esittely https://www.pinja.com/me-olemme-pinja/ Luettu 11.1.2021

2 2019 ohjelmistokehittäjien populaatioraportti https://slashdata-website-cms.s3.amazonaws.com/sample_reports/Ei-WEyM5bfZe1Kug_.pdf Luettu 30.12.2020

3 Python-instituutin kuvaus https://pythoninstitute.org/what-is-python/ Luettu 30.12.2020

4 Python-ohjelmoijien kyselyn tulokset 2019 https://www.jetbrains.com/lp/python-developers-survey-2019/ Luettu 30.12.2020

5 Katsaus Raspberry Pin tietoturvaan https://www.researchgate.net/publication/336966872_Security_Vulnerabilities_in_Raspberry_Pi-Analysis_of_the_System_Weaknesses Luettu 30.12.2020

6 Raspberry Pi 4 model B https://www.raspberrypi.org/products/raspberry-pi-4-model-b/ Luettu 30.12.2020

7 ISA:n artikkeli OPC UA:sta https://www.isa.org/intech-home/2019/november-december/features/opc-ua-the-united-nations-of-automation Luettu 31.12.2020

8 OPC UA:n esittelysivu https://opcfoundation.org/about/opc-technologies/opc-ua/ Luettu 31.12.2020

9 Infosivu Google Sheetsin API-ominaisuuksista https://developers.google.com/sheets/api Luettu 31.12.2020

10 HiveMQn julkinen MQTT-välittäjä testaukseen http://www.hivemq.com/demos/websocket-client/? Luettu 31.12.2020

11 Käytetty OPC UA -kirjasto https://github.com/FreeOpcUa/python-opcua Luettu 31.12.2020

12 Käytetty MQTT-kirjasto https://pypi.org/project/paho-mqt Luettu 31.12.2020

Main.py

```
from re import match
import sys, importlib, time, json, os

"""---Configuration---"""
Commands_status_track_limit = 10 # How many latest commands the pro-
gram keeps in memory. Cannot be changed after boot
Execution_brake_time = 0 #Delay after each cycle in seconds. Increase
in case the program hoggs too much CPU time

#---Internal Classes---
class Dummy_UI:
    def __init__(self):
        pass
    def get_orders(self):
        return []
    def response(self, text):
        pass

class bridge:
    def __init__(self, transformer, nodes_in, nodes_out, update_dep,
delay, excluded_nodes):
        """Entity that connects inbound and outbound nodes and facili-
tates required automatic processes"""
        self.last_val = {name{} for name in nodes_in}
        self.last_time = 0
        self.bridge_data = {}

        self.nodes_in = nodes_in
        self.nodes_out = nodes_out

        self.transformer = transformer
        self.update_dep = update_dep
        self.delay = delay
        self.excluded_nodes = {node : "" for node in excluded_nodes}

        def process():
            """Check how long ago the last update was."""
            if (time.time() - self.last_time) >= self.delay:

                """Check whether data has changed."""
                for i in self.nodes_in:
                    self.bridge_data.update({i:
nodes_in[i].get_data().copy()})

                """Exclude values marked as timestamps etc. from com-
parisons, if only_new_data is true"""
                cut_bridge_data = self.bridge_data
                cut_bridge_data.update(self.excluded_nodes)

                if self.last_val != cut_bridge_data or not update_dep:
                    """Transforms data and directs it to appropriate
datalists of outbound nodes"""
                    self.last_val = cut_bridge_data.copy()
                    self.last_time = time.time()
                    sready_data = self.transformer(self.bridge_data)
                    global Datalists
                    if sready_data:
```

```
                        for o in self.nodes_out:
                            try:
                                Datalists[o].append(sready_data)
                            except KeyError:
                                Datalists[o] = [sready_data]

        self.process = process

#---Internal functions---
def Load_Modules():
    """Loads modules for sendinf and receiving data and commands."""
    Mods_Path = sys.path[0] + "/Mods"

    global Data
    """Searches through all files in /Mods"""
    for File in [f for f in os.listdir(Mods_Path) if
os.path.isfile(Mods_Path + "/" + f)]:
        path = Mods_Path + "/" + File
        file = open(path)

        """Determines if a file is a basic connector or an UI (or
API)"""
        fline = file.readline()
        IO = match("^#RICO .* Connector$", fline)
        UI = match("^#RICO .* UI$", fline)

        if IO:
            ID = IO.group()[6:-10]
            spec = importlib.util.spec_from_file_location(ID, path)
            mod = importlib.util.module_from_spec(spec)
            spec.loader.exec_module(mod)
            Data["Mods"][ID] = mod

            #Add External Functions
            try:
                Data["Functions"][ID] = mod.Functions
            except:
                pass

            #Add External Transformers
            try:
                Data["Transformers"][ID] = mod.Transformers
            except:
                pass

        elif UI:
            ID = UI.group()[6:-3]
            spec = importlib.util.spec_from_file_location(ID, path)
            ui = importlib.util.module_from_spec(spec)
            spec.loader.exec_module(ui)
            Data["UIs"][ID] = ui.UI()

            #Add External Functions
            try:
                Data["Functions"][ID] = ui.Functions
            except:
                pass

            #Add External Transformers
            try:
```

```
                Data["Transformers"][ID] = ui.Transformers
            except:
                pass
        else:
            pass

        file.close()

def do_command(UI, c):
    """Wrapper for executing commands."""
    global Data

    command = {"module":None, "function":None, "args":{}}
    command.update(c)

    #If no module is given, just for qol
    stop = False
    if command["module"] == None:
        for a in Data["Functions"]:
            for b in Data["Functions"][a]:
                if b == command["function"]:
                    command["module"] = a
                    stop = True
                    break
            if stop:
                break

    r = Data["Functions"][command["module"]][command["function"]](UI,
Data, command["args"])
    if r:
        UI.response(r)

def get_transformer(tr):
    global Data
    #If no module is given, apropriate one is searched
    if type(tr) == str:
        tr = [tr]

    if len(tr) == 1:
        stop = False
        for a in Data["Transformers"]:
            for b in Data["Transformers"][a]:
                if b == tr:
                    tr = [a,b]
                    stop = True
                    break
            if stop:
                break
        raise Exception("Transformer specified invalidly: " + str(tr))

    return  Data["Transformers"][tr[0]][tr[1]]

#---USER ACCESSIBLE COMMANDS & TRANSFORMERS START---
def pass_value(UI, Data, args_in):
    """
    {"module":"Main", "function":"pass_value", "args":{"name": "",
"value":"", "transformer":["Main", "pass"]}
    Passes a single value to an existing node.
    If transformer is left blank, ['Main', 'pass'] will be used.
    """
```

```
    args = {"transformer":["Main","pass"]}
    args.update(args_in)

    Data["Nodes"][args["name"]].put_data([get_transformer(args["trans-
former"])(args["value"])])
    return("Value " + str(args["value"]) + " passed to node " +
args["name"])

def show_vals(UI, Data, args):
    """
    {"module":"Main", "function":"values", "args":{}}
    Gives all latest messages to receiving nodes, no arguments needed
    """
    UI.response(str(Datalists))

def show_memory(UI, Data, args):
    """"
    {"module":"Main", "function":"memory", "args":{}}
    Outputs the state of the system, no arguments needed
    """
    UI.response("Current_UI: " + str(UI))
    UI.response("UIs: " + str(Data["UIs"]))
    UI.response("Mods:" + str(Data["Mods"]))
    UI.response("Entities: " + str(Data["Entities"]))
    UI.response("Nodes: " + str(Data["Nodes"]))
    UI.response("Bridges: " + str(Data["Bridges"]))
    UI.response("Functions: " + str(Data["Functions"]))

def new_bridge(UI, Data, args_in):
    """
    {"module":"Main", "function":"new_bridge", "args":{"name": "",
"only_new_data":true, "delay":1, "nodes_in":[], "nodes_out":[],
"transformer":[]}}
    To modify a bridge, use this function to overwrite the old one.
    If no changes in data are required, use ['Main','pass'] as the
transformer
    """
    args = {"only_new_data":True, "delay":0, "transformer":None, "ex-
clude":[]}
    args.update(args_in)

    nodes_in = {}
    for ni in args["nodes_in"]:
        nodes_in.update({ni: Data["Nodes"][ni]})

    nodes_out = {}
    for no in args["nodes_out"]:
        nodes_out.update({no: Data["Nodes"][no]})

    Data["Bridges"].update({args["name"]:bridge(get_trans-
former(args["transformer"]), nodes_in, nodes_out,
args["only_new_data"], args["delay"], args["exclude"])})
    return("Created bridge " + args["name"])

def del_bridge(UI, Data, args):
    """
    {"module":"Main", "function":"del_bridge", "args":{"name":""}}
    Deletes a specified bridge
    """
    del Data["Bridges"][args["name"]]
```

```
    return("Deleted bridge " + args["name"])

def del_node(UI, Data, args):
    """
    {"module":"Main", "function":"del_node", "args":{"name":""}}
    Deletes a specified node.
    """
    del Data["Nodes"][args["name"]]
    return("Deleted node " + args["name"])

def disable_UI(UI, Data, args):
    """
    {"module":"Main", "function":"bye", "args":{}}
    Disables calling UI. Can only be reversed by rebooting program.
Cannot stop other UIs.
    It almost makes sense, trust me.
    """
    UI_name = ""
    for name, ent in Data["UIs"].items():
        if UI == ent:
            UI.response("Disabling this UI")
            UI_name = name

    if name:
        Data["UIs"].update({name + "---Disabled":Dummy_UI()})
        del Data["UIs"][name]
    else:
        raise Exception("Cannot terminate this UI, this UI cannot be
found... ???")

def terminate(UI, Data, args):
    """
    {"module":"Main", "function":"terminate", "args":{}}
    Nukes the program. Consider dumping the progam data before using
    """
    quit()

def pass_function(UI, Data, args):
    """
    {"module":"Main", "function":"pass", "args":{}}
    Does nothing, here just to perplex you by its existence
    """
    pass

def show_values(UI, Data, args):
    """
