Avoimen lähdekoodin reaaliaikaiset käyttöjärjestelmät

Turun yliopisto
Tietotekniikan laitos
LuK-tutkielma
Tietojenkäsittelytiede
Huhtikuu 2026
Frans Mörönen

TURUN YLIOPISTO
Tietotekniikan laitos

Frans Mörönen: Avoimen lähdekoodin reaaliaikaiset käyttöjärjestelmät

LuK-tutkielma, 26 s.
Tietojenkäsittelytiede
Huhtikuu 2026

Droonit ja IoT-laitteet ovat sulautettuja järjestelmiä, jotka toimivat usein reaaliaikaisilla käyttöjärjestelmillä. Ne on varustettu rajallisella laitteistolla, jossa laskentateho, muistikapasiteetti ja virtalähde ovat rajatut. Avoimen lähdekoodin reaaliaikaiset käyttöjärjestelmät ovat nopeuttaneet näiden järjestelmien kehitystä. Avoimen lähdekoodin kehityspaketti tarjoaa kehittäjille valmiit työkalut ja kirjastot, mutta edellyttää lisenssien yhteensopivuutta niiden hyödyntämiseksi.

Tässä kirjallisuuskatsauksessa tarkastellaan avoimen lähdekoodin reaaliaikaisia käyttöjärjestelmiä resurssirajoitteisille sulautetuille järjestelmille. Tutkielmassa perehdytään reaaliaikaisten ja yleiskäyttöisten käyttöjärjestelmien välisiin eroihin, arkkitehtuurillisia ratkaisuja rajallisen laitteiston tuomiin haasteisiin sekä avoimen lähdekoodin ohjelmistojen lisensoinnin rooliin ja vaikutukseen projektin kehityksessä. Tutkielman tarkastelu painottuu droonikontekstin ja lennonohjausjärjestelmien ympärille.

Tutkielmassa löydetään reaaliaikaisen ja yleiskäyttöisen käyttöjärjestelmän välille yksiselitteiset erot, mutta pehmeän ja kovan reaaliaikaisuuden käyttöjärjestelmien välinen raja määritellään kirjallisuudessa häilyvämmin. Tutkielma esittää neljä arkkitehtuurista ratkaisua rajallisen laitteiston haasteille: ChibiOS:n pienikokoisen ytimen, Zephyrin modulaarisen arkkitehtuurin, yhdistetyn reaaliaikaisen käyttöjärjestelmän sekä micro-ROS:n avulla mikrokontrollerien tehokkaan käyttämisen. Tuloksista selviää, ettei rajallisen laitteiston asettamia haasteita voida ratkaista vain yhdellä tavalla, vaan ratkaisun tulee olla räätälöity käyttötarkoitusta ajatellen. Lisenssin valinta vaikuttaa koko projektin elinkaareen ja lisenssien yhteensopimattomuus asettaa lisähaasteita.

Asiasanat: reaaliaikainen käyttöjärjestelmä, sulautetut järjestelmät, avoin lähdekoodi, rajallinen laitteisto, lisensointi, drooni

Sisällys

1 Johdanto

Sulautetuilta järjestelmiltä vaaditaan yhä vaativampia ominaisuuksia, mutta ne on usein varustettu rajallisella laitteistolla. Droonit ja IoT-laitteet (Internet of Things) edellyttävät reaaliaikaista käyttöjärjestelmää (Real-Time Operating System, RTOS) varmistamaan tehtävien suorittamisen tarkkojen aikamääreiden sisällä [1] [2]. Reaaliaikaiset käyttöjärjestelmät ovat deterministisiä, mikä erottaa ne yleiskäyttöisistä käyttöjärjestelmistä (General-Purpose Operating Systems, GPOS), sillä jälkimmäiseksi mainittu voi uhrata prosessien suoritusaikaa käyttökokemuksen parantamiseksi [3].

Reaaliaikaiset käyttöjärjestelmät toimivat usein rajallisilla resursseilla varustetuilla laitteilla [1]. Näitä ovat pieni muistikapasiteetti, matalat kellotaajuudet ja rajoitettu energian saanti, kuten akut tai paristot. Resurssirajoitteiselta alustalta reaaliaikainen käyttöjärjestelmä vie resursseja entisestään. Ytimen koolla, käyttöjärjestelmän modulaarisuudella ja resurssien hallinnan tehokkuudella vaikutetaan sovellusohjelmistolle, kuten lennonohjausjärjestelmälle (Flight Control Unit) käytettäväksi jäävien resurssien määrään [4] [5]. Lennonohjausjärjestelmissä mikrokontrolleri käsittelee rajallisella laitteistolla sensoridataa ja ohjaa droonin moottoreita tiukkojen aikamäärien sisällä reaaliaikaisesti.

Avoimen lähdekoodin kehityspaketit, kirjastot ja käyttöjärjestelmät ovat tasanneet kilpailua, antaneet pienemmille yrityksille mahdollisuuden kehittää sulautettuja järjestelmiä ja nopeuttaneet kehitystä [6] [7]. Avoimen lähdekoodin ohjelmistojen

avoimuus varmistetaan lisensoinnilla, mikä ohjaa näiden ohjelmistojen käytettävyyttä [8]. Ne määrittävät ohjelmiston levittämisen, muokkaamisen ja yhdistelemisen ehdot, eivätkä kaikki lisenssit ole keskenään yhteensopivia. Lisenssien yhteensopimattomuus tuo haasteita projektien kehittämiseen jo alkuvaiheessa, sillä lisenssien valinta tulee vaikuttamaan projektiin sen koko elinkaaren ajan [9].

Tässä tutkielmassa keskitytään rajalliselle laitteistolle suunnattuihin avoimen lähdekoodin reaaliaikaisiin käyttöjärjestelmiin. Tutkielma on kirjallisuuskatsaus, jossa aihetta tarkastellaan kolmella tutkimuskysymyksellä:

TK1: Mitä reaaliaikaiset käyttöjärjestelmät ovat ja kuinka ne eroavat yleiskäyttöisistä käyttöjärjestelmistä?

TK2: Millaisilla ratkaisuilla avoimen lähdekoodin reaaliaikaisissa käyttöjärjestelmissä on vastattu rajallisen laitteiston tuomiin haasteisiin?

TK3: Miten avoimen lähdekoodin ohjelmistojen kehittäminen ja levittäminen mahdollistetaan lisensseillä?

Tutkielmassa tarkastellaan tarkemmin drooneja rajallisen laitteiston osalta. Tutkielman kirjallisuus keskittyy näihin sovelluksiin reaaliaikaisten käyttöjärjestelmien osalta. Vaikka tarkasteltu kirjallisuus ei tutkielmassa kata muita rajatulla laitteistolla varustettuja sulautettuja järjestelmiä, on niitä kuitenkin olemassa. Esimerkiksi lääkinnällisille laitteille on tarkat vaatimukset niiden toiminnan osalta ja ne on usein varustettu rajallisella laitteistolla. Tutkielmassa ei myöskään tarkastella avoimen lähdekoodin ohjelmistoja suurella laskentateholla ja muistilla varustetuissa laitteissa, kuten ajoneuvoissa.

Tutkielma toteutettiin kirjallisuuskatsauksena. Tarkasteltu kirjallisuus valikoitui avoimen lähdekoodin osalta lisensointia käsitteleviin artikkeleihin. Reaaliaikaisten käyttöjärjestelmien osalta suljetuista ohjelmistoista löytyi heikosti tutkimusta, joten aiheksi valittiin avoimeen lähdekoodiin pohjautuviin käyttöjärjestelmiin. Tutkielman kirjallisuus on pääsääntöisesti vuodelta 2022 ja sitä uudempaa. Tämä vai-

kuttaa tutkittuun sovellusalueseen, miehittämättömiin drooneihin ja lennonohjausjärjestelmiin, sillä niiden kehitys on ollut viime vuosina nopeaa.

Tutkielma koostuu viidestä luvusta, joista luvussa 2 määritellään keskeiset käsitteet, kuten reaaliaikainen ja yleiskäyttöinen käyttöjärjestelmä, kova ja pehmeä reaaliaikaisuus, reaaliaikaisten käyttöjärjestelmien komponentit, perehdytään avoimen lähdekoodin ohjelmistojen lisensseihin ja kehitysalustoihin sekä määritellään rajallinen laitteisto ja lennonohjausjärjestelmät. Luvussa 3 vertaillaan avoimen lähdekoodin reaaliaikaisia käyttöjärjestelmiä, perehdytään tarkemmin Zephyriin, ChibiOS:ään ja Apache NuttX:ään ja tarkastellaan rajallisen laitteiston tuomiin haasteisiin kehitettyjä ratkaisuja. Luku 4 käsittelee tutkielman havaintoja pohdintana. Luku 5 päättää tutkielman yhteenvedolla ja vastaa johdannossa esiteltyihin tutkimuskysymyksiin.

