REAALIAIKAKÄYTTÖ-JÄRJESTELMÄ MULTIKOPTERIN LENNONOHJAUSTIETOKONEESSA

Olli Paloniemi

Opinnäytetyö
Toukokuu 2015
Tietotekniikka
Sulautetut järjestelmät ja elektroniikka

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikka
Sulautetut järjestelmät ja elektroniikka

PALONIEMI, OLLI:
Reaaliaikakäyttöjärjestelmä multikopterin lennonohjaustietokoneessa

Opinnäytetyö 48 sivua, joista liitteitä 9 sivua
Toukokuu 2015

Tämän opinnäytetyön tarkoituksena oli tarkastella multikopterin lennonohjaustietokoneelle tehdyn laiteohjelmiston suunnitteluun liittyviä asioita. Laiteohjelmiston pohjana toimi avoimen lähdekoodin FreeRTOS-reaaliaikakäyttöjärjestelmä. Laiteohjelmiston testaus tapahtui tarkoitusta varten rakennetussa lennonohjaustietokoneessa, joka asennettiin neliroottoriseen multikopteriin.

Työssä tutkittiin multikopterien ohjausmenetelmiä. Radio-ohjauselektroniikasta tuttua pulssinleveyden modulaatioon perustuvaa signaalia käytettiin yhdistämään lennonohjaustietokone multikopterin muihin komponentteihin. Lennonohjaustietokoneen antureita ja radiovastaanottimelta tulevaa ohjausta hyödynnettiin PID-säätimissä, joiden on tarkoitus vakauttaa multikopterin lento. Antureiden mittaustulokset yhdistettiin anturifuusiolla käyttäen Complementary-suodinta. Säätimiä ja anturifuusiota käytettiin eri ohjaustiloissa, jotka määrittävät sen, miten multikopterin lennonohjausjärjestelmä toimii. Lennonohjausjärjestelmän toiminnan reaaliaikaisuus todettiin mittaamalla käyttöjärjestelmän suoritus- ja vasteajat.

Opinnäytetyön avulla saatiin tietoa muun muassa säätötekniikasta, anturijärjestelmistä, anturifuusiosta, radio-ohjauksesta ja telemetriasta. Etenkin tietoa saatiin näiden kaikkien osa-alueiden soveltamisesta laiteohjelmistossa, jonka perustana on reaaliaikakäyttöjärjestelmä. Tämän opinnäytetyön tekemisessä syntyneitä ideoita on tarkoitus hyödyntää lennonohjausjärjestelmän jatkokehityksessä. Niistä voi olla hyötyä myös muun tyyppisten sulautettujen järjestelmien ja näiden laiteohjelmistojen kehityksessä, jotka ovat riippuvaisia samoista teknologioista.

Asiasanat: uav, multikopteri, sulautettu järjestelmä, reaaliaikakäyttöjärjestelmä

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree programme in ICT Engineering
Embedded Systems and Electronics

PALONIEMI, OLLI:
Real-Time Operating System on a Multirotor Flight Control Computer

Bachelor's thesis 48 pages, appendices 9 pages
May 2015

The purpose of this bachelor's thesis was to study the design and implementation of a firmware for a multirotor flight control computer. The firmware of the flight control computer was based on FreeRTOS, an open source real-time operating system. The testing of the software was done in a purpose built flight control computer, which was installed on a quadcopter.

The control methods of a multirotor were studied. Pulse-width modulation, a technique that is used frequently in radio-controlled electronics, was utilized to interconnect the components of the multirotor. The sensors on the flight control computer and the control signal from the radio-control receiver were utilized in PID-control algorithm, to stabilize the flight of the multirotor. Sensor data was combined using Complementary filter, a sensor fusion algorithm. The combinations of control and sensor fusion algorithms were used to create different flight control modes. The flight control modes define how the flight control system operates. Real-time operation of the flight control system was verified by measuring the execution and response times of the operating system.

This thesis provides information about control technology, sensor systems, sensor fusion, radio-control and telemetry. Information was obtained particularly about the combination of all these technologies in a firmware, which is based on a real-time operating system. In the future, the author of this thesis plans to utilize the ideas developed for this thesis to further develop the flight control system. These same ideas might be useful in other embedded system firmwares that are dependent on these same technologies.

Key words: uav, multirotor, embedded system, real-time operating system

SISÄLLYS

LYHENTEET JA TERMIT

UAV	Unmanned Aerial Vehicle, miehittämätön ilma-alus.
ISM	Industrial, Scientific and Medical radio bands. Joukko radiotaajuuskaistoja, jotka ovat vapaasti käytettävissä rajoitetulla lähetysteholla.
PWM	Pulse-width Modulation, pulssinleveysmodulaatio.
GPS	Global Positioning System. Yhdysvaltalainen satelliittipaikannusjärjestelmä.
PID	Proportional-Integral-Derivative. Kolmea säädintermiä käyttävä menetelmä säätötekniikassa.
AHRS	Attitude Heading Reference System. Järjestelmä joka kertoo lentokoneen asennon pituus-, pysty- ja vaaka-akseleilla maanpintaan ja kompassin pohjoissuuntaan verrattuna.
IIR	Infinite Impulse Response. Digitaalinen suodatusmenetelmä.
RTOS	Real-Time Operating System, reaaliaikakäyttöjärjestelmä.
FIFO	First In, First Out. Menetelmä jossa tiedonsiirtopuskuriin ensimmäisenä kirjoitettu tavu on myös ensimmäinen lukiessa saatava tavu.
GPL	GNU General Public license. GNU-projektin kehittämä avoimen lähdekoodin ohjelmistolisenssi.
BSD	Berkeley Software Distribution license. University of California, Berkeley –yliopistossa kehitetty avoimen lähdekoodin ohjelmistolisenssi.
IMU	Inertial Measurement Unit, inertiamittausyksikkö. Laite johon on integroitu useita eri anturitekniikoita laitteen liikkeiden havaitsemista varten.
AVR	Atmel-yhtiön kehittämä mikrokontrolleriperhe.
UART	Universal Asynchronous Receiver Transmitter. Sarjamuotoisen tiedon lähetys ja vastaanotto asynkronisesti eli tahdistamattomasti.
TCP	Transmission Control Protocol. Yhteydellinen tietoliikenneprotokolla.
PC	Personal Computer, henkilökohtainen tietokone.

Qt	Alustariippumaton ohjelmistojen kehitysympäristö PC- ja mobiililaitteille.
TCP/IP	Transmission Control Protocol / Internet Protocol. IP-protokollan käyttö OSI-mallin mukaisella verkkokerroksella laitteiden osoitteita ja pakettien reitittämistä varten, ja TCP-protokollan käyttö kuljetuskerroksella yhteydellisestä tiedonsiirtoa varten.
I²C	Inter-Integrated Circuit. Kahta johdinta käyttävä tiedonsiirtoväylä.
CPU	Central Processing Unit. Tietokoneen suoritin.

1 JOHDANTO

Moniroottorisesta pyöriväsiipisestä ilma-aluksesta, eli multikopterista on tullut yksi yleisimmistä muodoista miehittämättömälle pienoisilma-alukselle. Multikoptereita on kehitetty niin harrastajien kuin myös erityyppisiä kauko-ohjattavia UAV-lentolaitteita hyödyntävien yritysten käyttöön. Niiden käytössä multikoptereita on hyödynnetty mm. ilmakuvaukseen, voimalinjojen tarkastuksiin, alueiden valvontaan ja henkilöiden etsintään. Osa näistä multikoptereihin perustuvista sovelluksista on myös vaativammassa viranomais- ja sotilaskäytössä, jossa laitteiden toiminnan luotettavuudelle asetetut vaatimukset ovat paljon tiukemmat kuin harrastekäyttöön suunnitelluissa laitteissa.

Multikopterin lennonohjausjärjestelmän keskusyksikkönä toimiva lennonohjaustietokone on jatkuvan kehitystyön kohteena. Edistyneimmät lennonohjausjärjestelmät kykenevät kauko-ohjatun lennon lisäksi myös jo lähes täysin autonomiseen toimintaan. Multikopteri ja sen lennonohjaustietokone muodostavat vaativan reaaliaikajärjestelmän, jonka täytyy reagoida nopeasti muuttuviin olosuhteisiin. UAV-ominaisuudet tuovat järjestelmään omat lisävaatimuksensa, joiden toteuttaminen laiteohjelmistoon voi olla työlästä. Reaaliaikakäyttöjärjestelmä tarjoaa ympäristön, missä näiden ominaisuuksien ohjelmointi on helpompaa.

Tämän opinnäytetyön päämääränä on tarkastella reaaliaikakäyttöjärjestelmiä ja niihin perustuvan ohjelmiston kehitystyötä ja sovittamista multikopterille suunniteltuun lennonohjaustietokoneeseen. Kohdealustana on itse suunniteltu ja valmistettu lennonohjaustietokone ja tarkoitusta varten koottu multikopteri. Työssä käytetty reaaliaikakäyttöjärjestelmä on avoimeen lähdekoodiin pohjautuva FreeRTOS, mutta työn pyrkimyksenä on, että esitellyt menetelmät ovat yleispäteviä suurimpaan osaan markkinoilla olevista reaaliaikakäyttöjärjestelmistä. Tärkeä osa opinnäytetyötä on ohjelmisto käyttöjärjestelmän vaste- ja suoritusaikojen mittaamiseen, jolla laiteohjelmiston reaaliaikaisten osien toiminta pystytään varmistamaan.

2 MULTIKOPTERIT

2.1 Määritelmä

Multikopteri on pyöriväsiipinen ilma-alus, jossa nostovoiman tuottavana osana on kaksi tai useampia roottoreita. Multikopterin kokoonpano eli konfiguraatio määrittää roottorien asettelun ja niiden pyörimissuunnat. Erilaisia konfiguraatioita ovat mm. tri-, quad-, heksa- ja oktokopteri. Tästä nimeämiskäytännöstä käy ilmi roottorien lukumäärä. Roottorien lukumäärän lisäksi konfiguraatioon vaikuttaa se, missä kulmassa roottorijärjestelmä on kopterin vaaka- tai pituusakseliin verrattuna. Neliroottorinen multikopteri voi olla joko + tai X-konfiguraatiossa. Esimerkki X-konfiguraatiossa olevasta quadkopterista on kuvassa 1, jossa kuvan yläreuna on multikopterin etupuoli.

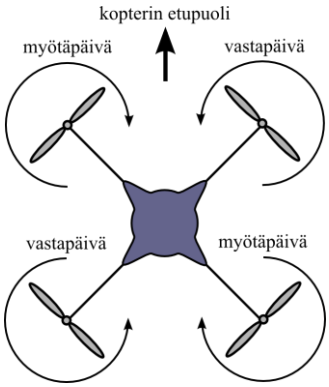
KUVA 1. Neliroottorinen multikopteri eli quadkopteri X-konfiguraatiossa

Elektronisen multikopterin toiminta eroaa tavanomaisesta helikopterista siten, että sen potkurien lapakulmat ovat yleensä kiinteät. Tavanomainen yhdellä nostovoimaa tuottavalla roottorilla varustettu helikopteri sisältää mekanismin roottorin lapakulmien muut-

tamiseen, jolla mahdollistetaan kopterin ohjaus. Yhden roottorin käyttö aiheuttaa vääntömomentin, joka saa helikopterin rungon kiertämään roottorin pyörimissuuntaa vastaan. Tämä vääntömomentti kumotaan helikopterissa kiertymissuunnan vastaisella pyrstöroottorilla. Multikopterissa vääntömomentin kumoaminen tapahtuu vastakkaisiin suuntiin pyörivillä roottoreilla ja ohjaus roottorien pyörimisnopeuksia säätämällä (Castillo 2005, 41). Elektronisen multikopterin yksinkertaisempi mekaniikka tekee siitä edullisemman yksiroottoriseen helikopteriin verrattuna.

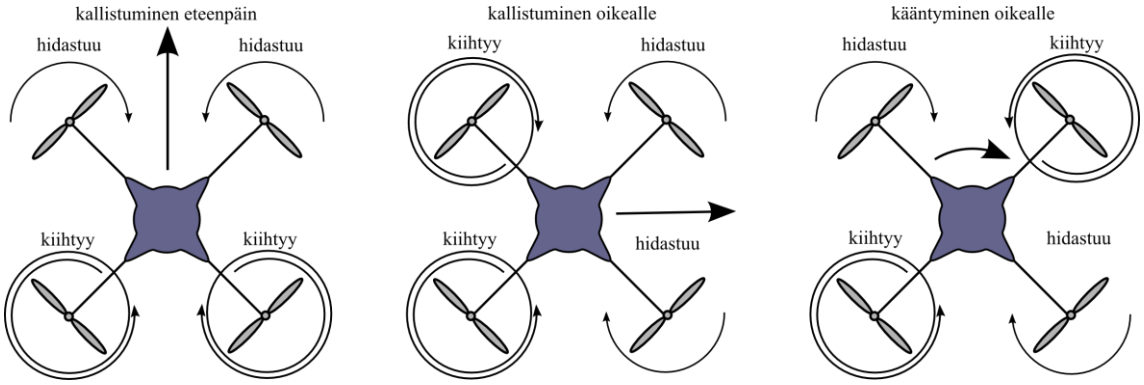
2.2 Multikopterin lentomekaniikka

Multikopterin tuottama nostovoima on sen kaikkien roottorien tuottaman nostovoiman summa. Quadkopterissa vastakkaisilla puolilla runkoa olevat roottorit muodostavat samaan suuntaan pyörivät parit. Kun puolet roottoreista pyörii myötäpäivään ja puolet vastapäivään (kuva 2), kumoutuvat multikopterin runkoon kohdistuvat vääntömomentit. (Castillo 2005, 42.) Täysin ideaaleissa olosuhteissa kopteri nousee tasaisesti, kun sen roottorit pyörivät samalla nopeudella ja kun niiden tuottama kokonaisvoima ylittää voiman jolla maa vetää puoleensa kopterin massaa. Todellisuudessa kopterin nousuun vaikuttaa paljon suurempi määrä muuttujia. Esimerkiksi pienet erot sähkömoottorien pyörimisnopeuksissa, propellien muodossa ja olosuhteet kuten tuuli ja maavaikutus aiheuttavat sen, että multikopterin ei ole mahdollista pysyä vakaana ilmassa, vaikka sen roottorien oletettaisiin pyörivän samalla nopeudella. Jotta olisi helpompi ymmärtää multikopterin käyttäytymistä, on kuitenkin hyödyllistä tarkastella sen ohjausmenetelmiä täysin ideaalin tilanteen näkökulmasta.