    {"module":"Main", "function":"values", "args":{}}
    Presents the last sent values to all outputting nodes
    """
    UI.response(str(Data["Last_values"]))

def dump_to_file(UI, Data, args):
    """
    {"module":"Main", "function":"dump", "args":{"name":""}}
    Dumps system memory to a file. If no filename is given, one will
be generated
    """
    try:
        name = args["name"]
        if not name:
```

```
            raise Exception("No dump file name given. One will be gen-
erated, this exception can be ignored.")
    except:
        name = ("rico_log_"+time.strftime("%Y-%m-%d-%H-%M-%S",
time.gmtime()))

    name = sys.path[0]+"/Logs/"+name
    """formats the ist to semi-readable form"""
    d = 0
    string = ""
    for c in str(Data):
        if c == "{":
            d += 1
            string += "\n" + "  "*d + c

        elif c == ",":
            string += c + "\n" + "  "*d

        elif c == "}":
            d -= 1
            string += "\n" + "  "*d + c

        else:
            string += c

        if d < 0:
            d=0

    f = open(name, "w")
    f.write(string)
    f.close
    return("Dump file created! " + name)

def sys_status(UI, Data, args):
    """
    {"module":"Main", "function":"status", "args":{}}
    Presents status information from certain internal processes
    """
    UI.response(str(Data["Sys_status"]))

    #Following transformers are for demonstration, testing and example
only.
def pass_transformer(raw):
    return raw

def to_str_transformer(raw):
    return str(raw)

def example_transformer(raw):
    try:
        r = "Random: " + str(raw["OPCUA_Node"][""]) + ", Message:" +
str(raw["MQTT_in"][""].decode('utf-8') + ", Time:" + str(time.time()))
        return(r)
    except KeyError:
        return(None)

#---USER ACCESSIBLE COMMANDS END---
#---Main---
Data = {
        "UIs":       {},
```

```
        "Mods":       {},
        "Entities":   {},
        "Nodes":      {},
        "Bridges":    {},
        "Last_values":{},
        "Sys_status":{
                      "Sending":{},
                      "Commands":[""]*Commands_status_track_limit
                      },
        "Functions": {
                      "Main":{
                              "memory":     show_memory,
                              "new_bridge": new_bridge,
                              "del_bridge": del_bridge,
                              "del_node":   del_node,
                              "pass_value": pass_value,
                              "terminate":  terminate,
                              "values":     show_values,
                              "pass":       pass_function,
                              "dump":       dump_to_file,
                              "status":     sys_status,
                              "bye":        disable_UI
                              }
                      },
        "Transformers": {#robots in disguise
                      "Main":{
                              "pass":    pass_transformer,
                              "to_str":  to_str_transformer,
                              "example": example_transformer
                              }
                      },
        "Configuration":{
                      "Brake" : Execution_brake_time
                      }
        }

Datalists = {}
Send_fail = []
Load_Modules()
for UI in Data["UIs"]:
    Data["UIs"][UI].response("RICO loaded, Modules: " +
str(list(Data["Mods"].keys())) + " ; UIs: " +
str(list(Data["UIs"].keys())))

#Main loop starts
while 1:
    #Collects incoming commands and executes them
    for UI in Data["UIs"].values():
        for c in UI.get_orders():
            try:
                do_command(UI, c)
                Data["Sys_status"]["Commands"].insert(0, "Executed
command " + str(c))
                Data["Sys_status"]["Commands"].pop()
            except SystemExit:
                for UI in Data["UIs"]:
                    Data["UIs"][UI].response("RICO terminated by a
command")
                quit()
            except:
```

```
                error = "Error executing command " + str(c) +
str(sys.exc_info())
                Data["Sys_status"]["Commands"].insert(0, error)
                Data["Sys_status"]["Commands"].pop()
                UI.response(error)

    #Clears data-to-be-sent from nodes where sending was succesfull.
Also updates latest values pool
    for e in Datalists:
        if Datalists[e]:
            Data["Last_values"].update({e:Datalists[e]})
        if e not in Send_fail:
            Datalists[e] = []
    Send_fail = []

    #Ćollects data
    for i in Data["Bridges"].values():
        i.process()

    #Send collected data
    for i in Datalists:
        if Datalists[i]:
            try:
                Data["Nodes"][i].put_data(Datalists[i])
                Data["Sys_status"]["Sending"].update({i:"Send suc-
cesfull at time " + time.asctime()})
            except:
                Send_fail.append(i)
                Data["Sys_status"]["Sending"].update({i:"Error sending
data to "+ i + str(sys.exc_info())})

    time.sleep(Data["Configuration"]["Brake"])
```