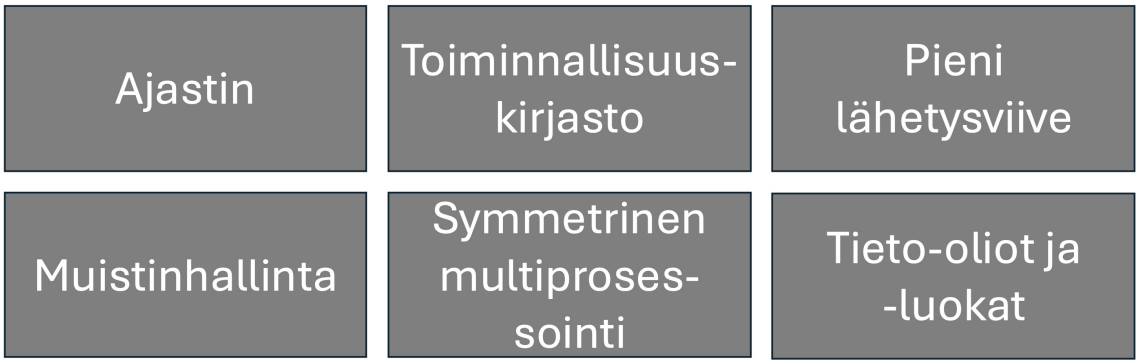
2 Tausta ja käsitteet

Yleiseen käyttöön tarkoitetuilla käyttöjärjestelmillä on lähtökohtaisesti erilainen arkkitehtuuri, kuin reaaliaikaisilla käyttöjärjestelmillä. Lisäksi kovan ja pehmeän reaaliaikaisuuden käyttöjärjestelmillä on erilaiset vaatimukset, joten niiden arkkitehtuurin välillä voi olla suuriakin eroja. Avoimen lähdekoodin ohjelmistojen päätymistä suljetuiksi suojataan lisenssein. Näillä lisensseillä on tarkat ehdot, eikä niitä voi käyttää yhdessä kaikkien muiden lisenssien kanssa. Lisensoinnin valinta on tärkeä osa koko projektin elinkaarta, sillä se määrittelee sille käytössä olevat työkalut yhteensopivuudellaan. Sulautettujen järjestelmien kehitykseen tarkoitetut ohjelmistokehykset, kuten micro-ROS, ovat helpottaneet sulautettujen järjestelmien kehitystä. Ohjelmistokehys robot operating system (ROS) on helpottanut etenkin liikkuvien robottien kehitystä.

2.1 Käyttöjärjestelmien määrittely

Reaaliaikainen käyttöjärjestelmä on prosessorilla ajettava matalan tason ohjelmisto, joka ajoittaa prosesseja, hallitsee oheislaitteita sekä takaa järjestelmän vakauden, hallittavuuden ja turvallisuuden [1]. Perinteisesti sulautetut järjestelmät ajavat matalan tärkeystason tehtäviä ja korkean tärkeystason tehtävät suoritetaan välissä keskeytyksinä. Reaaliaikaisilla käyttöjärjestelmillä on tarkat aikavaatimukset ja niiden tulee olla ennakoitavia, mikä tekee niistä reaaliaikaisia. Käyttöjärjestelmän täyttäessä reaaliaikaisen käyttöjärjestelmän vaatimukset täydellisesti sitä kutsutaan kovan

reaaliaikaisuuden käyttöjärjestelmäksi (Hard RTOS, HRTOS). Käyttöjärjestelmää, joka on ennakoitavissa, mutta ei keskeytä toimintaa virhetilanteessa, eli ei aina täytä sille asetettuja aikavaatimuksia kutsutaan vastaavasti pehmeän reaaliaikaisuuden käyttöjärjestelmäksi (Soft RTOS, SRTOS) [2]. Reaaliaikaiset käyttöjärjestelmät suorittavat tehtävät tärkeysjärjestyksessä, mikä erottaa ne yleiskäyttöisistä käyttöjärjestelmistä. Reaaliaikaisella käyttöjärjestelmällä tehtävien suorittamisjärjestys olisi jokaisella suorituskerralla samanlainen, jos sille annetut tehtävät pysyvät identtisinä. Tämä tekee järjestelmistä ennakoitavia. Reaaliaikainen käyttöjärjestelmä koostuu kuvassa 2.1 esitellyistä ominaisuuksista.



Kuva 2.1: Reaaliaikaisen käyttöjärjestelmän komponentit, perustuen [10]

Reaaliaikaiseen käyttöjärjestelmään kuuluu pääsääntöisesti kuusi komponenttia tai ominaisuutta: ajastin (aikatauluttaja, eng. scheduler), toiminnallisuuskirjasto (Function Library), pieni lähetysviive (Fast Dispatch Latency), käyttäjän määrittämät tieto-oliot ja -luokat (User-defined Data Objects and Classes), muistinhallinta (Memory Management) ja symmetrinen multiprosessointi (Symmetric Multiprocessing) [10]. Ajastin asettaa komponentit suoritettaviksi jonoon tärkeysjärjestyksessä. Toiminnallisuuskirjasto tarjoaa rajapinnan suorittimen ja ohjelmakoodin välillä. Järjestelmän lähetysviive on keskimääräinen aika, jossa reaaliaikainen käyttöjärjestelmä suorittaa jonossa olevan tehtävän. Käyttäjän määrittämät tieto-oliot ja -luokat

ovat ohjelmakoodissa määriteltyjä ominaisuuksia, joita reaaliaikainen käyttöjärjestelmä tulee suorittamaan. Muistinhallinnassa reaaliaikainen käyttöjärjestelmä osittaa muistia jokaiselle ohjelmalle, jotta niiden suoritus on mahdollista. Symmetrinen multiprosessointi on kokoelma yksittäisiä tehtäviä, joita reaaliaikainen käyttöjärjestelmä voi suorittaa yhtäaikaisesti.

Yleiskäyttöiset käyttöjärjestelmät ovat käyttöjärjestelmiä, joiden tarkoituksena on suoriutua monipuolisista tehtävistä. Yleiskäyttöiset käyttöjärjestelmät on suunniteltu erilaisten sovellusten käyttämiseen, eikä sillä ole aikavaatimuksia tehtävän suorittamiseen. Näillä käyttöjärjestelmillä ei ole mahdollista suorittaa reaaliaikaisia tehtäviä. Tällaisella käyttöjärjestelmällä voi olla useita eri käyttäjiä. Suorituskyvyn ja käyttökokemuksen parantamiseksi yleiskäyttöinen käyttöjärjestelmä uhraa taustalla olevien ohjelmien prosessointiaikaa kohdistaakseen prosessointitehoa aktiivisessa käytössä oleviin prosesseihin [3] [11]. Tästä seuraa, ettei reaaliaikaista käyttöjärjestelmää pysty suorittamaan suoraan yleiskäyttöisien käyttöjärjestelmien päällä.

Reaaliaikaisen ja yleisen käyttöjärjestelmän eroja ovat aikavaatimukset, ennakoitavuus ja monipuolisuus. Reaaliaikaiset käyttöjärjestelmät ovat rakennettu tiettyyn tarkoitukseen tarkat aikamääreet vaatimuksina. Reaaliaikainen käyttöjärjestelmä takaa, että tehtävä tullaan suorittamaan tietyssä aikamääreessä ja ennakoitavuus tekee käyttöjärjestelmästä luotettavan. Reaaliaikaiset käyttöjärjestelmät toimivat usein rajallisella laitteistolla varustetuissa sulautetuissa järjestelmissä.

Reaaliaikaisten käyttöjärjestelmien rakenne on riippuvainen sen luokittelusta. Kovan reaaliaikaisuuden järjestelmäksi luokiteltu käyttöjärjestelmä tekee laskentaa niin, että tulokset saadaan oikealla hetkellä [10]. Toiminta keskeytyy, jos tehtävää ei suoriteta määräajassa. Pehmeän reaaliaikaisuuden käyttöjärjestelmissä laitteisto jatkaa toimintaansa määräajan ylittämisestä huolimatta [12]. Järjestelmillä on erilaiset vaatimukset vasteajoissa, kuormituksessa ja aikataulutuksen ohjauksessa. Järjestelmien eroavaisuudet on esitelty taulukossa 2.1.

Taulukko 2.1: Kovan ja pehmeän reaaliaikaisuuksien erot.