KUVA 2. Quadkopterin roottorien pyörimissuunnat

Täysin ideaaleissa olosuhteissa horisontin tasossa oleva multikopteri leijuu vakaasti ilmassa, kun sen roottorit tuottavat voiman, joka kumoaa painovoiman vaikutuksen. Tästä lähtökohdasta multikopteria on mahdollista ohjata muuttamalla sen roottorien pyörimisnopeuksia. Kun pyörimisnopeuksia muutetaan, pyritään voimien summan muodostama kokonaisvoima pitämään samana. X-konfiguraatiossa oleva quadkopteri saadaan kallistumaan eteenpäin, kun sen edessä olevien roottorien pyörimisnopeutta lasketaan ja takana olevien roottoreiden nopeutta kasvatetaan (kuva 3). Tämän seurauksena multikopteri liikkuu eteenpäin, mutta menettää myös korkeutta. Korkeuden menetys voidaan kompensoida lisäämällä kokonaisvoimaa. Samalla tavalla kopteri saadaan liikkumaan myös taaksepäin, jos roottorien nopeuksia edessä kasvatetaan ja takana lasketaan. Multikopteri saadaan kallistumaan jommalle kummalle sivulle, jos toisen puolen roottoreiden nopeutta lasketaan ja toisen puolen roottoreiden nopeutta kasvatetaan, kuten kuvan 3 keskellä.

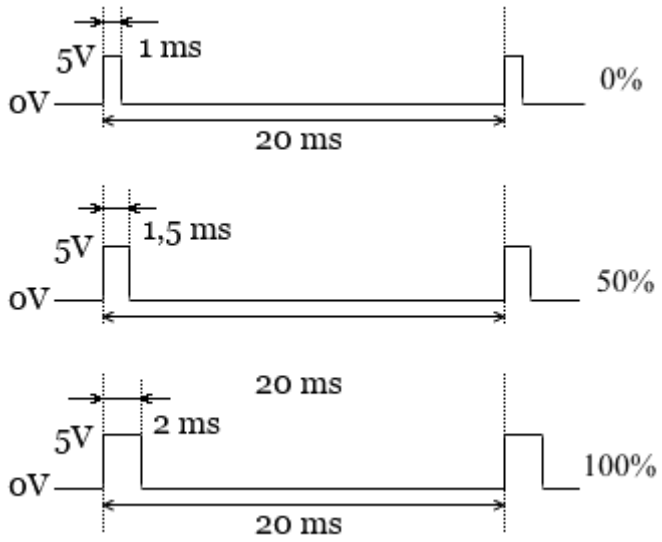


KUVA 3. Quadkopterin ohjaus roottorien pyörimisnopeuksia muuttamalla

Multikopterin on mahdollista kääntyä pystyakselinsa ympäri muuttamalla vastakkaisiin suuntiin pyörivien roottorien kierroslukujen suhdetta. Kun myötäpäivään pyörivien roottorien nopeutta hidastetaan ja vastapäivään pyörivien roottorien nopeutta kasvatetaan, pyörii multikopteri pystyakselinsa ympäri oikealle (kuva 3). Tässä tapauksessa multikopteri ei menetä korkeuttaan, jos kokonaisvoima pidetään muuttumattomana. Kiinteälapakulmaisia potkureita käyttävän multikopterin ohjaus perustuu edellä mainittujen menetelmien yhdistelmään. Näihin lisätään vielä lennonohjausjärjestelmän päättelemät korjaukset, joilla multikopterin lento mahdollistuu epäideaaleissa olosuhteissa. Lennonohjausjärjestelmä sisältää tarvittavat anturit ja säätötekniikan multikopterin lennon vakauttamiseen.

2.3 Radio-ohjauselektroniikka

Radio-ohjattavassa multikopterissa on radiovastaanotin, jolla vastaanotetaan radio-ohjaimen signaali. Harrastekäyttöön suunnatussa lennokkielektroniikassa tämä on erillinen komponentti, joka on suunniteltu toimimaan tietyllä taajuusalueella ja tottelee yleensä vain moduulin valmistajan omaa protokollaa. Yksi yleisimmin käytetyistä taajuuskaistoista on monikäyttöinen 2,4 GHz:n ISM-alue. Radiomoduuli antaa vastaanotetut ohjauskanavat lähtöjohtimistaan pulssinleveysmoduloituna eli PWM-signaalina (kuva 4). Ohjauskanavan PWM-signaali voi sellaisenaan hallita servomoottoria tai harjattoman sähkömoottorin nopeudensäädintä. Multikopterissa nämä ohjaussignaalit syötetään lennonohjaustietokoneelle, jossa vastaanotettu signaali vaikuttaa osana kopterin lennonohjausjärjestelmää.



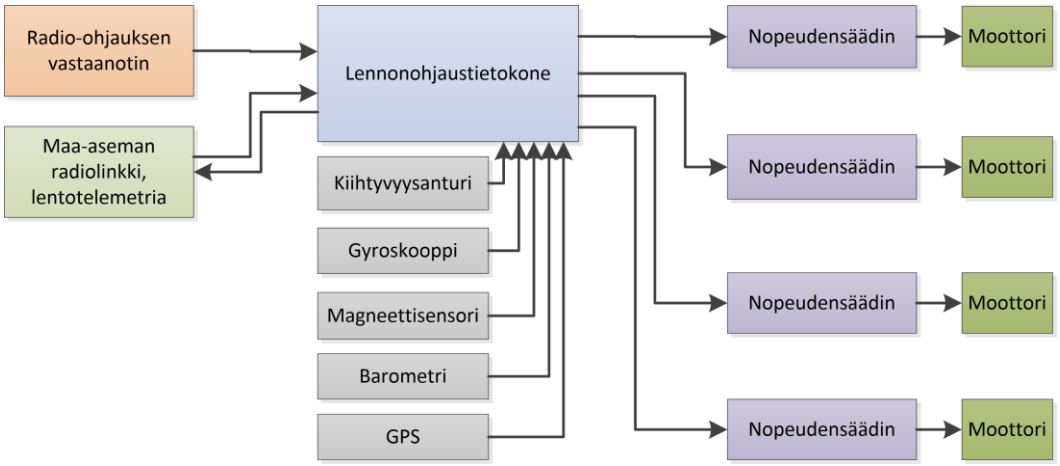
KUVA 4. PWM-signaalin ääripäät ja keskikohta 50 Hz taajuudella

Lennonohjaustietokoneessa on lähdöt multikopterin jokaisen moottorin nopeuden säätimelle. Näistä lähdöistä annetaan PWM-signaalit, joilla määrätään pyörimisnopeudet jotka nopeudensäätimet asettavat. Koska harrastekäyttöön suunnitelluissa edullisissa nopeudensäätimissä ei ole keinoa pyörimisnopeuksien kertomiseksi takaisin lennonohjaustietokoneelle, ei lennonohjaustietokone tiedä todellisia pyörimisnopeuksia. Syötettävien ohjaussignaalien tasoista voidaan päätellä vain roottoreiden suhteelliset pyörimisnopeudet. Tämä rajoitus ei haittaa multikopterin toimintaa, jos kunkin roottorin nopeudensäädin, moottori ja propelli ovat identtisiä.

3 LENNONOHJAUSJÄRJESTELMÄ

3.1 Lennonohjaustietokone

Multikopterin lennonohjausjärjestelmän ydin on lennonohjaustietokone. Se yhdistää toisiinsa radio-ohjausvastaanottimelta tulevien ohjaussignaalien lisäksi maa-aseman radiolinkin, joukon antureita ja moottoreiden nopeudensäätimet (kuva 5). Lennonohjaustietokoneen voidaan siis ajatella olevan multikopterin toimintaa ohjaava hermokeskus. Kuvassa 5 olevan kaavion tulopuolelle ei minimikokoonpanossa tarvittaisi muita lohkoja kuin radio-ohjauksen vastaanotin ja kolmiakselinen gyroskooppi. Edistyneemmät lennonohjaustietokoneet kytkeytyvät kuitenkin suureen määrän anturitekniikka. Usean eri anturitekniikan yhdistämisen eli anturifuusion avulla voidaan toteuttaa helpommin hallittavissa olevia manuaalisia radio-ohjaustiloja.



KUVA 5. Multikopterin järjestelmät

Satelliittipaikannusjärjestelmän, kuten GPS, paikkatiedon avulla voidaan multikopteri ohjelmoida lentämään autonomisesti. Tämä tapahtuu yleensä ennalta ohjelmoitua reittiä pitkin. Autonomisia toimintoja hallitaan maa-asemasta, johon lennonohjaustietokone on yhteydessä radiolinkin kautta. Maa-asema toimii myös lentotelemetrian vastaanottimena. Lentotelemetria on lennonohjaustietokoneen lähettämää dataa, johon voi lukeutua mm. barometrilla mitatusta ilmanpaineesta laskettu lentokorkeus, AHRS-järjestelmän avulla arvioitu multikopterin asento, akkupaketin jännite ja GPS-paikkatiedot. Multikopterin asento voidaan esittää maa-asemassa esimerkiksi keinohorisontin avulla ja paikkatieto karttanäkymässä.

3.2 Säätötekniikka

Lennonohjaustietokone hyödyntää säätötekniikkaa pitääkseen multikopterin asennon vakaana. Säädin on järjestelmä, joka ylläpitää automaattisesti jonkin suureen arvon halutussa tasossa. Säädinjärjestelmä koostuu vähintään yhdestä säätimestä, aktuaattorista ja prosessista. Aktuaattori on se osa järjestelmää, joka manipuloi jotakin fyysistä prosessia. Fyysisen prosessin lopputulos on se, mitä halutaan säätää. (Killian 2006, 2.) Jotta säätimen asetusarvo voitaisiin saavuttaa, täytyy prosessin tulosta mitata ja käyttää tätä hyödyksi säätimen ulostulon muodostamisessa. Tätä kutsutaan takaisinkytkennäksi. Mittausarvon ja asetusarvon erotuksesta saadaan erosuure (Killian 2006, 6.).

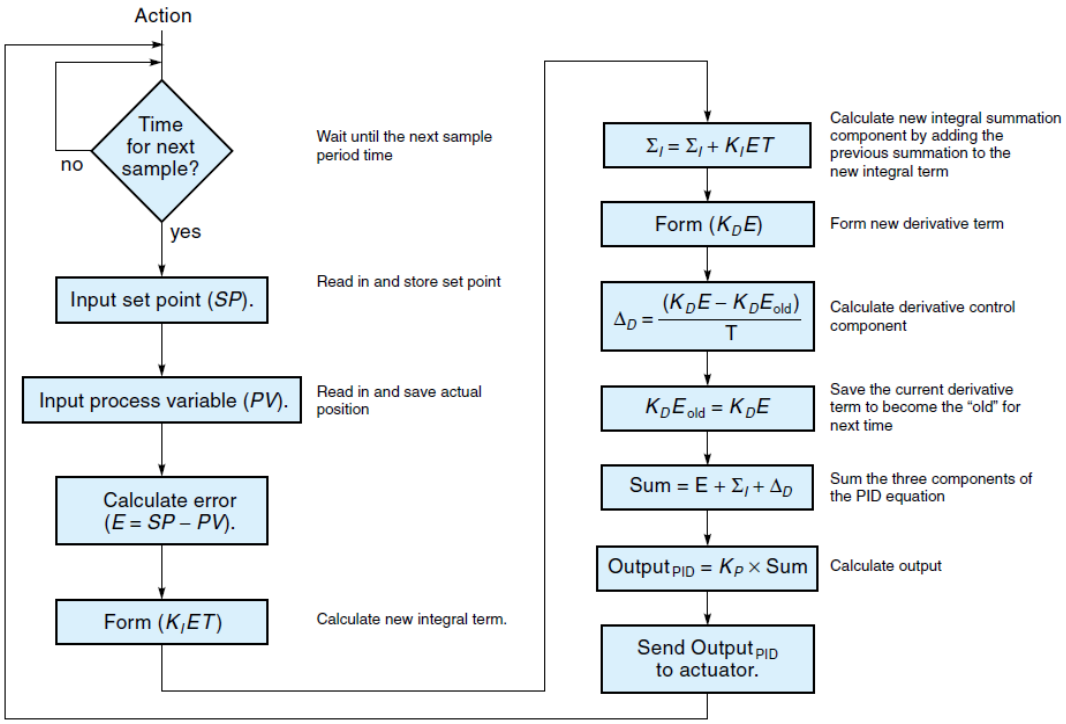
Multikopterin tapauksessa roottorit muodostavat järjestelmän aktuaattorit ja multikopterin liikkeet ovat se prosessi, mitä säätimellä tahdotaan hallita. Lennonohjaustietokoneeseen kytketyistä antureista saadaan prosessin mittausarvot. Mittausarvoja voidaan käsitellä, ennen kuin niitä käytetään erosuureen laskemisessa. Niitä voidaan esimerkiksi suodattaa ali- tai ylipäästösuotimilla, ja niiden tuloksia voidaan yhdistää anturifuusioalgoritmeilla. Säätimien asetusarvot saadaan lennonohjaustietokoneen ohjaustilasta riippuen joko suoraan radiovastaanottimen PWM-signaaleista mitatuista arvoista tai monimutkaisemmissa ohjaustiloissa toisten säätimien ulostuloista.