Textfile_UI.py

```
#RICO Textfile UI
from re import match, search
import json, os, sys

#If set true, updating file doesn't effect the program and this UI
doesn't print anything
Only_init = False
path = "/home/pi/Desktop/Lopputyo/input.txt"
class UI:
    def __init__(self):
        self.path = path
        self.FileEnt = open(path,"r")
        self.last_time = 0
        self.cmds = []

    def response(self, text):
        if not Only_init:
            print("FileUI: ", text)

    def get_orders(self):
        self.cmds = []

        time = os.stat(self.path).st_mtime
        if time == self.last_time:
            return([])
        else:
            self.last_time = time
            pass

        for line in self.FileEnt:
            if line[:1] != "#":
                try:
                    self.cmds.append(json.loads(line))
                except:
                    self.response("Error on input row: "+ line +
str(sys.exc_info()))

        self.FileEnt.seek(0)

        if Only_init:
            self.cmds.append({"module":"Main", "function":"bye",
"args":{}})
            self.FileEnt.close()

        return(self.cmds)

class stdout_node():
    def __init__(self):
        pass

    def put_data(self, data):
        print(str(data))

class printout_node():
    def __init__(self, args):
        self.filename = args["filename"]
```

```
        self.file = open(args["filename"], "a")

    def __del__(self):
        self.file.close()

    def put_data(self, data):
        self.file.write(str(data)+"\n")

def new_stdout_node(UI, Data, args_in):
    """
    {"module":"Textfile", "function":"new_stdout_node",
"args":{"name": ""}}
    Sets new node that when receiving data, prints it in console.
    """
    Data["Nodes"].update({args_in["name"]: stdout_node()})

def new_printout_node(UI, Data, args_in):
    """
    {"module":"Textfile", "function":"new_printout_node",
"args":{"name": "", "filename": ""}}
    Sets new node that when receiving data, prints it in a file de-
fined dy filename.
    """
    Data["Nodes"].update({args_in["name"]: printout_node(args_in)})


Functions = {"new_stdout_node":new_stdout_node,
"new_printout_node":new_printout_node}
```