Ominaisuus	**Kova reaaliaikaisuus**	**Pehmeä reaaliaikaisuus**
Vasteaika	Tarkasti vaadittu	Toivottu
Suorituskyky rasituksessa	Ennustettava	Heikkenevä
Tahdin hallinta	Ympäristö	Tietokone
Toiminnallinen turvallisuus	Kriittinen	Ei kriittinen
Datan koko	Pieni/keskikokoinen	Suuri
Tiedon eheys	Lyhytaikainen	Pitkäaikainen
Virheiden tunnistus	Autonominen	Käyttäjän avustama

Järjestelmät suunnitellaan arkkitehtuurilliselta rakenteeltaan erilaisiksi, koska niiden käyttötarkoitukset eroavat toisistaan. Pehmeän reaaliaikaisuuden käyttöjärjestelmiä käytetään käyttökohteissa, joissa aikamääreisiin pääseminen ei ole tärkein prioriteetti. Näitä käyttökohteita ovat järjestelmät, joissa aikamääreestä jääminen ei aiheuta vaaratilanteita, kuten käyttöliittymät sulautetuissa järjestelmissä [10] [13]. Kovan reaaliaikaisuuden käyttöjärjestelmiä käytetään turvallisuuskriittisissä järjestelmissä, kuten itseajavissa ajoneuvoissa, lennonohjaus- ja puolustusjärjestelmissä.

Yleiskäyttöisistä käyttöjärjestelmistä tunnetuimpia ovat Linux, Windows ja macOS. Näitä käyttöjärjestelmiä käytetään pääasiassa tietokoneiden käyttöjärjestelminä. Yleiskäyttöisten käyttöjärjestelmien etuna on niiden monipuoliset mahdollisuudet [14]. Ne ovat tehokkaita ja turvallisia. Yleisillä käyttöjärjestelmillä on mahdollista kirjoittaa monia eri ohjelmointikieliä ja hyödyntää kääntäjiä ohjelmien ajamisessa. Jokaiselle käyttöjärjestelmälle löytyy saatavilla olevia avoimia kirjastoja, joiden hyödyntäminen nopeuttaa sovellusten ja ohjelmistojen, kuten sulautettujen järjestelmien tai reaaliaikaisten käyttöjärjestelmien kehitystä.

Sulautetuille järjestelmille suunnattuja käyttöjärjestelmiä on useita, kuten Contiki [15], Contiki-NG, RIOT [16], Zephyr [17], Arm Mbed [18], Apache Mynewt [19], TinyOS [20] ja FreeRTOS [21]. Edellä mainitut käyttöjärjestelmät ovat kaikki avoimen lähdekoodin ohjelmistoja, jotka käyttävät eri lisenssejä. Avoimen lähdekoodin hyödyntämisen mahdollistavia lisenssejä on Apache 2.0 [22], BSD [23], GNU LGPL (Lesser Public License) [8] ja MIT [24]. Lisäksi on vielä kaupallisia reaaliaikaisia käyttöjärjestelmiä, kuten Blackberry QNX [25], Integrity RTOS [26], ja VxWorks [27]. Taulukossa 2.2 esitetään mitkä reaaliaikaisista käyttöjärjestelmistä ovat kovia ja mitkä pehmeitä aikavaatimuksiltaan sekä avoimen lähdekoodin käyttöjärjestelmien lisenssit.

Taulukko 2.2: Reaaliaikaisia käyttöjärjestelmiä, mukaillen [15].

Käyttöjärjestelmä	**Lisenssi**	**Reaaliaikaisuus**
Apache Mynewt	Apache 2.0	Pehmeä
Apache NuttX	Apache 2.0	Kova
Arm Mbed	Apache 2.0	Pehmeä
Blackberry QNX	Suljettu	Kova
ChibiOS	GNU GPL 3.0	Kova
Contiki	BSD	Pehmeä
Contiki-NG	BSD	Pehmeä
FreeRTOS	MIT	Kova
Integrity RTOS	Suljettu	Kova
RIOT	GNU LGPL	Kova
TinyOS	BSD	Pehmeä
VxWorks	Suljettu	Kova
Zephyr	Apache 2.0	Kova

Avoimen lähdekoodin ohjelmisto on lisenssin ehdoilla käytettävissä oleva valmis kehityspaketti [7]. Kehityspaketin etuna on, että se sisältää valmiita työkaluja. Sulautettujen järjestelmien, robottijärjestelmien ja reaaliaikaisten käyttöjärjestelmien kehitys nopeutuu, koska yritykset pystyvät hyödyntämään kehityksessä valmiita kehityspaketteja.

2.2 Avoimen lähdekoodin ohjelmistojen lisensointi

Avoin lähdekoodi voi olla vapaasti tai rajoitetusti käytettävissä, mutta kuka tahansa voi osallistua sen kehittämiseen [28]. Open Source Initiativen (OSI) kriteerien mukaan avoin lähdekoodi on vapaasti tarkasteltavissa, muokattavissa ja levitettävissä [29]. Avoin lähdekoodi on vapaata vain, jos se täyttää lisäksi Free Software Foundationin (FSF) asettamat kriteerit [8]. FSF:n kriteerit määrittelevät neljä vapautta: ohjelmiston ajamista mihin tahansa tarkoitukseen, sen tutkimista ja muokkaamista, kopioiden jakamista sekä muokattujen versioiden levittämistä.

Vapaa avoimen lähdekoodin ohjelmisto on edistänyt uusien teknologioiden leviämistä mahdollistamalla käyttäjille vapaasti saatavilla olevien ohjelmistojen hyödyntämisen sekä kehittäjille avoimen lähdekoodin ohjelmiston sisällyttämisen omiin toteutuksiinsa [9]. Yksilöllisten ja valmiiksi testattujen kirjastojen hyödyntäminen toteutuksissa tarjoaa uudelleen käytettäviä toimintoja ja mahdollistavat nopeamman julkaisutahdin [7]. Ohjelmiston saatavuus ja käyttöehdot määritellään sitä koskevissa lisensseissä. Avoimen lähdekoodin ohjelmistoissa lisenssit ilmaisevat, miten käyttäjät voivat käyttää ohjelmistoa edelleen erottelemalla käyttäjien oikeudet ja velvollisuudet toisistaan. Lisenssien merkitys on lisääntynyt myös akateemisessa yhteisössä viime vuosina. Kun ohjelmistojärjestelmien komponenttien määrä kasvaa, vaikeutuu samalla systeemiin sopivien lisenssien valinta ja niiden yhteensopivuuden tarkistus. Yritysmaailmassa avoimen lähdekoodin ohjelmistoja levitetään usein kalliiseen hintaan, joten näihin ristiriitoihin ei tule suhtautua kevyesti [9]. Lisenssien

moninaisuus hankaloittaa organisaatioiden mahdollisuuksia hallita yhteensopimattomuuksia, joita voi syntyä eri lisenssein lisensoitujen ohjelmistokirjastojen käytöstä. Lisenssien sallivuus vaihtelee laajasti sallivista lisensseistä, kuten MIT-lisenssistä suuresti rajoittavaan GNU GPL (General Public License) -lisenssiin.

Avoimen lähdekoodin lisenssit voidaan luokitella salliviin tai käyttäjänoikeus-lisensseihin. Jälkimmäinen ryhmä voidaan jakaa edelleen heikosti ja vahvasti suojaaviin lisensseihin. Käyttäjänoikeus (copyleft) on GNU:n kehittämä menetelmä, jonka tarkoituksena on estää GNU-ohjelmistoa muuttumasta suljetuksi ohjelmistoksi [8] [9]. Sallivalla lisenssillä on vain vähän vaatimuksia. Sellaisella lisenssillä julkaistua avoimen lähdekoodin materiaalia voidaan levittää osana laajempaa tuotetta lähes minkä tahansa muun lisenssin alaisena, kunhan alkuperäiset tekijät mainitaan. Vahvojen ja heikkojen käyttäjänoikeus-lisenssien välinen ero liittyy alkuperäisen tekijänoikeuden alaisesta teoksesta muokattujen versioiden levittämiseen. Vahvan käyttäjänoikeus-lisenssin alaisen materiaalin johdannaisteokset on levitettävä samalla lisenssillä. Mikäli materiaali on heikon käyttäjänoikeus-lisenssin alaisena, voidaan sen johdannaisteoksia levittää muilla lisensseillä, kunhan heikon käyttäjänoikeus-lisenssin alaista ohjelmistoa ei ole muokattu. Aiemmin esiteltyjen reaaliaikaisten käyttöjärjestelmien lisenssien lisäksi on muitakin vapaan avoimen lähdekoodin lisenssejä, kuten BSL 1.0 (Boost Software License) ja GNU GPL 2.0. Taulukossa 2.3 avoimen lähdekoodin ohjelmiston lisenssejä jaoteltuna kategorioihin.