PID-säädin koostuu kolmen eri säätötermin vaikutusten yhdistelmästä. Nämä ovat suhdesäätötermi (Proportional), integraalitermi (Integral) ja derivaattatermi (Derivative). Suhdesäätötermin vaikutuksesta aktuaattori tekee korjaavan toimenpiteen suhteessa erosuureeseen (Killian 2006, 459). Integraalitermin vaikutuksesta korjaavan toimenpiteen voimakkuus määräytyy suhteessa erosuureen aikaisempiin arvoihin kerrottuna ajan muutoksella (Killian 2006, 470). Derivointitermi määrä sen, kuinka suuri vaikutus erosuureen arvon muutosnopeudella on aktuaattoriin toimintaan. Näiden yhdistelmästä muodostetun PID-säätimen toimintaa voidaan kuvata seuraavalla yhtälöllä.

$$Output_{PID} = K_P E + K_I \sum E \Delta t + K_D \frac{\Delta E}{\Delta t}$$

missä $Output_{PID}$ on säätimen ulostulo, K_P on suhdesäätötermi, K_I on integrointitermi, K_D on derivointitermi, E on erosuure, Δt on ajan muutos ja ΔE on erosuureen muutos. (Killian 2006, 478)

PID-säädin voidaan toteuttaa ohjelmistopohjaisesti kuvan 6 sisältämän kaavion mukaisesti. Kuvan 6 mukainen algoritmi toimii kiinteällä aikavälillä, jonka kuluttua otetaan näytteet asetusarvosta ja mittausarvosta. Näistä lasketaan erosuure ja tämän jälkeen säädinparametrien mukaiset vaikutukset summataan ulostulomuuttujaan. Lopuksi ulostulomuuttujan arvo lähetetään aktuaattoriin.



KUVA 6. PID-säädinohjelman kaavio (Killian 2006)

Käytännössä PID-säätimen ohjelmointi C-kielellä voidaan tehdä kuvan 7 koodin mukaisesti. Aliohjelma pid_laske ottaa kaksi parametria; osoittimen PID-säätimen asetukset sisältävään tietueeseen ja kokonaisluvun, joka sisältää eroarvon. Säätötermit löytyvät osoittimen välityksellä muuttujista k_p, k_i ja k_d. PID-säätö tehdään kolmessa vaiheessa, joissa kunkin termin vaikutus summataan ulostulo-muuttujaan. Integrointitermin vaikutuksen laskemisessa käytettävän integraattori-muuttujan koko on rajattu Processing- ja Arduino-ohjelmointiympäristöistä lainatulla constrain-makrolla, muuttujan i_maksimi määrittämiin rajoihin. Derivointitermin vaikutuksen laskemisessa käytettävään derivaattori-muuttujaan sijoitetaan arvo d_suodatus aliohjelmasta. Tässä aliohjelmassa tehdään alipäästösuodatus derivaattorille, jolla vaimennetaan eroarvon muutoksissa mahdollisesti esiintyviä korkeataajuisia komponentteja.

```
int32_t pid_laske(struct PID *pid, int32_t eroarvo)
{
    int32_t ulostulo = 0;

    // Suhdesäätötermin vaikutus
    ulostulo += pid->k_p * eroarvo;

    // Integrointitermin vaikutus
    if (pid->k_i != 0){
        // Summataan integraattoriin integrointitermin, eroarvon ja aikavälin tulo
        pid->integraattori += pid->k_i * eroarvo * pid->dt;

        // Rajataan integraattorin koko
        pid->integraattori = constrain(pid->integraattori, -pid->i_maksimi, pid->i_maksimi);

        // Summataan integraattori ulostuloon
        ulostulo += pid->integraattori;
    }

    // Derivointitermin vaikutus
    if (pid->k_d != 0){
        // Lasketaan uusi derivaattorin arvo
        float uusi_derivaattori = (eroarvo - pid->eroarvo) / pid->dt;

        // Tallennetaan uusi eroarvo vanhan tilalle
        pid->eroarvo = eroarvo;

        // Suodatetaan uusi derivaattorin arvo ja tallenetaan vanhan tilalle
        pid->derivaattori = d_suodatus(uusi_derivaattori);

        // Summataan derivointitermin ja derivaattorin tulo ulostuloon
        ulostulo += pid->k_d * pid->derivaattori;
    }

    return ulostulo;
}
```

KUVA 7. PID-säätimen koodi

3.3 Anturifuusio

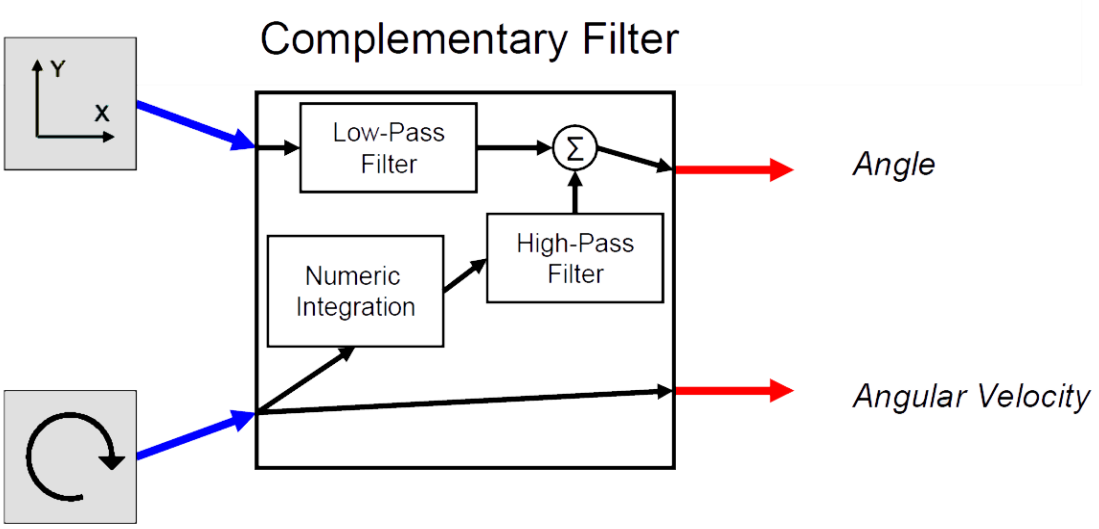
Anturifuusiolla tarkoitetaan useiden anturien tuottamien mittaustulosten yhdistämistä. (Castillo 2005, 221) Anturifuusiota voidaan käyttää mm. AHRS-järjestelmässä tuottamaan luotettava arvio anturijärjestelmän asennosta. Yksittäisellä anturilla, kuten kulmanopeutta mittaavalla gyroskoopilla, voidaan tuottaa myös arvio anturin asennon muutoksista integroimalla kulmanopeus. Gyroskoopin ongelma on kuitenkin ryömintä (drift), jonka vuoksi integroimalla arvioitu asento ei pysy muuttumattomana vaikka anturi olisi liikkumatta (Castillo 2005, 205.). Tämä on seurausta siitä kun gyroskoopin ilmoittama kulmanopeus poikkeaa nollasta vaikka anturi olisi liikkumatta. Ryömintä voidaan kumota käyttäen anturifuusiota yhdistämään kiihtyvyysanturin mittaustuloksien vaikutus gyroskoopilla arvioituun asentoon.

Multikopterien lennonohjausjärjestelmissä anturifuusioon soveltuvia ja yleisesti käytettyjä matemaattisia menetelmiä ovat mm. Kalman-suodin variaatioineen, suuntakosinimatriisi (ArduPilot 2015) sekä complementary filter (Multiwii 2014). Complementary filter, eli vapaasti suomennettuna täydentävä suodin, on anturifuusiomenetelmä jota harkittiin jo 1970-luvulla USA:n sukkulaohjelman käyttöön (Higgins 1975, 1).

Complementary filter (kuva 8) on matemaattisesti yksinkertainen, kun sitä verrataan muihin anturifuusiomenetelmiin. Kuvan 8 vasemmasta yläkulmasta tuodaan suotimeen kahden kiihtyvyysanturin mittauksista laskettu anturijärjestelmän kallistuskulma. Toinen antureista on pystyakselin ja putoamiskiihtyvyyden suuntainen järjestelmän ollessa tasaisella alustalla. Toinen kiihtyvyysanturi on joko vaaka- tai pituusakselin suuntainen. Kallistuskulma lasketaan trigonometrisesti näiden kahden välillä arkustangentin avulla. Kuvan 8 vasemmasta alakulmasta tuodaan suotimeen gyroskoopin ilmoittama kulmanopeus. Gyroskoopin täytyy olla sijoitettuna niin, että se mittaa kulmanopeutta 90° vaakatasossa vastakkaiselta akselilta kuin miltä toinen kiihtyvyysantureista mittaa kiihtyvyyttä. Tällöin kumpikin anturitekniikka tuottaa tietoa anturijärjestelmän liikkeistä samalta akselilta. Kulmanopeus integroidaan jolloin saadaan gyroskoopin arvio kallistuksesta. Kahden eri anturin mittaukset täydentävät toisiaan muodostaen lopullisen kulman.

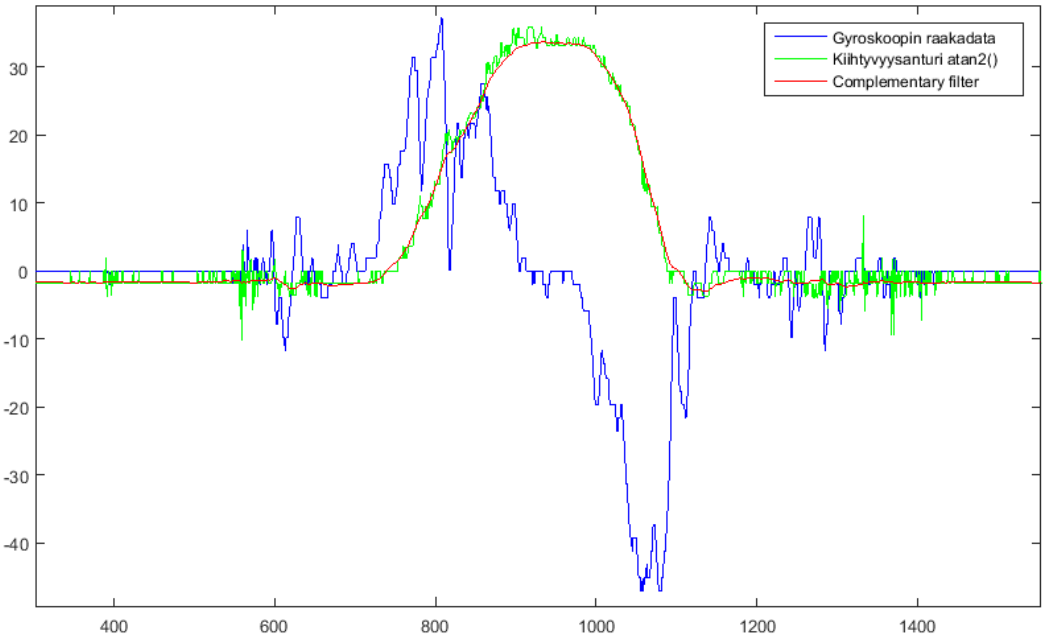
$$angle = \alpha * (angle + gyro * dt) + (1 - \alpha) * accelerometer$$

Tässä yhtälön vasemman puolen *angle* on uusi kulman arvio, vasemman puolen suluissa oleva *angle* on vanha kulman arvio, *gyro* on mitattu kulmanopeus, *dt* on näytteiden välinen aika. Nämä muodostavat integroimalla luodun arvion kulmasta. Muuttuja *accelerometer* on kiihtyvyysanturien mittauksista laskettu kulma. Anturien vaikutuksien väliset painoarvot, kertoimet α ja (1 - α), vaikuttavat yhtälössä IIR-tyyppisien yli- ja alipäästösuotimien tavoin. (Colton 2007, 14.)



KUVA 8. Complementary suotimen toiminnallisuutta kuvaava kaavio (Colton 2007)

Complementary-suotimen etuja ovat sen matemaattinen yksinkertaisuus, nopean suoritus ja sen myötä myös vähäinen suorittimen kuormitus. Complementary-suotimen toiminta voidaan nähdä kuvassa 9. Kuvaaja on tuotettu multikopterin pituusakselin kallistusta mittaavien anturien datasta. Kuvassa on gyroskoopin raakadata, kahden kiihtyvyysanturin muodostama kallistuskulma sekä lennonohjaustietokoneen AHRS-järjestelmän osana käytetyn Complementary-suotimen muodostama kallistuskulma. Datan tallennuksen aikana multikopteria kallistettiin noin 30 astetta pituusakselilla, ja tämän jälkeen se palautettiin takaisin tasaiselle alustalle.