OPCUA_Connector.py

```
#RICO OPCUA Connector
import sys
from time import sleep
import opcua

#---Functions---
def new_node(UI, Data, args_in):
    """
    {"module":"OPCUA", "function":"new_node", "args":{"name": "",
"connection": "", "nodeid":"", "keys":[]}}
    Creates a new OPCUA Client. A OPCUA Connection must be created
first and given as an argument to this here function.
    """

    args={"keys":[13]}
    args.update(args_in)

    try:
        Data["Nodes"][args["name"]] = OPCUA_Node(Data["Enti-
ties"][args["connection"]], args["nodeid"], args["keys"])
        return("Created an OPC UA node " + args["name"])
    except KeyError:
        return("Error making node {}! This function requires a name,
connection and a nodeid to be defined.".format(args["name"]))
    except:
        return("Error making node {}! ".format(args["name"]) +
str(sys.exc_info()))

def new_connection(UI, Data, args_in):
    """
    {"module":"OPCUA", "function":"new_connection", "args":{"name":
"", "ip":"", "user":"" ,"password": ""}}
    Creates a new OPCUA Connection.
    """

    args={"user":"","password":""}
    args.update(args_in)

    try:
        Data["Entities"][args["name"]] = OPCUA_Connection(args["ip"],
args["user"], args["password"])
        return("Created OPC UA connection " + args["name"])
    except KeyError:
        return("Error making connection {}! This function requires
both a name and ip to be defined.".format(args["name"]))
    except:
        return("Error making connection {}! ".format(args["name"]) +
str(sys.exc_info()))

def node_set_keys(UI, Data, args):
    """
    {"module":"OPCUA", "function":"node_set_keys", "args":{"name": "",
"keys":[]}}
    Sets new keys for an existing OPCUA node.
    """
```

```
    Data["Nodes"][args["name"]].set_keys(args["keys"])
    return("Set keys "+ str(args["keys"]) +" for node " +
args["name"])

class OPCUA_Node:
    def __init__(self, connection, nodeid, keys):
        self.connection = connection
        self.nodeid = nodeid
        self.keys = keys
        self.node = self.connection.client.get_node(self.nodeid)

    def _try_value(self, attr):
        try:
            return attr.Value.Value.to_string()
        except:
            return attr.Value.Value

    def set_keys(self, keys):
        self.keys = args

    def get_data(self):
        #Gets the data
        values = {}
        try:
            values = {"": self._try_value(self.node.get_attrib-
ute(13))}
        except:
            pass
        for i in self.keys:
            values.update({i : self._try_value(self.node.get_attrib-
ute(int(i)))})

        return values

    def put_data(self, values):
        self.connection.client.set_values([self.node], values)

class OPCUA_Connection:
    def __init__(self, ip, user="", passw=""):

        self.ip = ip
        self.us = user
        self.pw = passw

        self.client = opcua.Client(self.ip)
        self.client.connect()

    def __del__(self):
        self.client.close_session()

Functions = {
    "new_node": new_node,
    "new_connection": new_connection
    }
```