Taulukko 2.3: Lisenssien kategorisointi

Lisenssi	**Tyyppi**
Apache 2.0	Salliva
BSD	Salliva
BSL 1.0	Salliva
GNU GPL 3.0	Vahva käyttäjänoikeus
GNU LGPL	Heikko käyttäjänoikeus
MIT	Salliva

Täydellä tai osittaisella käyttäjänoikeus-lisenssillä tarkoitetaan sitä osaa ohjelmistosta, joka on asetettava saataville saman käyttäjänoikeus-lisenssin alaisena ohjelmistoa muutettaessa tai levitettäessä [9]. Nämä eroavat heikoista ja vahvoista käyttäjänoikeus-lisensseistä, eikä niitä tule sekoittaa keskenään. Täyden käyttäjänoikeus-lisenssin alainen ohjelmisto käsittää ohjelmiston kaikki osat ja ne tulee levittää saman lisenssin alla, kun taas heikon käyttäjänoikeus-lisenssin alainen ohjelmisto käsittää ohjelmiston, jossa osa muutoksista voidaan levittää toisen lisenssin alaisena.

Ongelmaksi syntyykin se, miten kehittäjät pystyvät sisällyttämään kolmannen osapuolen ohjelmistoja omiin toteutuksiin ilman lisenssirikkomuksia. Yhteensopimattomuudesta esimerkkinä toimii alkuperäisen BSD-lisenssin yhteensopimattomuus kaikkien GNU GPL:n versioiden kanssa [8] [9]. Myös esimerkiksi Apache 2.0 -lisenssi on yhteensopimaton GPL 2.0:n kanssa, koska Apache 2.0:n ehdot asettavat lisenssille rajoituksia, joita GPL 2.0:ssa ei ole [22].

2.3 Robotiikan ohjelmistokehykset ja rajallinen laitteisto

Robotiikan ohjelmistokehykset

Robotiikan ohjelmistokehykset on suunniteltu purkamaan monimutkainen ohjelmisto pienempiin ja paremmin hallittaviin osiin [6]. Avoimen lähdekoodin kehitysalustoista suurimman roolin on saanut robotiikan saralla ROS. Se pyrkii vastaamaan standardoinnin puutteeseen ja helpottamaan sekä nopeuttamaan robottijärjestelmien kehitystä. ROS on avoimeen lähdekoodiin perustuva ohjelmistokehitysalusta, jota käytetään Linuxin päällä. Vaikka ROS 1:nä tunnettu ROS:n ensimmäinen versio yksinkertaistaa robottien kehitystä, sisältää se monia ongelmia. Se sisältää yksittäisen vikakohdan, datapakettien toimittaminen Wi-Fi-verkkojen yli takkuilee eikä se sisällä lainkaan sisäänrakennettuja turvallisuusmekanismeja.

ROS 2 pyrkii vastaamaan edeltäjänsä ROS 1:n puutteisiin [6]. ROS 1:n ongelmien korjaaminen oli rakenteellisista syistä johtuen vaikeaa. Esimerkiksi yksittäisen vikakohdan korjaaminen olisi vaatinut jokaisen asiakaskirjaston päivittämisen yksilöllisillä ratkaisuilla. Myös turvallisuuden korjaaminen oli mahdollista, mutta ratkaisu osoittautui vaikeaksi ylläpitää ja olisi vaatinut jatkokehitystä. ROS 2 kehitettiin täysin eri arkkitehtuurilla kuin ROS 1. Sitä kehitettäessä ei vastattu pelkästään edeltäjän ongelmiin, vaan uuden arkkitehtuurin lisäksi ROS 2:sta tehtiin käytännöllisempi. Yksi kehittäjiä helpottava muutos on ROS:n toimiminen myös Windowsilla ja macOS:llä. ROS 2:n perustana toimii avoin viestintästandardi Data Distribution Service (DDS), jota käytetään kriittisessä infrastruktuurissa. DDS mahdollistaa ROS 2:n turvallisen hyödyntämisen sulautetuissa ja reaaliaikaisissa järjestelmissä sekä monirobottijärjestelmien keskinäisen kommunikoinnin.

Mikrokrontrollerien tehokkaaseen käyttöön kehitetty micro-ROS on ROS:iä vastaava ohjelmistopino [30]. ROS 2:sta poiketen micro-ROS korvaa DDS-

viestintästandardin XRCE-DDS-protokollalla. Protokolla yhdistää mikrokontrollerit micro-ROS -solmuina ROS 2:n verkkoon agentti-asiakas-mallilla. Tällä mahdollistetaan pääsy mikrokontrollereihin tunnetuilla ROS 2 -työkaluilla. micro-ROS käyttää samaa Apache License 2.0 -lisenssiä kuin ROS 2. micro-ROS:n rakenne on kerrostettu ja modulaarinen.

micro-ROS tukee kolmea avoimen lähdekoodin reaaliaikaista käyttöjärjestelmää, joita ovat FreeRTOS, Zephyr ja NuttX. Se on siirrettävissä mille tahansa reaaliaikaiselle käyttöjärjestelmälle, joka tukee POSIX-rajapintaa. Siirtämistä saattaa kuitenkin rajoittaa lisenssien yhteensopimattomuus käytettävän reaaliaikaisen käyttöjärjestelmän kanssa [9].

Rajallinen laitteisto ja sulautetut järjestelmät

Rajallisella laitteistolla tarkoitetaan sulautettuja järjestelmiä, jotka on varustettu huomattavasti rajallisemmalla laskentateholla, muistikapasiteetilla ja energiavaroilla kuin yleiskäyttöiset tietokoneet. Rajallinen laitteisto voidaan yleisesti luokitella matalan ja korkean tason laitteisiin niiden suorituskyvyn perusteella. Korkean tason laite kykenee suorittamaan yleiskäyttöistä käyttöjärjestelmää, johon matalan tason laite ei kykene sen rajallisten resurssien vuoksi. Tarkempi luokittelu muistin määrän perusteella on määritelty IETF:n RFC 7228 -dokumentissa [31], jonka perusteella luokan 0 laitteet sisältävät alle 10 kilotavua RAM-muistia, luokan 1 laitteet noin 10 kilotavua ja luokan 2 noin 50 kilotavua.

Rajallisen laitteiston mikrokontrollerien kellotaajuudet vaihtelevat muutamista megahertseistä satoihin megahertseihin. Lennonohjausjärjestelmissä tyypillinen kellotaajuus on 180 MHz ja RAM-muisti 196 kilotavua [1]. Nämä mikrokontrollerit ajavat ohjelmakoodia usein suoraan flash-muistista koska erillinen keskusmuisti on rajallinen tai puuttuu kokonaan. Muistinhallintayksikön puuttuminen tarkoittaa, ettei käyttöjärjestelmä voi käyttää perinteistä virtuaalimuistia prosessien eristämi-

seen, mikä asettaa lisävaatimuksia reaaliaikaiselle käyttöjärjestelmälle ja sen muistin hallinnalle.

Lennonohjausjärjestelmä (Flight Control Unit) on miehittämättömän ilmaaluksen eli yhdenlaisen droonin sulautettu ohjausjärjestelmä [1]. Lennonohjausjärjestelmä vastaa laitteen vakauttamisesta, navigoinnin ohjaamisesta ja ohjauskomentojen toteuttamisesta. Lennonohjausjärjestelmä koostuu neljästä pääkomponentista: mikrokontrollerista, sensoreista, toimilaitteista ja tietoliikennerajapinnoista. Mikrokontrolleri on järjestelmän laskentayksikkö, joka käsittelee sensoridataa ja ohjaa moottoreita reaaliaikaisesti. Sensoridatan käsittelyn on tapahduttava reaaliaikaisesti, sillä viiveet ohjaussilmukassa voivat vaarantaa lennokin vakauden ja käyttöturvallisuuden. Lennonohjausjärjestelmän ohjelmistoarkkitehtuuri käsittelee sensoridataa, joiden perusteella se ohjaa droonia toistuvissa sykleissä. Syklit on toteutettava deterministisesti ja tiukkojen aikarajojen puitteissa, mikä tekee lennonohjausjärjestelmästä kovan reaaliaikaisuuden sovelluksen. Lennonohjausjärjestelmän laitteistoalusta on tyypillisesti rajallisen laitteiston mikrokontrolleri. Avoimen lähdekoodin lennonohjausohjelmistot, kuten ArduPilot ja PX4, ovat mahdollistaneet tutkimus- ja kehitystyön nopean kehittymisen muokattavalla lähdekoodilla ja vaihdettavilla laitteistoalustoilla [4] [32] [33]. Reaaliaikaisen käyttöjärjestelmän valinta vaikuttaa suoraan rajallisella laitteistolla varustetun järjestelmän toimintaan.