KUVA 9. Gyroskoopin raakadata, kiihtyvyysanturien datasta arvioitu kulma ja Complementary-suotimen muodostama kulma

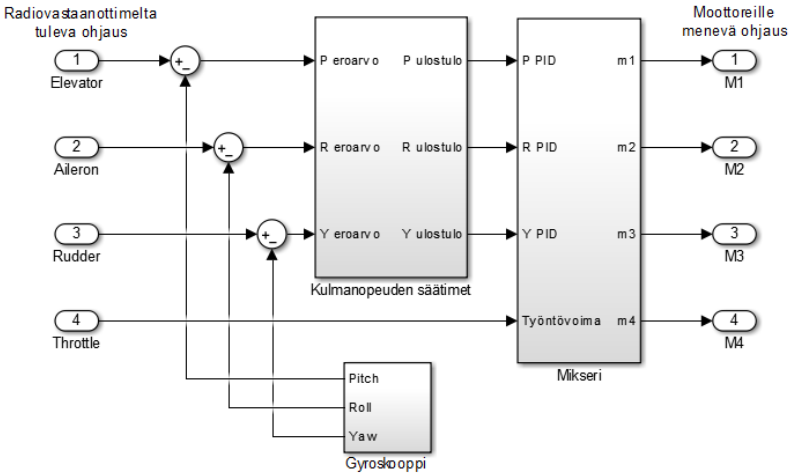
3.4 Multikopterin lennonohjausjärjestelmän ohjaustilat

Multikopterin lennonohjausjärjestelmä voi toimia useassa erilaisessa ohjaustilassa. Lennonohjaustietokoneeseen voi olla ohjelmoituna useita eri ohjaustiloja, joita voidaan vaihtaa esimerkiksi ohjaussignaalin välityksellä. Eri ohjaustilat määräävät sen, millä tavalla radio-ohjausta, säätötekniikkaa, anturidataa ja sen fuusiota käytetään lennonohjaustietokoneessa moottorien pyörimisnopeuksien laskemiseen. Ohjaustilat vaihtelevat manuaalisista lähes täysin autonomisiin, joissa multikopteri lentää ohjelmoitua reittiä pitkin. Vaativampi manuaalinen ohjaustila lisää multikopterin lentäjän työkuormaa ja määrittää sen, kuinka paljon taitoa multikopterin lentämiseen tarvitaan. Esimerkit eri ohjaustilojen englanninkielisistä nimityksistä on otettu kahdesta avoimen lähdekoodin lennonohjausjärjestelmästä, jotka ovat MultiWii ja ArduPilot.

3.4.1 Multikopterin kulmanopeuksiin perustuva ohjaustila

Eräs yksinkertaisimmista ohjaustiloista käyttää hyödykseen pelkästään kolmen akselin gyroskooppianturien mittaamia kulmanopeuksia. Siinä säätimillä pyritään pitämään

kulmanopeudet muuttumattomana. Lentäjän näkökulmasta tässä tilassa ohjaussauvan liike muuttaa multikopterin asentoa. Kun sauva vapautetaan, pyrkii multikopteri säilyttämään sen hetkisen asennon. Tämä on eräs yksinkertaisimmista ja myös vaativimmista ohjaustiloista. Koska tämä tila mahdollistaa ns. taitolennon, kutsutaan sitä englanniksi acrobatic-tilaksi tai lyhennettynä ACRO-tilaksi (MultiWii). Toinen nimi tilalle on RATE-tila (ArduPilot), jossa nimi viittaa kulman muutoksen nopeuteen (rate) eli kulmanopeuteen, jota ohjaustila hyödyntää. Kuvassa 10 nähdään yksinkertaistettu kaavio ohjaustilan toiminnasta. Radiovastaanottimelta tulevat ohjaussignaalit kertovat asetusarvot kulmanopeuksille. Kulmanopeuden säätimet -lohko pitää sisällään säätimet kullekin akselille. Mikseri-lohko laskee säätimien ulostuloista moottorien nopeudensäätimille asetettavat kierrosnopeudet.

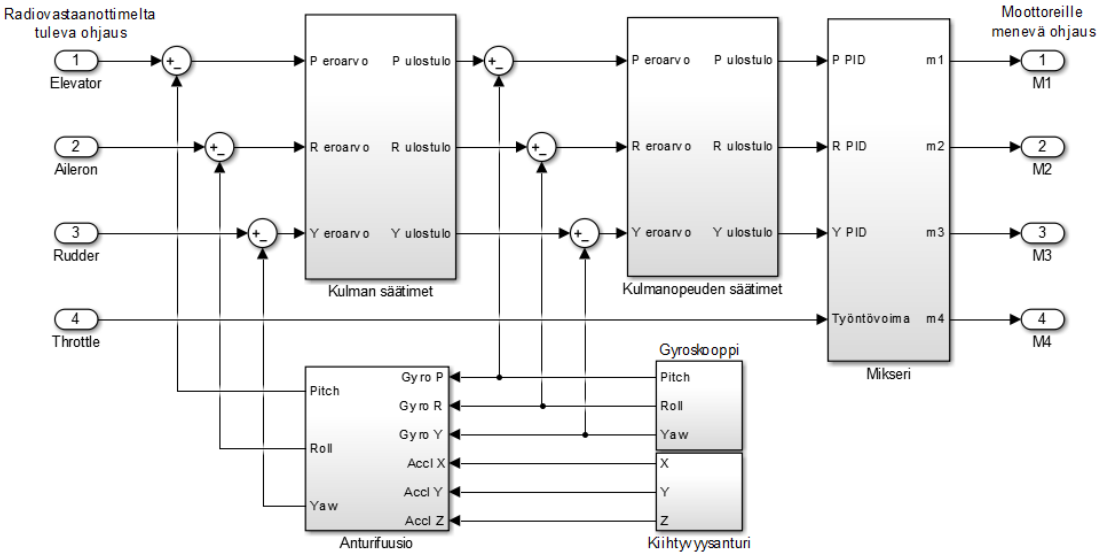


KUVA 10. Kaavio multikopterin kulmanopeuteen perustuvasta ohjaustilasta

3.4.2 Multikopterin asentoon perustuva ohjaustila

Tässä ohjaustilassa käytetään hyödyksi gyroskooppianturien mittaamia kulmanopeuksia sekä AHRS-järjestelmän tuottamaa arviota kopterin asennosta. Ohjaustilassa säätimillä pyritään pitämään multikopterin asento muuttumattomana ja horisontin tasossa. Lentäjän näkökulmasta tässä tilassa ohjaussauvan liikkeet kallistavat multikopteria pois horisontin tasosta. Kun sauva vapautetaan, palautuu multikopteri takaisin horisontin tasoon. Tämä ohjaustila on huomattavasti helpompi lentää kuin pelkkään kulmanopeuteen perustuva ohjaustila. Sitä kutsutaan stabilize-tilaksi (ArduPilot) ja ANGLE- tai ACC-tilaksi (MultiWii). ACC-nimitys tulee tilassa hyödynnetyistä kiihtyvyysantureista. Kuvassa 11 nähdään yksinkertaistettu kaavio ohjaustilan toiminnasta. Tämä ohjaustila on

laajennus kuvan 10 ohjaustilaan. Radiovastaanottimelta tulevat ohjaussignaalit kertovat asetusarvot kunkin akselin kulmalle. Anturifuusio-lohko tuottaa gyroskooppianturien ja kiihtyvyysanturien mittadatasta arvion multikopterin asennosta. Kulman säätimet -lohko pitää sisällään säätimet kullekin akselille. Sen ulostuloja käytetään asetusarvoina kulmanopeuden säätimille.



KUVA 11. Kaavio multikopterin asentoon perustuvasta ohjaustilasta

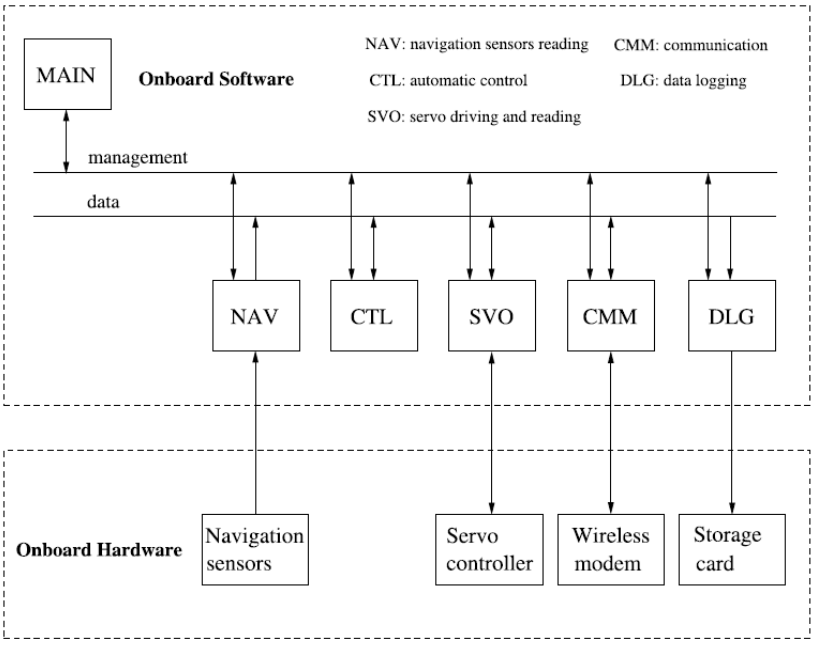
3.4.3 GPS-avusteiset ja autonomiset ohjaustilat

Edistyneemmät ohjaustilat lisäävät edellä kuvattujen ohjaustilojen päälle vielä lisää säädintasoja. Näiden säätimien asetusarvot voidaan muodostaa esimerkiksi GPS-paikkatiedoista. Paikkatiedon avulla multikopterin asemaa voidaan korjata, jos olosuhteet kuten tuuli siirtävät sitä. Tällä tavoin multikopteri saadaan pysymään samassa sijainnissa silloinkin, kun ohjausliikkeitä ei tehdä. Paikkatiedon avulla multikopterin toiminta voidaan automatisoida, ja se voi esimerkiksi lentää ohjelmoitua reittiä pitkin. Lennonohjausjärjestelmän autonomiset ohjaustilat tekevät multikopterista UAV:n eli autonomisen ilma-aluksen. Avoimen lähdekoodin lennonohjausjärjestelmät kuten MultiWii sekä ArduPilot sisältävät GPS-avusteiset ohjaustilat ja ohjelmoidun reittilennon mahdollistavat toiminnot.

3.5 Laiteohjelmiston toteutus

Lennonohjaustietokoneen laiteohjelmiston toteutus voidaan perustaa karkeasti jaettuna kahteen eri lähestymistapaan. Yksinkertaisimmillaan laiteohjelmiston toiminta perustuu yhteen suureen silmukkaan (foreground/background tai super-loop), jossa suoritetaan halutut toimet vuoron perään tai odotetaan keskeytyksien luomia asynkronisia tapahtumia (Labrosse 2002, 62). Tätä ratkaisua käytetään mm. MultiWii-lennonohjausjärjestelmässä. Silmukkaan perustuvan laiteohjelmiston koodin kanssa työskentelystä voi tulla kuitenkin hyvin vaikeata, kun järjestelmään lisättyjen toimintojen määrä kasvaa suureksi.

Toinen tapa lennonohjaustietokoneen laiteohjelmiston toteuttamiseen on tehtävien jakaminen useampaan silmukkaan, joiden välillä vaihdetaan vuorontajan avulla. Tätä varten laiteohjelmistoon tarvitaan reaaliaikaydin. Reaaliaikaytimen ympärille rakennettu käyttöjärjestelmä tarjoaa laiteohjelmistolle monia muita hyödyllisiä palveluita vuoronnuksen lisäksi. UAV-lennonohjausjärjestelmissä yleisesti käytettyjä reaaliaikakäyttöjärjestelmiä ovat mm. QNX Neutrino, VxWorks ja RTLinux (Guowei, Ben & Tong 2011, 12). Kuvassa 12 nähdään esimerkki UAV-lennonohjausjärjestelmän tehtävien jakamisesta useaksi eri prosessiksi. Yksittäisiä tärkeitä prosesseja voivat olla esimerkiksi moottoreiden ohjaus, anturidatan käsittely, navigaatio ja kommunikaatio.



KUVA 12. Esimerkki UAV lennonohjausjärjestelmän tehtävistä (Guowei, 2011)

4 REAALIAIKAKÄYTTÖJÄRJESTELMÄT

4.1 Reaaliaikajärjestelmä

Järjestelmä on kokonaisuus, joka yhdistää joukon tuloja ja lähtöjä (Laplante & Ovaska 2012, 3). Järjestelmän tuloihin tulee herätteitä ja sen lähdöistä saadaan vaste tietyn ajan kuluessa. Järjestelmän reaaliaikaisuudella tarkoitetaan sitä, että se täyttää sen vasteajoille annetut vaatimukset (deadlines). Se, mitä tapahtuu, jos vaatimuksia ei täytetä, riippuu siitä, onko järjestelmä määritelty kovaksi vai pehmeäksi. Kovassa reaaliaikajärjestelmässä yksikin vasteaikavaatimuksen täyttämättä jättäminen johtaa katastrofaaliseen virheeseen. Pehmeä reaaliaikajärjestelmä on huomattavasti anteeksiantavaisempi, vasteajoissa myöhästyminen vain heikentää järjestelmän suorituskykyä, mutta ei johda järjestelmälle tuhoisaan virheeseen. (Laplante & Ovaska 2012, 6.)