MQTT_UI.py

```
#RICO MQTT_UI UI
import paho.mqtt.client as mqtt
import json

Broker_IP = "broker.mqttdashboard.com"

#These must not be the same!
in_topic = "RICO/in"
out_topic = "RICO/out"

Functions = {}
Transformers = {}

class UI:
    def __init__(self):
        self.commands = []
        self.in_topic_string = in_topic
        self.out_topic_string = out_topic
        self.subscribe = True
        self.qos = 2
        self.retain = False
        self.ip = Broker_IP

        self.in_client = mqtt.Client()
        self.in_client.connect(self.ip)
        if self.out_topic_string:
            self.out_client = mqtt.Client()
            self.out_client.connect(self.ip)

        def on_message(client, userdata, msg):
            try:
                msg = json.loads(msg.payload.decode('utf-8'))
                self.commands.append(msg)
            except:
                self.out_client.publish(self.out_topic_string, "Inva-
lid command "+str(msg.payload), self.qos, self.retain)

        self.in_client.subscribe(self.in_topic_string)
        self.in_client.on_message = on_message
        self.in_client.loop_start()

    def get_orders(self):
        ret_coms = self.commands
        self.commands = []

        if not ret_coms:
            ret_coms = []

        return(ret_coms)

    def response(self, text):
        if self.out_topic_string:
            self.out_client.publish(self.out_topic_string, text,
self.qos, self.retain)
```

MQTT_Connector.py

```
#RICO MQTT Connector
import paho.mqtt.client as mqtt

#---Functions---
def new_client(UI, Data, args):
    """
    {"module":"MQTT", "function":"new_client", "args":{"name": "",
"IP":""}}
    Creates a new MQTT Client.
    """
    Data["Entities"][args["name"]] = MQTT_Client(args["ip"])
    return("Created MQTT client " + args["name"])

def new_topic(UI, Data, args):
    """
    {"module":"MQTT", "function":"new_topic", "args":{"name": "",
"client": "", "topic": "", subscribe: False, qos: 2, retain: False}}
    Creates a new MQTT topic pointer and optionally subscription to
said topic. A MQTT client must be made before using tins function.
    """
    Data["Nodes"][args["name"]] = MQTT_Topic(Data["Enti-
ties"][args["client"]].client, args["topic"], subscribe =
args.get('subscribe', False), qos = args.get('qos', 2), retain =
args.get('retain', False))
    return("Created MQTT topic " + args["name"])

#---Classes---
class MQTT_Client:
    def __init__(self, ip):
        self.client = mqtt.Client()
        self.ip = ip
        self.client.connect(self.ip)

    def __exit__(self):
        client.disconnect()
        pass

class MQTT_Topic:
    def __init__(self, client, topic_string, subscribe = False, qos =
2, retain = False):
        self.data_output = {}
        self.client = client
        self.topic_string = topic_string
        self.subscribe = subscribe
        self.qos = qos
        self.retain = retain

        def on_message(client, userdata, msg):
            try:
                append_data = {"":msg.payload}
            except:
                append_data = {"":""}

            items = ["dup", "info", "mid", "payload", "properties",
"qos", "retain", "state", "timestamp", "topic"]
            for i in items:
                try:
```

```
                    append_data[i] = msg.__getattribute__(i)
                except:
                    pass
            self.data_output.update(append_data)

        if subscribe:
            try:
                client.subscribe(topic_string)
                client.on_message = on_message
                client.loop_start()
            except:
                pass

    def get_data(self):
        return self.data_output

    def put_data(self, data):
            for row in data:
                self.client.publish(self.topic_string, row, self.qos,
self.retain)


#---User accessible funstions---
Functions = {
    "new_client": new_client,
    "new_topic": new_topic
    }

Transformers = {}
```