3 Reaaliaikaiset käyttöjärjestelmät

Reaaliaikaisia käyttöjärjestelmiä määritellään aikavaatimuksiin vastaamisen perusteella kovan ja pehmeän reaaliaikaisuuden käyttöjärjestelmiin. Turvallisuutta ja tehokkuutta vaativissa tehtävissä käytössä on pääsääntöisesti kaupalliset käyttöjärjestelmät, mutta rajallisen laitteiston sovelluksissa on edetty pitkälti avoimeen lähdekoodiin pohjautuvilla projekteilla. Nämä projektit ovat lähteneet hakemaan ratkaisuja rajallisen laitteiston tuomiin haasteisiin eri tavoin.

3.1 Vertailu rajallisella laitteistolla

Avoimen lähdekoodin pohjalta kehitettyjen reaaliaikaisten käyttöjärjestelmien lisäksi on olemassa myös kaupallisia versioita. Reaaliaikaisten käyttöjärjestelmien kaupalliset versiot tarjoavat usein vähemmän rajapintoja, joiden kanssa työskennellä ja muuttavat myös harvemmin arkkitehtuuriaan tutkimustyön perusteella [34]. Yksityisesti kehitettäviä kaupallisia reaaliaikaisia käyttöjärjestelmiä suositaan tarkkuutta ja turvallisuutta vaativissa järjestelmissä [35]. Tällaisia sovelluksia voivat olla avaruudessa käytettävät sulautetut järjestelmät, kuten satelliitit tai itseajavien autojen reaaliaikaiset käyttöjärjestelmät.

Reaaliaikaisten käyttöjärjestelmien kehitykseen käytetään avoimen lähdekoodin reaaliaikaisia käyttöjärjestelmiä ja työkaluja. Osa autovalmistajista ja autoille ohjelmistoja kehittävistä yrityksistä kehittävät itseajavien autojen vaatimia käyttöjärjestelmiä Linuxilla toimivan ROS:n avulla [36]. BMW on valinnut ROS:n sen avoimen

lähdekoodin ja olemassa olevien kirjastojen ansiosta. Näiden ohjelmistojen käyttö tasaa kilpailua ja edistää yritysten kehitystä säästämällä aikaa sekä rahaa, kun käytettävissä on valmiita työkaluja. Aiemmin taulukossa 2.2 esiteltyjen reaaliaikaisten järjestelmien käyttökohteita on listattuna seuraavaan taulukkoon 3.1.

Taulukko 3.1: Reaaliaikaisia käyttöjärjestelmiä

Käyttöjärjestelmä	**Käyttökohteita**
Apache Mynewt	Langattomat ja resurssirajoitteiset IoT-laitteet
Apache NuttX	Korkeintaan 64 bittiset mikrokontrollerit
Arm Mbed	Arm Cortex-M -mikrokontrollereille
Blackberry QNX	Turvakriittiset sulautetut järjestelmät
ChibiOS	Resurssirajoitteiset sulautetut järjestelmät
Contiki	Pienet ja vähän virtaa kuluttavat IoT-laitteet
Contiki-NG	Luotettavan tiedonsiirron IoT-laitteet
FreeRTOS	Mikrokontrollerit sulautetuissa järjestelmissä
Integrity RTOS	Turvakriittiset ja verkottuneet sovellukset
RIOT	Matalan tason IoT-laitteet
TinyOS	Vähävirtaiset langattomat laitteet
VxWorks	Turvakriittiset sulautetut järjestelmät
Zephyr	Resurssirajoitteiset ja turvalliset laitteet

Blackberry QNX on kaupallinen reaaliaikainen käyttöjärjestelmä [36]. QNX:n yleisimpiä sovelluskohteita ovat autot, puhelimet, IoT-laitteet ja lääkinnälliset laitteet. Käyttöjärjestelmällä on autoteollisuuden korkeimman tason turvallisuussertifikaatti 26262 ASIL-D, SIL 3 -sertifiointi teollisen automaation järjestelmille ja lääkinnällisten laitteiden luokan kolme IEC 62304 -sertifiointi. Valmiit sertifioinnit QNX:n

kaltaisissa reaaliaikaisissa käyttöjärjestelmissä lisäävät kilpailua avoimien ja suljettujen ohjelmistojen välillä.

Reaaliaikaista käyttöjärjestelmää hyödyntävän laitteen käyttötarkoitus vaikuttaa pitkälti käyttöjärjestelmän valintaan. Laitteeseen, joka ei ole turvakriittinen riittää heikompi reaaliaikainen käyttöjärjestelmä. Kevyempi käyttöjärjestelmä sopii laitteisiin, joissa on rajallinen laskentateho tai muisti. Resurssien asettamat rajoitteet vaikuttavat siihen, millainen reaaliaikainen käyttöjärjestelmä laitteeseen voidaan valita. Esimerkiksi akuilla toimivien droonien käyttöaikaan vaikuttaa järjestelmän energiatehokkuus.

ArduPilot [32] ja PX4 autopilot [33] ovat lennonohjausohjelmistoja. Lennonohjausohjelmistoja käytetään drooneissa, näissä laitteissa rajallinen laitteisto on yksi haasteista reaaliaikaista käyttöjärjestelmää implementoidessa [4]. Näistä ArduPilot käyttää Apache NuttX:ää [37] reaaliaikaisena käyttöjärjestelmänä ja PX4 ChibiOS:ää [38]. PX4 autopilot käytti ArduPilotin kanssa aiemmin samaa reaaliaikaista käyttöjärjestelmää, mutta vaihtoi vuonna 2018 käyttämään kevyempää ChibiOS:ää. ChibiOS:n ytimen koko on 5 kilotavua, kun Apache NuttX:n ydin on kooltaan 20 kilotavua, mikä tekee ChibiOS:sta lähtökohtaisesti suotuisamman rajalliselle laitteistolle. Näiden käyttöjärjestelmien vertailussa ChibiOS todettiin paremmaksi reaaliaikaiseksi käyttöjärjestelmäksi sen keveyden ja testeissä suoriutumisen ansiosta. Se voitti Apache NuttX:n lähes jokaisessa testissä, vaikka molemmat suoriutuivat kaikista testeistä. Apache NuttX ja ChibiOS eroavat rakenteeltaan kuitenkin olennaisesti. ChibiOS:n ollessa kevyempi se käyttää vähemmän resursseja ja siten sillä on pienempi vasteaika. ChibiOS suojaa säikeiden samanaikaisen ajamisen semaforeilla, kun taas Apache NuttX mutekseilla. ChibiOS:n aikatauluttaja asettaa säikeet tärkeysjärjestykseen yksinkertaisemmalla logiikalla, kun Apache NuttX käyttää monimutkaisempaa aikatauluttajaa.

Zephyr on valmis reaaliaikaisen käyttöjärjestelmän toteutus, joka sisältää turvallisuuskonfiguroitavan ja modulaarisen kernelin, monen alustan sopivuuden, avoimen lähdekoodin ja yhdistettävyyden ominaisuudet. Se on kompaktin ytimen ansiosta kevyenä käyttöjärjestelmänä suunniteltu rajallisten resurssien IoT- ja sulautetetuille laitteille [5]. Zephyr on laajasti hyödynnettävissä oleva reaaliaikainen käyttöjärjestelmä, joka tukee standardeja, kuten POSIX API. Sillä on Linux Foundationin isännöimä (host) Apache 2.0 lisenssi. Zephyrin perusideana on antaa kehittäjien keskittyä loppukäyttäjien käyttöliittymiin alhaisen tason käyttöliittymien uudelleenimplementoimisen sijaan [17]. Se koostuu modulaarisista komponenteista, jotka on mahdollista koota tarpeen mukaan sisältämään vain tarvittavat toiminnallisuudet. Zephyrin modulaariset kerrokset esitellään kuvassa 3.1.

Kuva 3.1: Zephyrin modulaarinen arkkitehtuuri [17].

Zephyrin modulaarisia osia ovat kolmannen osapuolen kirjastot (Third Party Libraries), sovelluspalvelut (Application Services), väliohjelmisto ja verkkotoiminnot (Middleware Networking), ydin (Kernel), sekä laitteiston abstraktiokerros HAL (Hardware Abstraction Layer). Kolmannen osapuolen kirjastot laajentavat käyttöjärjestelmän ominaisuuksia ilman, että kehittäjän tarvitsee itse toteuttaa niitä.

Sovelluspalvelut sisältävät esimerkiksi valmiita ohjelmistokomponentteja tai järjestelmäpalveluja kehityksen tueksi. Väliohjelmisto ja verkkotoiminnot sisältävät väliohjelmistoja sekä esimerkiksi viestintäprotokollia tai verkko-ominaisuuksia. Ydin hallitsee resursseja ja sisältää ajastimen, joka huolehtii tehtävien suorittamisjärjestyksestä. Zephyr on kovan reaaliaikaisuuden käyttöjärjestelmä, joten ydin huolehtii lisäksi keskeytyksistä. Zephyr-käyttöjärjestelmä kokoaa kerroksista modulaarisen kokonaisuuden.