Käytännössä vasteaikavaatimus voi tarkoittaa järjestelmän reagointia johonkin ulkoiseen, reaalimaailmasta tulevaan herätteeseen. Mikrokontrollerissa tämä voi olla tulonastan tilamuutos, ajastimen arvon täsmäys vertailurekisterissä olevaan arvoon tai mikä tahansa muu oheislaitteen luoma keskeytyspyyntö. Näitä tapahtumia yhdistävä asia on kuitenkin se, että ne ovat asynkronisia. Reaaliaikajärjestelmä ei ole millään tavalla tietoinen siitä, milloin nämä herätteet tulevat tapahtumaan. Tästä huolimatta jokaiselle herätteelle on pystyttävä takaamaan enimmäisaika, jonka kuluttua saadaan vaste. Vasteajoille asetetaan järjestelmän toiminnan kannalta tarpeelliset vaatimukset, ja näiden toteutuminen tai toteuttamatta jättäminen kertoo, onko kyseinen järjestelmä tai sen osa kova vai pehmeä. Useimmissa käytännön reaaliaikajärjestelmissä on yhdistelmä sekä kovia että pehmeitä vasteaikavaatimuksia (Labrosse 2002, 48).

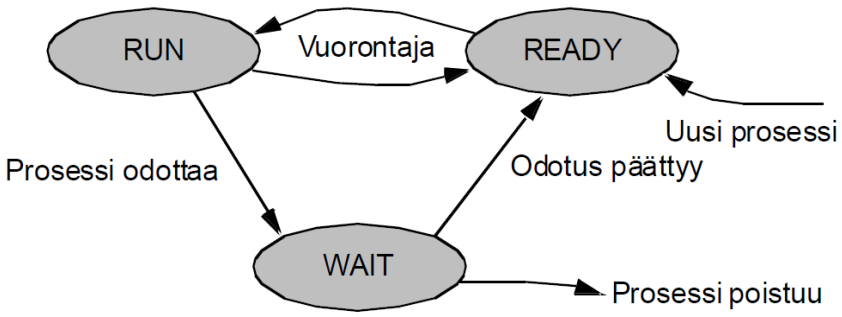
4.2 Käyttöjärjestelmä ja moniajo

Jotta toisistaan erillisten sovelluksien suorittaminen olisi mahdollista, tarvitaan laitteiston ja sovellusohjelmien väliin liitännäksi ohjelmisto, jota kutsutaan käyttöjärjestelmäksi. Käyttöjärjestelmässä on sovellusohjelmien toimintaympäristön lisäksi myös joukko palveluita. Reaaliaikakäyttöjärjestelmässä on ajastukseen, sovelluksien väliseen viestintään ja synkronointiin liittyviä järjestelmäkutsuja. Usean ohjelman yhtäaikaiseen suori-

tukseen kykenevää käyttöjärjestelmää kutsutaan moniajokäyttöjärjestelmäksi. Sovelluksien suoritus on kuitenkin yksisuoritinjärjestelmissä vain näennäisesti rinnakkaista (Haikala 2003, 88.). Yksi sovellus on kerrallaan ajettavana ja muut ovat pysäytettyinä. Irrottavaa vuoronnusmenetelmää käyttävässä reaaliaikakäyttöjärjestelmässä sovellukset eivät tiedä, että niiden suoritus pysäytetään ajoittain.

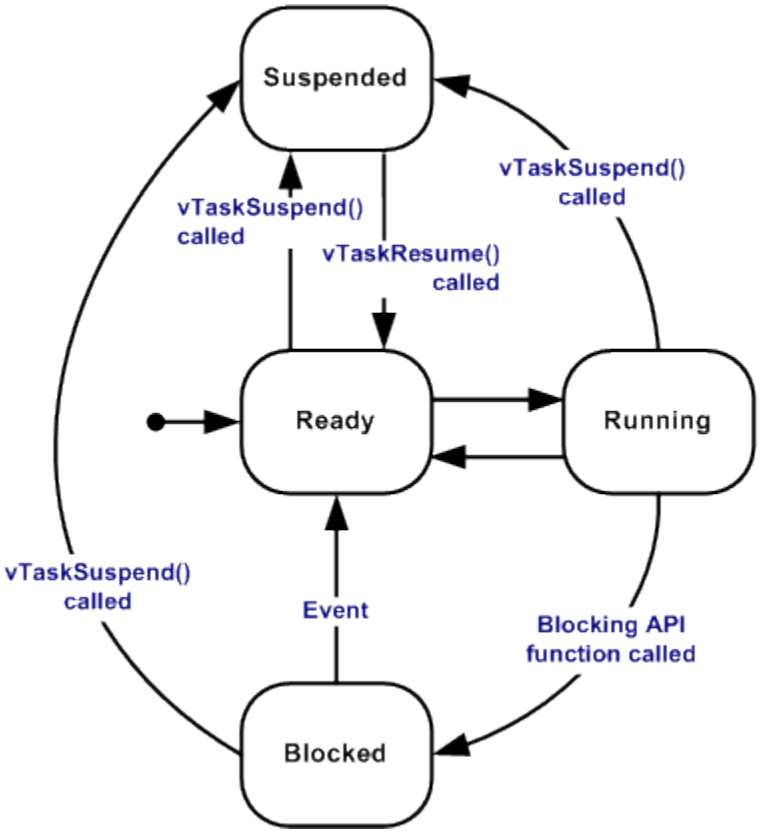
4.3 Prosessit

Käyttöjärjestelmässä suoritettavana olevia ohjelmia kutsutaan prosesseiksi. Vaihtoehtoinen nimitys prosessille on taski, jota käytetään usein reaaliaikakäyttöjärjestelmissä. Prosessilla on aina määriteltävissä oleva tila. Se on joko ajossa RUN-tilassa, odottamassa ajoon pääsyä READY-tilassa tai odottamassa jotain tapahtumaa WAIT-tilassa (kuva 13). Tiloja käyttöjärjestelmä säilyttää prosessielementissä. Se on prosessikohtainen tietorakenne, joka on olemassa jokaiselle olemassa olevalle prosessille. Prosessielementti sisältää myös muita käyttöjärjestelmän ytimelle tärkeitä tietoja, kuten prosessin prioriteetin. (Haikala 2003, 36.)



KUVA 13. Prosessin tilat (Haikala 2003)

Useissa käyttöjärjestelmissä on muitakin tiloja näiden kolmen lisäksi. FreeRTOS-reaaliaikakäyttöjärjestelmä sisältää näiden kolmen perustilan vastineiden lisäksi Suspended-tilan, joka on käytössä, kun taski pysäytetään järjestelmäkutsulla (kuva 14). Prosessielementtiä kutsutaan FreeRTOS-käyttöjärjestelmässä nimellä Task Control Block. Se sisältää taskin prioriteettiluvun, mutta ei taskin tilaa (Real Time Engineers Ltd. 2015.). Taskien tiloja FreeRTOS säilyttää erillisissä taulukoissa Ready- ja Blocked-tiloille. Taski voi siis olla vain yhdessä taulukossa kerrallaan. Tämä ratkaisu auttaa RTOS-ydintä toimimaan nopeammin, mikä on tarpeellista hitailla suorittimilla.



KUVA 14. FreeRTOS-taskin tilat (Real Time Engineers Ltd. 2015)

Taskille asetettu prioriteettinumero määrittää sen, mikä taski on käyttöjärjestelmän ytimen näkökulmasta kaikista tärkein. Korkeimman prioriteetin omaava Ready-tilassa oleva taski valikoituu seuraavaksi suoritukseen otettavaksi taskiksi. Järjestelmän toiminnan kannalta on tärkeää, että kukin taski siirtyy aina välillä Blocked-tilaan. Taskit voidaan laittaa esimerkiksi delay-funktiolla odottamaan aikamäärän kulumista. Muussa tapauksessa alemman prioriteetin taskit eivät pääse ollenkaan ajoon, kun yksi taski pitää suorittimen varattuna jatkuvasti.

4.4 Ydin

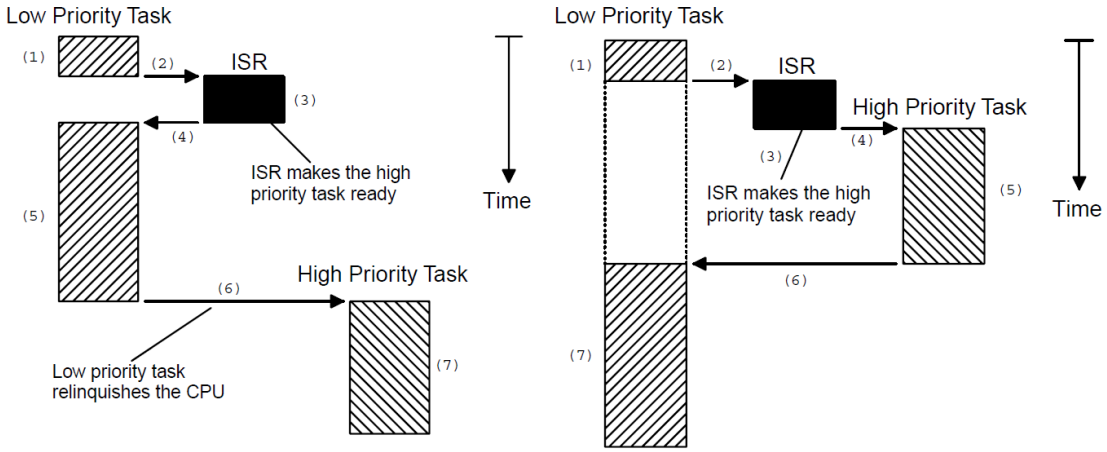
Kernel, eli käyttöjärjestelmän ydin, jakaa suoritinajan taskeille ja tarjoaa näille palveluja. Reaaliaikakäyttöjärjestelmän ydin on karsittu mahdollisimman pieneksi. Näin on tehty siksi, koska RTOS on monesti käytössä ympäristössä, missä tallennustilaa on vähän. Järjestelmän suunnittelu tehdään tarkoituksenmukaisesti. Tämän vuoksi ytimen mukana ei tule suurta määrää laiteajureita, kuten PC-käyttöjärjestelmässä, jonka täytyy toimia mahdollisimman monessa eri laitekokoonpanossa ja eri käyttötarkoituksissa. (Silberschatz, Galvin & Gagne 2005, 697.)

4.4.1 Vuorontaja

Vuorontaja on se osa ytimestä, joka päättää, mikä taski otetaan seuraavaksi ajoon. Prioriteettipohjaista menetelmää hyödyntävä käyttöjärjestelmäydin tekee tämän valinnan taskille asetetun prioriteettiluvun perusteella. Se, milloin käyttöjärjestelmä vaihtaa ajossa olevaa taskia, riippuu siitä, onko kyseessä irrottava vai ei-irrottava vuoronnusmenetelmä. Vuoronnusmenetelmä määrittää sen, milloin tapahtuu ympäristön vaihto (context switch). (Haikala 2003, 34.)

4.4.2 Irrottava ja ei-irrottava vuoronnusmenetelmä

Kuvan 15 vasemmassa piirroksessa nähdään esimerkki ei-irrottavasta vuoronnusmenetelmästä. Siinä ajossa oleva alemman prioriteetin taski pitää suoritinta hallussaan. Taskin suorituksen aikana tapahtunut keskeytys voi tuoda korkeamman prioriteetin taskin WAIT-tilasta READY-tilaan, mutta ympäristön vaihtoa ei tehdä kesken toisen taskin suorituksen. Korkeamman prioriteetin taski joutuu odottamaan kunnes alemman prioriteetin taski siirtää itsensä jollakin järjestelmäkutsulla joko WAIT- tai READY-tilaan. Tämän jälkeen vuorontaja päättää ottaa ajoon READY-tilassa odottaneen korkeimman prioriteetin taskin, jonka suoritus alkaa heti vuorontajan koodin jälkeen tapahtuvasta ympäristön vaihdosta.



KUVA 15. Ei-irrottava (vasen) ja irrottava (oikea) toimintaperiaate (Labrosse 2002)

Kuvan 15 oikeassa piirroksessa nähdään esimerkki irrottavasta vuoronnusmenetelmästä. Samaan tapaan kuin edellisessä esimerkissä, on alemman prioriteetin taski ajossa. Kes-

ken taskin suorituksen tullut keskeytys siirtää korkeamman prioriteetin taskin READY-tilaan. Koska nyt on kyseessä irrottava vuoronnusmenetelmä, on ytimellä lupa irrottaa suorituksessa ollut alemman prioriteetin taski. Vuorontaja päättää, että ajoon otetaan korkeamman prioriteetin taski, ja ympäristön vaihto tapahtuu kesken alemman prioriteetin taskin suorituksen.

Opinnäytetyötä varten ohjelmoidussa lennonohjausjärjestelmässä käytetään FreeRTOS-reaaliaikakäyttöjärjestelmää irrottavassa tilassa. Irrottava vuoronnusmenetelmä takaa nopeat vasteajat lennonohjausjärjestelmän taskeille. Esimerkkinä voidaan käyttää anturidatan lukemista. Inertiamittausyksikkö MPU-6050 ilmoittaa anturidatan olevan puskurissa valmiina syöttämällä lyhyen jännitepulssin yhteen lähtönastoistaan. Datan vastaanotto synkronoidaan reaaliaikakäyttöjärjestelmään AVR-mikrokontrollerin tulonastan tilamuutoksen havaitsevan keskeytyksen avulla. Mahdollisesti ajossa oleva alemman prioriteetin taski irrotetaan suorituksesta ja anturidataa käsittelevä taski pääsee ajoon. Jos sama tehtäisiin ei-irrottavassa tilassa, jouduttaisiin synkronoinnin tapahtumista odottamaan siihen asti, kunnes alemman prioriteetin taski antaa suorittimen käyttöön.