3.2 Arkkitehtuurilliset ratkaisut rajalliselle laitteistolle

Droonien tarpeen kasvaessa ammattikäytössä kasvaa laitteiston, kuten sensorien ja kameroiden, tarve lennokeissa [1]. Samalla laitteen turvallisuus muuttuu yhä oleellisemmaksi järjestelmän toiminnan keskeytyessä vikatilassa. Lennokkiin lisättäessä monimutkaisempia sensoreita ja kantokykyvaatimusten noustessa, monimutkaistuu myös reaaliaikaisten käyttöjärjestelmien ominaisuudet. Näiden ominaisuuksien ohjaaminen on haastavaa lennonohjaimille laitteisto- ja ohjelmistotasoilla. Ammattikäyttöön ja yksityiseen käyttöön tarkoitettujen droonien välillä on suuri kuilu. Yksityiskäyttäjille tarkoitettujen tuotteiden hinnat ovat laskeneet johtuen pitkälti avoimen lähdekoodin ja laitteiston kehityksestä. Ammattikäytössä käytetään kuitenkin yhä kalliita välineitä, mikä vaikeuttaa laitteistoon käsiksi pääsyä satunnaisille käyttäjille. Uuden laitteiston kehittämisen ollessa kallista, tukeudutaan usein kaupallisiin lennonohjaimiin, kuten Pixhawkiin tai Cuaviin. Linuxiin pohjautuvat reaaliaikaiset käyttöjärjestelmät ovat usein liian monimutkaisia droonien kehitykseen. Nämä tekijät ovat johtaneet kehityksen keskittymisen jo olemassa oleville alustoille.

Reaaliaikaisen käyttöjärjestelmän ollessa tärkeä osa droonia, on niiden vaatimusten kasvuun ehdotettu uusi reaaliaikainen käyttöjärjestelmä, joka yhdistää ai-

katauluttajista saapumisjärjestykessä (First Come First Serve, FCFS) ja aikaisimman määräajan (Earliest Deadline First, EDF) järjestyksessä tehtäviä käsittelevät logiikat hybridiaikatauluttajaksi [1]. Näiden aikatauluttajien yhdistäminen varmistaa, että kriittiset tehtävät eivät keskeydy, sillä FCFS varmistaa pienellä viiveellä kriittisten tehtävien toteuttamisen ja EDF hallitsee jäljelle jääviä tehtäviä tärkeysjärjestyksessä.

Hybridiaikatauluttajaa ehdottava työ esittelee kolme kontribuutiota [1], joita ovat markkinoiden kanssa kilpailukykyinen lennonohjainlaitteisto, uusi kovan reaaliaikaisuuden käyttöjärjestelmä rajallisen laitteiston sulautetuille järjestelmille ja droonien ammattikäyttöön suunniteltu laitteisto-ohjelmisto-kokonaisuus. Ohjelmisto on rakennettu seuraavien viiden periaatteen varaan: kaikki tehtävät suoritetaan yhtäjaksoisesti yhdellä CPU:lla, prosessorien välillä ei ole datariippuvuuksia, prosessien suoritusaika on vakio, kaikki määräajat osuvat jakson loppuun ja suoritukseen valitaan aina korkeimman prioriteetin prosessi.

Nykyiset sulautetut järjestelmät ja reaaliaikasovellukset vaativat usein verkkoyhteyksiä ja graafista käyttöliittymää. Vaatimusten muuttuminen on ajanut kehittäjiä hakemaan keinoja yhdistää reaaliaikainen ja yleiskäyttöinen käyttöjärjestelmä, koska reaaliaikasovelluksissa tarvitaan usein vain muutamia reaaliaikaisen käyttöjärjestelmän ominaisuuksia. Käyttöjärjestelmien yhdistämistä on koitettu useilla menetelmillä, kuten muuttamalla yleiskäyttöinen käyttöjärjestelmä reaaliaikaiseksi käyttöjärjestelmäksi, mutta tämä menetelmä vaatii kernelin muuttamista reaaliaikaisen suorityskyvyn saavuttamiseksi [39]. Käyttöjärjestelmiä on myös yritetty isännöidä samalla koneella, mutta menetelmä vaatii reaaliaikasovelluksen jakamisen kahteen osaan, eikä järjestelmät voi kommunikoida keskenään perinteisten POSIX-rajapintojen avulla. Vaativiin reaaliaikasovelluksiin ehdotettu yhdistetty reaaliaikainen käyttöjärjestelmä (Compounded RTOS, cRTOS) jakaa fyysisen koneen hyper-

visorilla ja ajaa reaaliaikaista käyttöjärjestelmää rinnakkain yleiskäyttöisen käyttöjärjestelmän kanssa. Ratkaisu ei vaadi kernelin muokkaamista.

Yhdistetty reaaliaikakäyttöjärjestelmä koostuu kahdesta virtuaalikoneesta, joissa toisessa toimii reaaliaikainen ja toisessa yleiskäyttöinen käyttöjärjestelmä. Virtuaalikoneita ajetaan rinnakkain erillisillä ytimillä ilman, että ne estävät toisiaan [39]. Vaatimuksena laitteiston prosessorilla tulee olla vähintään kaksi ydintä. Käyttöjärjestelmät on yhdistetty osioivalla hypervisorilla (Partitioning Hypervisor), mikä mahdollistaa reaaliaikaisen suorituskyvyn eristämällä ja osoittamalla tietokoneen resursseja, ytimiä ja muistia virtuaalikoneille.

Sulautetut järjestelmät hyödyntävät mikrokontrollereita ohjaustoiminnoissaan ja niitä voi olla useita integroituna sulautettuun järjestelmään, mikä vaikeuttaa niiden tehokasta käyttöä. micro-ROS mahdollistaa mikrokontrollerien käytön tehokkaasti ROS 2:n tavoin [30]. Mikrokontrollerien ohjaamisen mahdollistaa micro-ROS:in sisältämät kaikki tärkeimmät ROS-toiminnot, kuten solmut (Nodes), julkaisijat (Publishers), tilaukset (Subscribers), parametrit (Parameters) ja elinkaarihallinnan (Lifecycle). Tämä mahdollistaa sulautettujen järjestelmien kehittämisessä, että kaikkea ohjelmistoa voi käyttää samoilla ROS-työkaluilla ja -rajapinnoilla, riippumatta laskentalaitteistosta tai käyttöjärjestelmästä. micro-ROS tukee tärkeimpiä avoimen lähdekoodin reaaliaikaisia käyttöjärjestelmiä, kuten FreeRTOS:ää, Zephyriä ja NuttX:ää.

4 Pohdinta

Rajallisen laitteiston tuomiin haasteisiin on pyritty vastaamaan vaihtelevin menetelmin. Näihin haasteisiin on vastattu avoimen lähdekoodin ohjelmistoilla niiden mahdollistaman nopean kehityksen ansiosta [6] [7]. Avoimen lähdekoodin ohjelmistojen käyttö on kuitenkin tarkasti lisensoitu tarkoituksena estää niitä päätymästä suljetuiksi ohjelmistoiksi [8]. Tässä kirjallisuuskatsauksessa esiteltiin reaaliaikaisia käyttöjärjestelmiä ja niiden eroja yleiskäyttöisiin käyttöjärjestelmiin. Zephyrin ratkaisuihin perehdyttiin tarkemmin. Tutkielmassa perehdyttiin lennonohjausjärjestelmistä ArduPilotin ja PX4 Autopilotin käyttämiin reaaliaikaisiin käyttöjärjestelmiin sekä käsiteltiin lisensoinnin tuomia haasteita yhteensopivuudelle.

Tässä tutkielmassa käsitellyn kirjallisuuden perusteella voi todeta, että avoimen lähdekoodin reaaliaikaiset käyttöjärjestelmät ovat nopeuttaneet ja muuttaneet sulautettujen järjestelmien kehitystä [6] [7]. Kuten tutkielmassa huomasimme, niiden pohjalta järjestelmien kehittäminen ei ole mutkatonta. Rajallisen laitteiston tuomat haasteet eivät rajoitu yksinään rajallisiin resursseihin, vaan ne kulminoituvat lisäksi reaaliaikaisuusvaatimusten ja lisenssien yhteen sovittamisen tuomiin haasteisiin [4] [9].

Reaaliaikaisia ja yleiskäyttöisiä käyttöjärjestelmiä erottavista deterministisyydestä ja niille asetetuista aikavaatimuksista oltiin kirjallisuudessa johdonmukaisesti yhtä mieltä [2] [3] [10]. Katsauksessa esiin tuotu ehdotettu yhdistetty reaaliaikainen

käyttöjärjestelmä kuitenkin hämärtää järjestelmien välistä rajaa pyrkimällä mahdollistamaan niiden samanaikaisen ajamisen yhdellä moniytimisellä prosessorilla [39].

Tutkielmassa käsitelty kirjallisuus määritteli kovan ja pehmeän reaaliaikaisuuden välisen rajaa häilyvämmin [2] [10]. Tämä johtuu esimerkiksi Zephyrin monipuolisuudesta ja sen taipumisesta moneen käyttötarkoitukseen, mikä hankaloittaa Zephyrin luokittelua vain yhteen kategoriaan [17]. Modulaarinen ja räätälöitäväksi suunniteltu reaaliaikainen käyttöjärjestelmä jättää tulkinnan varaa aina tutkimuksen kohteen mukaan teoreettiselle määrittelylle. Tutkielman perusteella voidaan todeta ettei käyttöjärjestelmää valittaessa kovan ja pehmeän reaaliaikaisuuden välinen jako ole niin yksiselitteinen kuin se on taulukossa 2.2 tuotu esille.

Käsitellyssä kirjallisuudessa esitettiin useita ratkaisuja rajallisen laitteiston tuomille haasteille. Tutkielma esittelee kirjallisuudessa vertaillun ChibiOS:n ja Apache NuttX:n väliset erot ja niiden keskenään poikkeavat ratkaisut joista ChibiOS jättää pienikokoisella 5 kilotavun ytimellä enemmän resursseja prosessorin ja muistin käyttöön [4]. Toisena modulaariseksi suunniteltu Zephyr mahdollistaa järjestelmän kokoamisen kevyeksi valitsemalla vain käyttötarpeeseen vaaditut komponentit [17]. Kolmanneksi ratkaisuksi tutkielma esittelee yhdistetyn reaaliaikaisen käyttöjärjestelmän lähestymistavan yhdistää reaaliaikainen ja yleiskäyttöinen käyttöjärjestelmä hypervisorilla [39]. micro-ROS puolestaan tuo mikrokontrollerit tehokkaasti käytettäviksi tunnettujen ROS 2 -rajapintojen taakse [30]. Tutkielmassa esille tuotujen ratkaisujen monimuotoisuudesta voidaan päätellä, että rajallisen laitteiston tuomia haasteita ei pystytä ratkaisemaan vain yhdellä tavalla. Valittava ratkaisu tulee suunnitella joka kerta sovelluksen vaatimusten perusteella. Pienikokoinen ydin mahdollistaa rajallisenkin prosessorin käytön, kun taas vaativamman ja monipuolisemman järjestelmän toteuttamiseen sopii modulaarinen ratkaisu [4] [17].

Tutkielmassa käsiteltiin yksinkertaistetusti lisensoinnin rajatapuksia niiden yhteensopimattomuuden tuomien haasteiden esittelemiseksi. Todellisuudessa lisensoin-

ti on laajempi käsite jonka sivuuttamisella voi olla oikeudellisia seuraamuksia [9]. Lisenssit varmistavat kuitenkin avoimen lähdekoodin ohjelmistojen säilymisen avoimina sekä niiden nopean kehityksen yhteensopivien lisenssien alla. Tämän perusteella voidaan katsoa lisenssivalinnan olevan tärkeä tehtävä heti projektin alussa, sillä se voi vaikuttaa koko projektin kehitykseen, mikäli tarpeelliset kolmannen osapuolen kirjastot ei ole käytettävissä.

Tutkielman rajoitteiksi voidaan katsoa sen keskittyminen droonikontekstiin, eikä tulokset ole suoraan yleistettävissä muihin rajallisen laitteiston sovelluksiin, kuten lääkinnällisiin laitteisiin. Tämä rajaa tutkielman kirjallisuutta tarkastelemaan avoimen lähdekoodin reaaliaikaisia käyttöjärjestelmiä tietystä näkökulmasta valikoitujen rajallisten laitteistojen sovelluksissa. Kovan ja pehmeän reaaliaikaisuuden luokittelu katsauksessa perustuu lähteisiin, jotka eivät ole täysin yhteneväisiä tulkinnoissaan, mikä heikentää näiden erojen esiin tuotua yksiselitteisyyttä [2] [10] [13]. Lisenssien osalta tarkastelu rajoittui yhteensopivuuden haasteisiin, eikä lisensoinnin muita muotoja käsitellä tutkielmassa.

Tutkielman rajoitteet huomioiden voidaan kuitenkin päätellä, että avoimen lähdekoodin ohjelmistot, kuten reaaliaikaiset käyttöjärjestelmät tai sovelluskehykset ovat mahdollistaneet rajallisen laitteiston kehityksen nopeutumisen [6] [7]. Silti niiden hyödyntämiseen vaaditaan huolellisuutta ja perehtyneisyyttä käyttöjärjestelmän arkkitehtuurista, resurssien hallinnasta ja lisensoinnista. Tutkielman painottuessa droonikontekstiin ja lennonohjausjärjestelmiin, jää muut rajallisen laitteiston sovellusalueet, kuten lääkinnälliset tai autonomiset laitteet huomiotta.

5 Yhteenveto

Tutkielmassa tarkasteltiin avoimen lähdekoodin reaaliaikaisia käyttöjärjestelmiä resurssirajoitteisille sulautetuille järjestelmille. Tutkielma toteutettiin kirjallisuuskatsauksena, jossa käsiteltiin reaaliaikaisten käyttöjärjestelmien perusperiaatteita ja niiden eroja yleiskäyttöisiin käyttöjärjestelmiin, rajallisen laitteiston asettamia rajoitteita, avoimen lähdekoodin lisenssejä sekä arkkitehtuurisia ratkaisuja rajallisen laitteiston tuomiin haasteisiin.

Tutkielmassa vastattiin ensimmäiseen tutkimuskysymykseen **"Mitä reaaliaikaiset käyttöjärjestelmät ovat ja kuinka ne eroavat yleiskäyttöisistä käyttöjärjestelmistä?"** määrittelemällä käyttöjärjestelmät ja käsittelemällä niiden välisiä eroja. Käyttöjärjestelmiä erottaa deterministisyys. Reaaliaikainen käyttöjärjestelmä toimii deterministisesti eli suorittamalla tehtävät tärkeysjärjestyksessä ennakoivasti ja toistettavasti. Yleiskäyttöinen käyttöjärjestelmä voi uhrata prosessointiaikaa muille tehtäville parhaan käyttökokemuksen mahdollistamiseksi [3].

Tutkielmassa vastattiin toiseen tutkimuskysymykseen **"Millaisilla ratkaisuilla avoimen lähdekoodin reaaliaikaisissa käyttöjärjestelmissä on vastattu rajallisen laitteiston tuomiin haasteisiin?"** esittelemällä neljä eri käyttötarpeisiin sopivaa ratkaisua. ChibiOS lähestyy ongelmaa pienikokoisen ytimen kautta jättämällä ohjelmistolle mahdollisimman paljon resursseja käyttöön. Zephyr on ratkaissut ongelman modulaarisella arkkitehtuurilla, joka mahdollistaa käyttötarkoitukselle turhien komponenttien poistamisen. Yhdistetty reaaliaikainen käyttöjärjestelmä on

reaaliaikaisen ja yleiskäyttöisen käyttöjärjestelmän yhdistämiseksi ehdotettu ratkaisu. Yhdistetty reaaliaikainen käyttöjärjestelmä suorittaa rinnakkain virtualisoituja käyttöjärjestelmiä yhdellä moniytimisellä prosessorilla jossa käyttöjärjestelmät ovat eristettyinä omille ytimilleen reaaliaikaisuuden mahdollistamiseksi. micro-ROS yhdistää ROS 2:n mikrokontrollereihin tunnetuilla ROS 2 -työkaluilla.

Tutkielmassa vastattiin kolmanteen tutkimuskysymykseen **"Miten avoimen lähdekoodin ohjelmistojen kehittäminen ja levittäminen mahdollistetaan lisensseillä?"** käsittelemällä minkälaisiin luokkiin lisenssit jakautuvat. Luokkia on vahvat ja heikot käyttäjänoikeus-lisenssit, kuten GNU GPL ja GNU LGPL sekä sallivat lisenssit, kuten Apache 2.0 ja BSD. Käyttäjänoikeus-lisenssit vaativat ohjelmistojen levittämistä samalla tai yhteensopivalla lisenssillä, kun taas sallivat lisenssit mahdollistavat ohjelmiston levittämisen lähes minkä tahansa lisenssin alaisena. Lisensseillä varmistetaan ettei avoimen lähdekoodin ohjelmistot päädy suljetuiksi. Tutkielman perusteella lisenssien yhteensopimattomuus tuo haasteita lisenssin valintaan ja ohjelmistojen kehitykseen.