4.5 Sanomanvälitysmekanismit ja synkronointi

Taskien välistä viestintää varten on olemassa erilaisia sanomanvälitysmekanismeja. Näillä taski voi lähettää ja vastaanottaa sanomia yhdeltä tai useammalta taskilta, sekä myös keskeytyksiltä. Yksi yksinkertaisimmista sanomanvälitysmekanismeista on postilaatikkomekanismi. Käyttöjärjestelmä tarjoaa järjestelmäkutsut postilaatikon luomiseen, sinne lähettämiseen ja sieltä vastaanottamiseen. Postilaatikkomekanismi ei ole siis sidottu mihinkään yksittäiseen taskiin, vaan sinne voi kuka tahansa lähettää ja sieltä voi kuka tahansa lukea. (Haikala 2003, 112.)

Sanomajonon voidaan ajatella olevan useamman postilaatikon muodostama taulukko. Samalla tavalla kuin postilaatikkomekanismin tapauksessa, voi mikä tahansa taski tai keskeytys lähettää sanomia mihin tahansa laatikkoon. Yleensä sanomajonot toimivat FIFO-periaatteella (First In, First Out). Tämä tarkoittaa, että ensimmäinen jonoon lähetetty sanoma on myös ensimmäinen, mikä voidaan sieltä vastaanottaa. Taski voi jäädä järjestelmäkutsulla odottamaan sanoman vastaanottamista jonosta. Yleensä tämä ytimen

tarjoama järjestelmäkutsu sisältää myös valinnaisen timeout-toiminnon, jolla taski saadaan heräämään, jos viestiä ei ole tullut tietyssä ajassa. (Labrosse 2002, 61.)

Sanomanvälitystä voi käyttää taskien välisen tiedonsiirron lisäksi myös taskien synkronointiin eli tahdistamiseen. Lähetyksellä voidaan taski synkronoida tapahtumaan. Tapahtuma voi olla esimerkiksi keskeytyspalvelu. Viestiä odottamaan jäänyt taski saadaan heräämään, jos taskin prioriteetti on tarpeeksi korkea ja keskeytyspalvelu lähettää järjestelmäkutsulla sille sanoman. Viestin lähettämisen jälkeen pakotettu ympäristön vaihto saa taskin heräämään. Muita ytimessä synkronointiin käytettävissä olevia mekanismeja ovat mm. semaforit (Labrosse 2002, 57). Semaforien muita käyttötarkoituksia reaaliaikajärjestelmissä ovat mm. poissulkemiset, jolla estetään pääsy tiettyyn koodin osaan (Haikala 2003, 112). Binäärisellä semaforilla tehdyllä synkronoinnilla pystyy siirtämään vain tiedon yksittäisestä tapahtumasta mutta ei ylimääräistä informaatiota sen ohessa.

Lennonohjausjärjestelmässä anturidataa käsittelevä taski käyttää sanomajonoa synkronointimenetelmänä. Tällä tavalla datan vastaanottamisen yhteydessä taskeille voidaan välittää tieto siitä, mistä data on peräisin. Kun useita eri antureita on kytkettynä samaan väylään ja näiden dataa luetaan samaan vastaanottopuskuriin, eivät taskit ole tietoisia siitä mitä puskurissa on ja missä järjestyksessä. Sanomajono ratkaisee tämän ongelman säilyttämällä kunkin anturin yksilöivän merkin jonossa oikeassa järjestyksessä, aina siihen asti kunnes dataa käsittelevä taski tyhjentää puskurista kunkin anturin mittaustulokset.

Esimerkki FreeRTOS-taskin koodista ja synkronoinnista nähdään kuvassa 16. Tämä on anturidataa lukevan taskin lähdekoodi, josta on poistettu synkronoinnin jälkeiset tapahtumat. FreeRTOS-taskit ohjelmoidaan aliohjelmiksi, jotka eivät tuota paluuarvoa. Taskille voidaan asettaa sen prosessielementtiin dataa, jolla yksilöidään se muiden taskien joukossa. Tämä voidaan tehdä taskin alustustoimenpiteenä, kuten kuvassa 16. Synkronointiin tarvittava sanomajono alustetaan tämän jälkeen järjestelmäkutsulla xQueueCreate. Taskin ohjelmasilmukassa odotetaan sanoman vastaanottamista. Jos ohjelman suoritus pääsisi etenemään silmukan ulkopuolelle, on tätä tilannetta varten järjestelmäkutsu vTaskDelete asetettu silmukan jälkeiseen koodiin. Ilman tätä aliohjelmakutsua olisi järjestelmän käyttäytyminen ennalta arvaamatonta, kun suoritus etenee silmukan ulkopuolelle ja poistuu taskista.

```
void vIMUinputTask(void *pvParameters)
{
    // Asetetaan prosessielementtiin taskin yksilöivä merkki
    vTaskSetApplicationTaskTag(NULL,(void *) TASK_TAG_IMUINPUT);

    // Alustetaan sanomajono, muuttuja on esitelty QueueHandle_t tyyppisenä
    xIMUsyncQueue = xQueueCreate(10, sizeof(uint8_t));

    // Tähän muuttujaan luetaan synkronoinnin yhteydessä jonon ensimmäinen tavu
    uint8_t message;

    // Taskin silmukka
    while(1){

        // Synkronointi sanomajonon vastaanottamiseen
        if (xQueueReceive(xIMUsyncQueue,&message,portMAX_DELAY)){

            // Tehdään täällä synkronoinnin jälkeiset toimenpiteet
        }
    }

    // Jos taski poistuu silmukasta, se tuhoaa itsensä
    vTaskDelete(NULL);
}
```

KUVA 16. Esimerkki FreeRTOS-taskista ja synkronoinnista

4.6 Avoimen lähdekoodin RTOS-ytimet

FreeRTOS on avoimen lähdekoodin reaaliaikakäyttöjärjestelmä, jonka lähdekoodi on lisensoitu muokatulla GPL-lisenssillä. Lisenssissä oleva poikkeus mahdollistaa omien sovellusohjelmien kehityksen ilman että näiden lähdekoodi tulee GPL-lisenssin piiriin. Tämä mahdollistaa sen käytön suljetun lähdekoodin omaavassa projektissa ja suljetun lähdekoodin omaavan tuotteen kaupallisen jakelun. GPL-lisenssiin lisätty poikkeus kuitenkin edellyttää, että ytimeen linkitettyjen sovellusohjelmien koodi ei muokkaa tai laajenna RTOS-toiminnallisuutta, tai että lopputuote ei ole itsessään uusi laajennettu reaaliaikakäyttöjärjestelmä. Osa FreeRTOS-reaaliaikakäyttöjärjestelmän lisenssiehdoista on nähtävissä työn lopussa olevan liitteen 1 alussa, joka on GPL-lisenssin ehtojen piiriin kuuluva tiedosto port.c. Useita reaaliaikakäyttöjärjestelmiä on saatavilla hyvin samankaltaisella muokatulla GPL-lisenssillä. Näitä ovat mm. ChibiOS/RT, BeRTOS ja DuinOS.

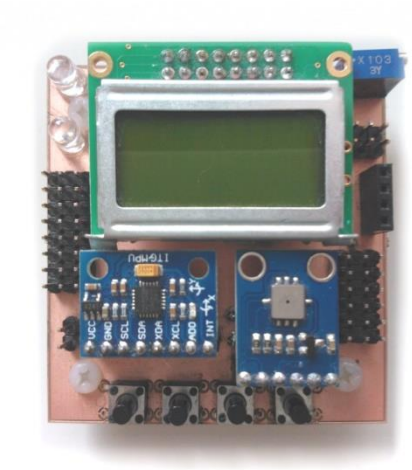
Enemmän vapauksia ytimen muokkaamista varten antaa esimerkiksi BSD-ohjelmistolisenssi. BSD-ohjelmistolisenssin ehtojen piiriin kuuluvan lähdekoodin muutoksia ei tarvitse julkaista vaikka sitä käyttävät tuotteet olisivat kaupallisessa tai muuten julkisessa jakelussa. Lisenssiteksti täytyy kuitenkin esittää tuotteen jakelun yhteydessä. BSD-ohjelmistolisenssin tapauksessa voi siis koko projektin lähdekoodi olla suljettu muokkauksista riippumatta. BSD-ohjelmistolisenssiä käyttäviä reaaliaikaytimiä ovat mm. Atomthreads, cocoOS, Nut/OS ja NuttX.

FreeRTOS valikoitui monen vaihtoehdon joukosta opinnäytetyössä käytettäväksi reaaliaikakäyttöjärjestelmäksi. Eniten käyttöjärjestelmän valintaan vaikutti internetistä löytyvän ohjeistuksen ja käyttäjäkokemuksien määrä, sekä tuki eri laitteistoarkkitehtuureille. FreeRTOS sisältää kaikki tarvittavat toiminnot ja järjestelmäkutsut joita lennonohjaustietokoneen laiteohjelmistossa tarvitaan. Tämän vuoksi sen toiminnallisuutta ei tarvitse laajentaa. Ainoastaan tiedostoon port.c tarvitsi tehdä muutokset joilla käyttöjärjestelmä saatiin sovitettua käytetylle mikrokontrollerille. Tuki sekä irrottavalle että ei-irrottavalle vuoronnusmenetelmälle katsottiin myös eduksi.

5 JÄRJESTELMÄN KEHITYSTYÖ

5.1 Laitteisto

Multikopterin lennonohjaustietokone on tavallisesti tehty kooltaan 50 x 50 mm:n piirilevylle, jonka kulmissa ovat kiinnitysreiät. Kiinnitysreikien keskipisteiden väliset mitat ovat 45 x 45 millimetriä. Levyn mitat ovat vakiintuneet de facto –standardiksi, jota eri valmistajat ja harrastajat käyttävät, kun suunnittelevat multikopteriin sen asennuspaikan. Lennonohjaustietokone sijoitetaan lähelle multikopterin massakeskipistettä. Piirilevyssä on tuloliitännät ohjaussignaaleille ja lähdöt moottorien nopeudensäätimille. Myös liitännät monen tyyppisille ulkoisille oheislaitteille ovat mahdollisia, esimerkiksi GPS-moduulille.



KUVA 17. Lennonohjaustietokone

Näitä periaatteita noudattaen suunniteltu lennonohjaustietokone on esitetty kuvassa 17. Levy on suunniteltu Altium Designer -ohjelmistolla. Levyn piirikaavio on liitteessä 2. Ohjaussignaaliliitäntöjen lisäksi siinä on mahdollisuus ulkoisen radiomoduulin ja GPS-moduulin käyttöön. Painikkeet ja näyttö mahdollistavat mm. säädinparametrien muokkaamisen suoraan piirilevyltä itseltään. Levyssä inertiamittausyksikkönä on MPU-6050 ja barometrina Bosch BMP085. MPU-6050 sisältää kolmen akselin gyroskoopit, joilla kulmanopeuden mittausalueeksi voidaan asettaa minimissään ± 250°/s ja maksimissaan ± 2000°/s. Se sisältää myös kolmen akselin kiihtyvyysanturit, joilla kiihtyvyyden mittausalueeksi voidaan asettaa minimissään ± 2g ja maksimissaan ± 16g. Mittausalueiden resoluutio on 16-bittiä. (Invensense 2015, 7)

Barometrilla BMP085 voidaan mitata ilmanpaine väliltä 300 - 1100 hPa, ja sen resoluutio on enintään 19-bittiä. Nämä vastaavat +9000 metrin ja -500 metrin korkeuksia merenpinnan tason ilmanpaineeseen verrattuna. (Bosch Sensortec 2015, 2) Anturit on liitetty lennonohjaustietokoneeseen breakout-piirilevyillä GY-521 ja GY-65. Tämä mahdollistaa anturitekniikan vaihtamisen helposti uudempaan, edellyttäen että uusi osa käyttää samoja mekaanisia ja sähköisiä liitäntöjä.

5.2 Laitteiston ja RTOS-ytimen yhteensovitus

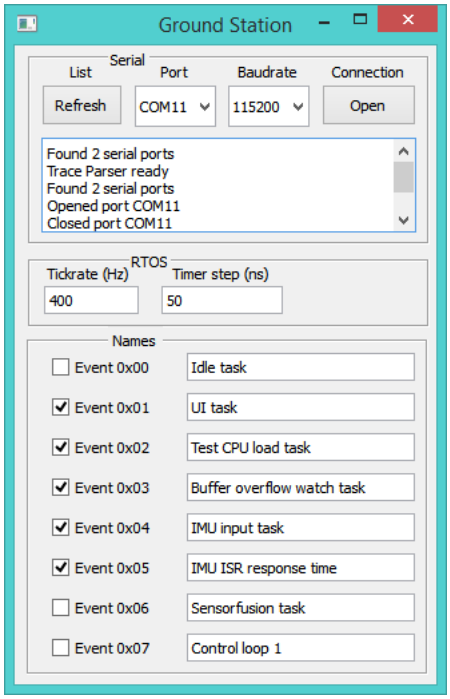
FreeRTOS-reaaliaikakäyttöjärjestelmän versio 8.0.1 oli lennonohjaustietokoneen ohjelmiston pohjana. FreeRTOS-reaaliaikakäyttöjärjestelmän kyseinen versio ladattiin kehittäjien verkkosivuilta (Real Time Engineers Ltd. 2015). FreeRTOS-jakelupaketissa tuli 133:lle eri alustalle tarkoitetut versiot, tässä mukaan luettuna eri kehitysympäristöille sovitetut versiot. FreeRTOS-reaaliaikakäyttöjärjestelmän ydin on tarkoitettu käännettäväksi lähdekoodista samassa yhteydessä sovellusohjelmien kanssa. Sovellusohjelmat eli taskit toteutetaan aliohjelmina, ja ne käynnistetään ytimen xTaskCreate-järjestelmäkutsulla. Tämän vuoksi oli tärkeätä saada ensin käyttöjärjestelmän ydin kääntymään ja toimimaan, jotta ohjelmiston muiden osien kehitystyö voisi jatkua.

FreeRTOS 8.0.1 ei sisällä lennonohjaustietokoneessa käytetylle AT-mega324-mikrokontrollerille valmista versiota. Jakelun mukana tuleva 8-bitin AVR-arkkitehtuurille tarkoitettu versio on tehty ATmega323-mikrokontrollerille. ATmega323:n oheislaitteita hyödyntävä koodi tarvitsee muutoksia toimiakseen ATmega324:n kanssa. Merkittävin muutos koski käyttöjärjestelmän toiminnalle tärkeän ajastinkeskeytyksen toimintaan saattamista. Koska ajastimien konfigurointiin tarkoitetut rekisterit ovat erilaiset kuin ATmega324 mikrokontrollerissa, täytyi rekisterien käyttöön liittyvät komennot ensin muuttaa sopiviksi.

Muita koodiin tehtyjä muutoksia ovat mm. ajastinkeskeytyspalvelun käyttämien SIGNAL-lohkojen vaihtaminen nykyisissä kehitystyökaluissa enemmän käytetyiksi ISR-lohkoiksi. Kaikki alustakohtainen koodi on tiedostoissa port.c ja portmacro.h. ATmega324-mikrokontrollerille sovitettu versio FreeRTOS 8.0.1-reaaliaikakäyttöjärjestelmän tiedostosta port.c on liitteessä 1.

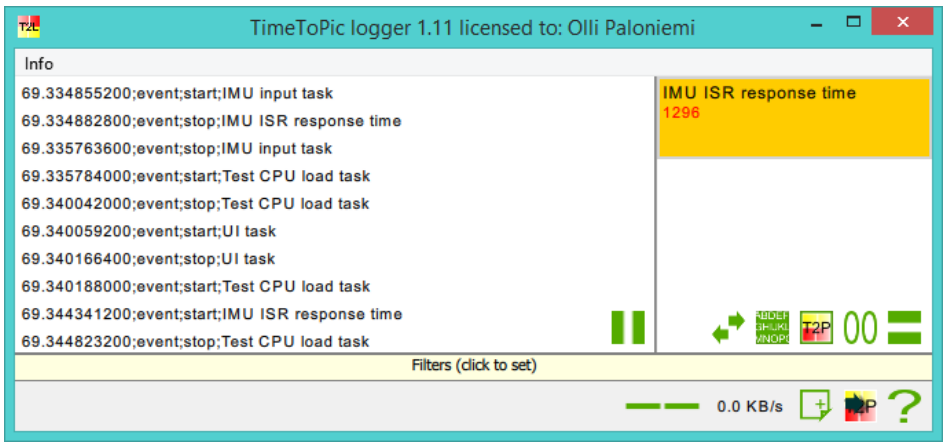
5.3 Suoritusaikojen mittaus ja visualisointi

Lennonohjausjärjestelmän reaaliaikaisten osien toiminnan varmistaminen on tärkeä osa ohjelmiston kehitystyötä. Yksittäisten tapahtumien suoritusaikojen seuranta onnistuu toki esimerkiksi oskilloskoopilla mittaamalla johonkin lähtönastaan ohjelmoitua tilamuutosta, mutta reaaliaikakäyttöjärjestelmän lukuisien taskien ja laitteiston keskeytyspalvelujen vuorovaikutuksen seuranta tällä menetelmällä on hyvin työlästä. Tarpeellisia mitattavia asioita ovat mm. taskien suoritusajat ja keskeytyspalveluista syntyvien synkronointien vasteajat. Yksi näistä synkronoiduista taskeista, jonka vaste- ja suoritusaika olivat tarpeen mitata, on kuvan 16 anturidataa lukeva taski.



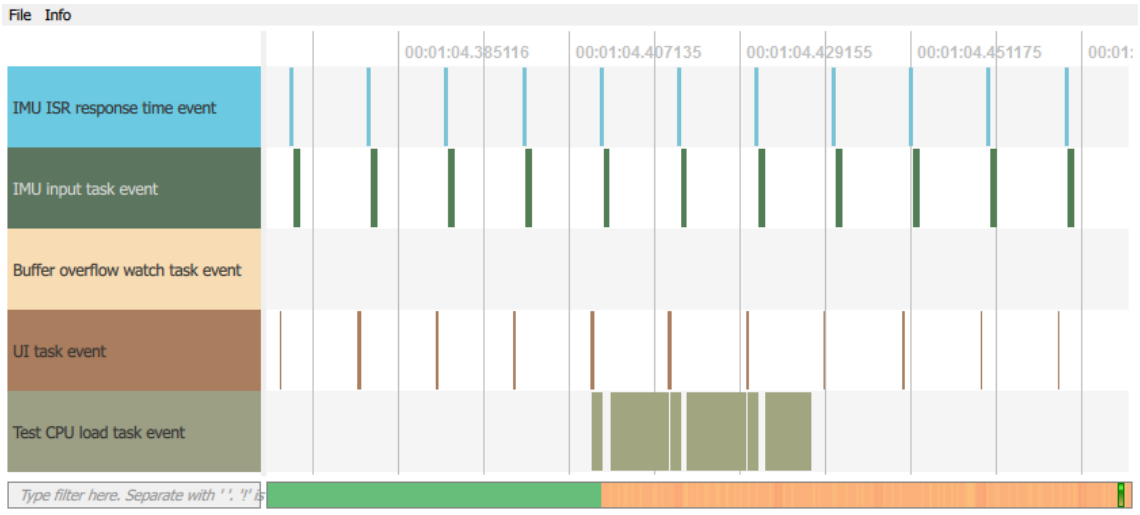
KUVA 18. Maa-aseman asetukset sarjaportille ja telemetrialle

Ratkaisuna mittausjärjestelmän tarpeelle oli käyttää radiomoduulia välittämään mittausdataa lennonohjaustietokoneesta PC:lle. Tarkoitusta varten tehty maa-asemaohjelmisto (kuva 18) vastaanottaa radiolinkillä välitetyn datan sarjaportista. Data on lähtöisin reaaliaikakäyttöjärjestelmän trace-makroista ja omista lentotelemetrian lähetykseen tarkoitetuista ohjelmamakroista. Makroissa ohjelmiston tiedonsiirtopuskuriin työnnetään aikaleimattua dataa, joka kehystetään ja lopulta siirretään tiedonsiirtopuskurista laitteiston UART-lähetyspuskuriin tavu kerrallaan. Tärkeä osa ohjelmistoa on myös mekanismi jolla lähetetyn datan määrää voidaan rajoittaa. Näin tiedonsiirto voidaan pitää radiolinkin kapasiteetin rajoissa.



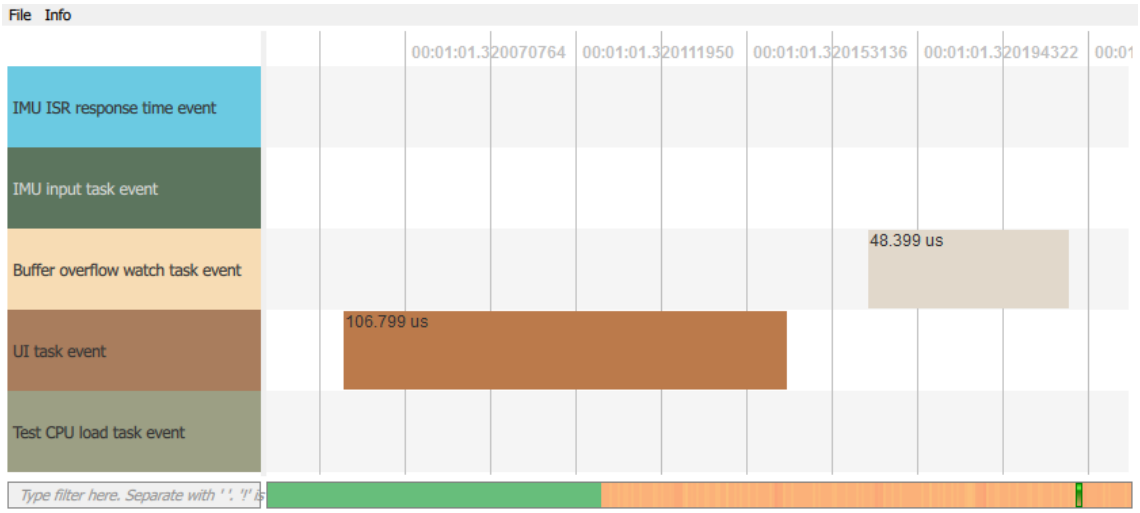
KUVA 19. TimeToPicLogger-ohjelman vastaanottamaa dataa

Trace-data visualisoitiin Erkki Salosen ja Herwood Technologies Oy:n kehittämällä TimeToPic-ohjelmistolla. TimeToPic (kuva 20) ja sen ohessa tuleva logger-ohjelma (kuva 19) on pääasiassa tarkoitettu datan keräämiseen testattavasta ohjelmasta niin, että niiden toiminnallisuus on upotettu koodin sekaan ohjelmamakroilla, jotka käyttävät TCP-socket-yhteyttä viestimiseen. Nämä makrot hoitavat viestinnän logger-ohjelmaan sen tuntemassa lokiformaatissa. Tämä järjestely toimii hyvin, jos testattava ohjelma on esimerkiksi Qt-kirjastoja käyttävä PC-ohjemisto, mutta ei multikopterin lennonohjaus-tietokoneen laiteohjelmiston tapauksessa. Lennonohjaustietokoneen mikrokontrollerissa ei ole tarvittavaa laskentatehoa, muistia, tiedonsiirtokapasiteettia saati edes TCP/IP-pinoa, joilla viestintä logger-ohjelmistoon mahdollistuisi. Multikopterin tapauksessa tarvittiin väliin siksi maa-asemaohjelmisto hoitamaan TCP-lähetys logger-ohjelmaan.



KUVA 20. Vastaanotetun datan visualisointi TimeToPic-ohjelmassa

Kun trace-data vastaanotettiin logger-ohjelmaan, voitiin kirjoitettu lokitiedosto visualisoida TimeToPic-ohjelmalla. Visualisointi on mahdollista tehdä logger-ohjelmasta napin painalluksella, jolloin siihen mennessä kirjoitettu loki siirtyy TimeToPic-ohjelman näkymään. Tämä ratkaisu ei mahdollista täysin reaaliaikaisesti päivittyvää näkymää taskien suoritusajoista tai lentotelemetriasta, mutta sille ei ollut vielä tarvetta tätä ohjelmistoa tehdessä. Kuvassa 21 näkyy kahden taskin suoritusajat. Nämä ovat käyttöliittymän taski UI task event ja puskureiden ylivuodoista ilmoittava taski Buffer overflow watch task event. TimeToPic-ohjelman tekemästä visualisoinnista on myös mahdollista mitata ympäristön vaihtoon kuluva aika käyttämällä ohjelman mittatyökalua taskien välisen ajan mittaamiseen.

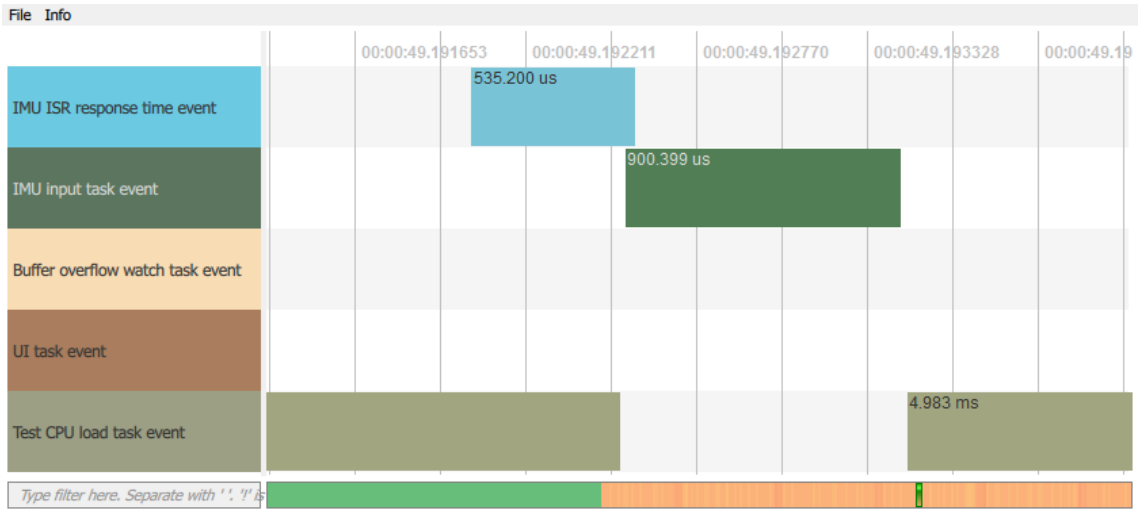


KUVA 21. Taskin suoritusajan mittaus

5.4 RTOS-taskien synkronointi

Toinen oleellinen mitattava asia oli käyttöjärjestelmän reagointi ulkoisiin herätteisiin. Esimerkki synkronoidusta taskista nähtiin kuvassa 16. Kyseisen anturidataa käsittelevän IMU input task event -taskin suoritus näkyy kuvassa 22 vihreänä. Tämän taskin heräämiseen kuluvaa aikaa mittaa kuvassa sinisenä esitetty IMU ISR response time event -tapahtuma. Inertiamittausyksikkö MPU-6050 antaa jännitepulssin, joka näkyy yhdessä mikrokontrollerin tulonastoista. Tästä syntyneen keskeytyspalvelun päätyttyä alkaa datan lukeminen I²C-väylästä, ja viimeisen tavun vastaanoton jälkeen tapahtuu synkronointi viestinvälityksellä käyttöjärjestelmän taskiin. Kun vihreällä merkitty taski herää, lopetetaan kuvassa sinisenä näkyvän tapahtuman mittaus. Lopputuloksena IMU ISR

response time event -tapahtuman suoritusajasta saadaan vasteaika anturidatan lukemisesta sen käsittelyn alkamiseen. IMU input task event –taskin pituus kertoo anturidatan käsittelyyn kuluvan suoritusajan.