Jatkotutkimuksena tulisi tarkastella rajallisen laitteiston tuomia haasteita esimerkiksi lääkinnällisissä, autonomisissa tai turvakriittisissä laitteissa. Rajallista laitteistoa hyödynnetään lähes kaikilla sulautettujen järjestelmin sovellusalueilla. Koska useassa artikkelissa korostui kaupallisten ja suljettujen reaaliaikaisten käyttöjärjestelmien eduksi valmiit sertifikaatit ja sertifioitavuus, toisena jatkotutkimusaiheena tulisi tarkastella avoimen lähdekoodin reaaliaikaisten käyttöjärjestelmien kehittämisestä syntyvien kokonaiskustannusten vertailu kaupallisiin reaaliaikaisiin käyttöjärjestelmiin. Osana jatkotutkimusta voisi lisäksi tutkia tarkkaan säädellyille sovellusalueille avoimeen lähdekoodiin perustuvien ratkaisujen hyväksyttämistä, sekä lisensoinnin vaikutusta sertifiointiprosessiin osana koko projektin kehityskaarta.

Lähdeluettelo

[1] R. Rico, J. Rico-Azagra ja M. Gil-Martínez, "Hardware and RTOS Design of a Flight Controller for Professional Applications", *IEEE Access*, vol. 10, s. 134 870–134 883, 2022. DOI: 10.1109/ACCESS.2022.3232749.

[2] V. Ivashko, O. Krulikovskyi, S. Haliuk ja A. Samila, "Review of operating systems used in unmanned aerial vehicles", *Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska*, vol. 15, nro 1, maaliskuu 2025. DOI: 10.35784/iapgos.6786.

[3] J. Chen, H. Zhang, R. He, C. Du, J. Cui ja X. Sun, "Design and implementation of a real-time simulation platform for embedded applications on general-purpose operating systems", *SIMULATION*, vol. 99, nro 12, 2023. DOI: 10.1177/00375497231189285.

[4] M. Zhang, M. Timmerman, L. Perneel ja T. Goedemé, "Which Is the Best Real-Time Operating System for Drones? Evaluation of the Real-Time Characteristics of NuttX and ChibiOS", teoksessa *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2021, s. 582–590. DOI: 10.1109/ICUAS51884.2021.9476878.

[5] J. Wang, X. Zhao, G. Qin et al., "Research and Implementation of Zephyr RTOS Porting on RISC-V Platforms", teoksessa *2025 7th International Conference on Frontier Technologies of Information and Computer (ICFTIC)*, 2025, s. 1141–1146. DOI: 10.1109/ICFTIC68075.2025.11324933.

[6] S. Macenski, T. Foote, B. Gerkey, C. Lalancette ja W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild", *Science Robotics*, vol. 7, nro 66, 2022. DOI: 10.1126/scirobotics.abm6074.

[7] J. Haviland ja P. Corke, "Robotics Software: Past, Present, and Future", *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 7, 2024.

[8] R. Stallman. "The GNU Project". GNU Project - Free Software Foundation, updated 2025-01-20, viitattu 2026-03-15. (2025), url: https://www.gnu.org/gnu/thegnuproject.

[9] G. M. Kapitsaki, N. D. Tselikas ja I. E. Foukarakis, "An insight into license tools for open source software systems", *Journal of Systems and Software*, vol. 102, s. 72–87, 2015.

[10] H. Kopetz ja W. Steiner, *Real-time systems: design principles for distributed embedded applications*. Springer Nature, 2022.

[11] S. Canbaz ja G. Erdemir, "Performance analysis of real-time and general-purpose operating systems for path planning of the multi-robot systems", *International Journal of Electrical and Computer Engineering*, 2022.

[12] I. Corporation, *What is a Real-Time System?*, Viitattu: 12.6.2025, 2024. url: https://www.intel.com/content/www/us/en/learn/what-is-a-real-time-system.html.

[13] K. Pothuganti, A. Haile ja S. Pothuganti, "A comparative study of real time operating systems for embedded systems", *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 4, nro 6, 2016.

[14] S. Yang, S. Li, B. Hao, Q. Duan, W. Chen ja F. Zhou, "Real-Time Vehicle Operating System Analysis, Construction and Testing", teoksessa *2024 IEEE 22nd International Conference on Industrial Informatics (INDIN)*, 2024. DOI: 10.1109/INDIN58382.2024.10774284.

[15] G. Oikonomou, S. Duquennoy, A. Elsts, J. Eriksson, Y. Tanaka ja N. Tsiftes, ”The Contiki-NG open source operating system for next generation IoT devices”, *SoftwareX*, vol. 18, s. 101 089, 2022, ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2022.101089. url: https://www.sciencedirect.com/science/article/pii/S2352711022000620.

[16] RIOT Community. ”RIOT - The friendly Operating System for the Internet of Things”. Viitattu 15.3.2026. (2026), url: https://www.riot-os.org/.

[17] A. Eliasz, *Zephyr RTOS Embedded C Programming: Using Embedded RTOS POSIX API*. Springer, 2024.

[18] Arm Limited. ”Mbed”. Viitattu 15.3.2026. (2026), url: https://os.mbed.com/.

[19] Apache Software Foundation. ”Apache Mynewt”. Viitattu 15.3.2026. (2025), url: https://mynewt.apache.org/.

[20] OSRTOS. ”TinyOS - Open Source RTOS”. Viitattu 15.3.2026. (2025), url: https://osrtos.com/rtos/tinyos/.

[21] AWS Open Source. ”FreeRTOS”. Viitattu 15.3.2026. (2024), url: https://www.freertos.org/.

[22] Apache Software Foundation. ”Apache License, Version 2.0”. Viitattu 15.3.2026. (2004), url: https://www.apache.org/licenses/LICENSE-2.0.html.

[23] Open Source Initiative. ”The 3-Clause BSD License”. Viitattu 15.3.2026. (), url: https://opensource.org/license/BSD-3-clause.

[24] MIT License. ”The MIT License (MIT)”. Viitattu 15.3.2026. (2026), url: https://mit-license.org/.

[25] QNX. ”High-Performance Embedded Solutions”. Viitattu 15.3.2026. (2026), url: https://qnx.software/.

[26] Green Hills Software. ”INTEGRITY RTOS”. Viitattu 15.3.2026. (2026), url: https://ghs.com/products/rtos/integrity.html.

[27] Wind River. ”VxWorks RTOS | Real-Time Operating System | Wind River”. Viitattu 15.3.2026. (2026), url: https://www.windriver.com/products/embedded/vxworks.

[28] M.-W. Wu ja Y.-D. Lin, ”Open source software development: an overview”, *Computer*, vol. 34, nro 6, s. 33–38, 2001. DOI: 10.1109/2.928619.

[29] Open Source Initiative. ”OSI Approved Licenses”. Viitattu 15.3.2026. (2025), url: https://opensource.org/licenses.

[30] micro-ROS. ”micro-ROS”. Viitattu 15.3.2026. (2026), url: https://micro.ros.org/.

[31] C. Bormann, M. Ersue ja A. Keränen, ”Terminology for Constrained-Node Networks”, IETF, RFC 7228, 2014. DOI: 10.17487/RFC7228. url: https://www.rfc-editor.org/rfc/rfc7228.

[32] ArduPilot. ”ArduPilot - Versatile, Trusted, Open”. Viitattu 16.3.2026. (2024), url: https://ardupilot.org/.

[33] PX4 Autopilot. ”Open Source Autopilot for Drones - PX4 Autopilot”. Viitattu 16.3.2026. (2026), url: https://px4.io/.

[34] T. Kim, ”A Practical Cache Partitioning Method for Multi-Core Processor on a Commercial Safety-Critical Partitioned RTOS”, *IEEE Access*, 2025.

[35] A. Essetty, A. Daghouri, S. Bah ja Z. Guennoun, ”Integrating a Real Time Operating-System Layer to a Memory-Constrained Electrical Power System for a Nanosatellite”, teoksessa *International Conference on Electrical Systems and Smart Technologies*, Springer, 2024, s. 315–323.

[36] J. Ren ja D. Xia, *Autonomous driving algorithms and Its IC Design*. Springer, 2023.

[37] Apache Software Foundation. ”Apache NuttX”. Viitattu 15.3.2026. (2026), url: https://nuttx.apache.org/.

[38] ChibiOS. ”ChibiOS Homepage”. Viitattu 16.3.2026. (2026), url: https://www.chibios.org/dokuwiki/doku.php.

[39] C.-F. Yang ja Y. Shinjo, ”Compounded Real-Time Operating Systems for Rich Real-Time Applications”, *IEEE Access*, vol. 13, s. 26 079–26 104, 2025. DOI: 10.1109/ACCESS.2025.3538561.