KUVA 22. Vasteajan mittaaminen

Kun vaste- ja suoritusajat on mahdollista mitata, on lennonohjausjärjestelmän aikakriittisten osien suunnittelu huomattavasti helpompaa. Lennonohjausjärjestelmän ohjelmiston eri osien toimintataajuudet ja näiden välinen synkronointi on mahdollista optimoida niin että saavutetaan paras mahdollinen suorituskyky. Lisäksi korkean prioriteetin lennonohjaustaskeilta jäljelle jäävä suoritinaika on mahdollista hahmottaa ja hyödyntää. Kaikki vapaana oleva suoritinaika, jos keskeytyksien viemää aikaa ei oteta huomioon, voitaisiin esittää visualisoimalla järjestelmän tyhjäkäyntitaskin suoritusajat.

6 POHDINTA

Tämän opinnäytetyön tarkoituksena oli tutkia reaaliaikakäyttöjärjestelmän soveltamista osana multikopterin lennonohjausjärjestelmää. Työssä kehitettiin sekä laitealusta että reaaliaikakäyttöjärjestelmää käyttävä laiteohjelmisto. Opinnäytetyön aihe oli itsenäisesti valittu, työssä kuvattua järjestelmää ei ole tehty minkään yrityksen käyttöön. Työllä ei siksi ollut kenenkään ulkopuolisen asettamia vaatimuksia, vaan sen oli tarkoitus täyttää lähinnä tekijänsä itse asettamat henkilökohtaiset tavoitteet.

Ennen varsinaisen opinnäytetyöaiheen aloittamista oli tehtynä jo paljon alustavaa työtä, joka loi edellytykset työn mahdollistumiseen. Lennonohjausjärjestelmän laitealustana toimiva lennonohjaustietokone suunniteltiin Tampereen Ammattikorkeakoulussa keväällä 2014 osana piirilevysuunnittelukurssin projektityötä. Suunnittelun tein yhteistyössä Simo Sihvosen kanssa. Lennonohjaustietokoneen prototyypin piirilevy valmistettiin piirilevyjyrsimellä ja komponentit juotettiin käsin. Laiteohjelmiston suunnittelun aloitin levyn valmistuttua sen tulo- ja lähtöliitäntöjen PWM-ajureiden ohjelmoinnilla ja tämän jälkeen reaaliaikaytimen sovittamisella laitteistoon.

Laiteohjelmiston suoritusaikojen seurannan osana hyödyntämääni TimeToPic-ohjelmistoon sain lisenssin Herwood Technologies Oy:ltä. Kolmannen osapuolen tarjoaman dataloggerin ja visualisaattorin käyttäminen projektissa säästi paljon aikaa ja vaivaa. Ilman toimivaa visualisointia olisi työskentely ohjelmiston parissa ollut ns. sokeaa, eli tietoa siitä tapahtuvatko ohjelmiston sisäiset asiat oletetulla tavalla ei olisi ollut. Lennonohjaustietokoneen ulkoisten liitäntöjen toimivuus testattiin oskilloskoopilla ja logiikka-analysaattorilla. Varsinainen lennonohjausjärjestelmän toiminnan testaaminen tehtiin asentamalla lennonohjaustietokone multikopteriin.

Opinnäytetyö saavutti ja ylitti sille itse työn alussa asettamani tavoitteet, jotka olivat toimivan anturidataa, säätötekniikkaa ja reaaliaikakäyttöjärjestelmää käyttävän laiteohjelmiston tekeminen. Tämän työn aiheen puitteissa tämä kokonaisuus tehtiin multikopterin lennonohjaustietokoneeseen. Tehty laiteohjelmisto on reaaliaikaytimen toiminnallisuutta hyvin hyödyntävä. Laitealustassa on kuitenkin vielä paljon potentiaalia, jota ohjelmisto ei täysin hyödynnä. Siksi projektille onkin lukuisia jatkokehitysmahdollisuuksia, joista yksi kehityssuunta ovat autonomiset toiminnot.

Reaaliaikakäyttöjärjestelmä ohjelmointiympäristönä soveltuu erittäin hyvin UAV-järjestelmän käyttöön, joten jatkokehitys tähän suuntaan on jo hyvin valmisteltu. Kun laitealustan vaihtaminen on tarpeellista, tulee reaaliaikakäyttöjärjestelmä osaltaan helpottamaan lähdekoodin siirtoa toiseen laitearkkitehtuuriin. Tämän opinnäytetyön tekemisen ohessa syntyneet ideat ovat sovellettavissa multikopterien lisäksi myös moneen muuhun laitteeseen. Tästä yhtenä esimerkkinä voi mainita kameragimbaalit, jotka multikopterin ohjaustietokoneen tavoin hyödyntävät inertiamittausyksiköitä ja säätötekniikkaa.

LÄHTEET

Castillo, P., Lozano, R. & Dzul, A., E. 2005. Modelling and Control of Mini-Flying Machines. London: Springer.

Killian, C. 2006. Modern control technology: components and systems. New York: Delmar.

Higgins, W. 1975. A Comparison of Complementary and Kalman Filtering. IEEE Trans. Aerospace and Electronic Systems, Vol. AES-li, no. 3, May 1975. Luettu 13.4.2015.
http://goo.gl/CLX6G8

Colton, S. 2007. The Balance Filter. Luettu 29.3.2015.
http://goo.gl/TToqG8

ArduPilot open-source flight controller. 2015. AHRS. DIY Drones. Luettu 26.3.2015.
https://github.com/diydrones/ardupilot/tree/master/libraries/AP_AHRS

MultiWii open-source flight controller. 2014. IMU.cpp. Luettu 26.3.2015.
https://code.google.com/p/multiwii/source/browse/trunk/MultiWii_shared/IMU.cpp

Labrosse, J. 2002. µC/OS, The Real-Time Kernel – 2nd Edition. San Francisco: CMP Books.

Guowei, C., Ben M., C. & Tong, H., L. 2011. Unmanned Rotorcraft Systems. London: Springer.

Laplante, P. & Ovaska, S. 2012. Real-time systems design and analysis: tools for the practitioner. New Jersey: John Wiley & Sons, Inc.

Haikala, I. & Järvinen, H. 2003. Käyttöjärjestelmät. Helsinki: Talentum Media Oy.

Silberschatz, A., Galvin, P. & Gagne, G. 2005. Operating System Concepts – 7th Edition. New Jersey: John Wiley & Sons, Inc.

MPU-6050 Product Specification. 2013. Invensense. Luettu 11.4.2015.
http://www.invensense.com/mems/gyro/documents/PS-MPU-6000A-00v3.4.pdf

BMP085 Datasheet. 2008. Bosch Sensortec. Luettu 11.4.2015.
http://www.spezial.cz/pdf/BMP085_Flyer_Rev.0.2_March2008.pdf

FreeRTOS dokumentaatio. 2015. Real Time Engineers Ltd. Luettu 17.3.2015.
http://www.freertos.org/RTOS.html

TimeToPic. 2015. Herwood Technologies Oy. Luettu 19.3.2015.
http://timetopic.net/

LIITTEET

Liite 1. port.c 1 (8)

```
/*
    FreeRTOS V8.0.1 - Copyright (C) 2014 Real Time Engineers Ltd.
    All rights reserved

    VISIT http://www.FreeRTOS.org TO ENSURE YOU ARE USING THE LATEST VERSION.

    ***************************************************************************
     *                                                                         *
     *    FreeRTOS provides completely free yet professionally developed,      *
     *    robust, strictly quality controlled, supported, and cross            *
     *    platform software that has become a de facto standard.               *
     *                                                                         *
     *    Help yourself get started quickly and support the FreeRTOS           *
     *    project by purchasing a FreeRTOS tutorial book, reference            *
     *    manual, or both from: http://www.FreeRTOS.org/Documentation          *
     *                                                                         *
     *    Thank you!                                                           *
     *                                                                         *
    ***************************************************************************

    This file is part of the FreeRTOS distribution.

    FreeRTOS is free software; you can redistribute it and/or modify it under
    the terms of the GNU General Public License (version 2) as published by the
    Free Software Foundation >>!AND MODIFIED BY!<< the FreeRTOS exception.

    >>!   NOTE: The modification to the GPL is included to allow you to     !<<
    >>!   distribute a combined work that includes FreeRTOS without being   !<<
    >>!   obliged to provide the source code for proprietary components     !<<
    >>!   outside of the FreeRTOS kernel.                                   !<<

    FreeRTOS is distributed in the hope that it will be useful, but WITHOUT ANY
    WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
    FOR A PARTICULAR PURPOSE.  Full license text is available from the following
    link: http://www.freertos.org/a00114.html

    1 tab == 4 spaces!

    ***************************************************************************
     *                                                                         *
     *    Having a problem?  Start by reading the FAQ "My application does     *
     *    not run, what could be wrong?"                                       *
     *                                                                         *
     *    http://www.FreeRTOS.org/FAQHelp.html                                 *
     *                                                                         *
    ***************************************************************************

    http://www.FreeRTOS.org - Documentation, books, training, latest versions,
    license and Real Time Engineers Ltd. contact details.

    http://www.FreeRTOS.org/plus - A selection of FreeRTOS ecosystem products,
    including FreeRTOS+Trace - an indispensable productivity tool, a DOS
    compatible FAT file system, and our tiny thread aware UDP/IP stack.

    http://www.OpenRTOS.com - Real Time Engineers ltd license FreeRTOS to High
    Integrity Systems to sell under the OpenRTOS brand.  Low cost OpenRTOS
    licenses offer ticketed support, indemnification and middleware.
```

```
    http://www.SafeRTOS.com - High Integrity Systems also provide a safety
    engineered and independently SIL3 certified version for use in safety and
    mission critical applications that require provable dependability.

    1 tab == 4 spaces!
*/

/*

Changes from V2.6.0

            + AVR port - Replaced the inb() and outb() functions with direct
            memory access. This allows the port to be built with the 20050414
            build of WinAVR.
*/

#include <stdlib.h>
#include <avr/interrupt.h>

#include "FreeRTOS.h"
#include "task.h"

/*-----------------------------------------------------------
 * Implementation of functions defined in portable.h for the AVR port.
 *----------------------------------------------------------*/

/* Start tasks with interrupts enables. */
#define portFLAGS_INT_ENABLED                    ( ( StackType_t ) 0x80 )

/* Hardware constants for timer 1. */
#define portCLEAR_COUNTER_ON_MATCH               ( ( uint8_t ) 0x08 )
#define portPRESCALE_8                           ( ( uint8_t ) 0x02 )
#define portCLOCK_PRESCALER                      ( ( uint32_t ) 8 )
#define portCOMPARE_MATCH_A_INTERRUPT_ENABLE     ( ( uint8_t ) 0x02 )

/*-----------------------------------------------------------*/

/* We require the address of the pxCurrentTCB variable, but don't want to know
any details of its type. */
typedef void TCB_t;
extern volatile TCB_t * volatile pxCurrentTCB;

/*-----------------------------------------------------------*/

/*
 * Macro to save all the general purpose registers, the save the stack pointer
 * into the TCB.
 *
 * The first thing we do is save the flags then disable interrupts.  This is to
 * guard our stack against having a context switch interrupt after we have
 * already pushed the registers onto the stack - causing the 32 registers to be
 * on the stack twice.
 *
 * r1 is set to zero as the compiler expects it to be thus, however some
 * of the math routines make use of R1.
 *
 * The interrupts will have been disabled during the call to portSAVE_CONTEXT()
 * so we need not worry about reading/writing to the stack pointer.
 */

#define portSAVE_CONTEXT()                                                  \
            asm volatile ( "push    r0                      \n\t"       \
                           "in      r0, __SREG__            \n\t"       \
```

```
                        "cli                            \n\t"       \
                        "push   r0                      \n\t"       \
                        "push   r1                      \n\t"       \
                        "clr    r1                      \n\t"       \
                        "push   r2                      \n\t"       \
                        "push   r3                      \n\t"       \
                        "push   r4                      \n\t"       \
                        "push   r5                      \n\t"       \
                        "push   r6                      \n\t"       \
                        "push   r7                      \n\t"       \
                        "push   r8                      \n\t"       \
                        "push   r9                      \n\t"       \
                        "push   r10                     \n\t"       \
                        "push   r11                     \n\t"       \
                        "push   r12                     \n\t"       \
                        "push   r13                     \n\t"       \
                        "push   r14                     \n\t"       \
                        "push   r15                     \n\t"       \
                        "push   r16                     \n\t"       \
                        "push   r17                     \n\t"       \
                        "push   r18                     \n\t"       \
                        "push   r19                     \n\t"       \
                        "push   r20                     \n\t"       \
                        "push   r21                     \n\t"       \
                        "push   r22                     \n\t"       \
                        "push   r23                     \n\t"       \
                        "push   r24                     \n\t"       \
                        "push   r25                     \n\t"       \
                        "push   r26                     \n\t"       \
                        "push   r27                     \n\t"       \
                        "push   r28                     \n\t"       \
                        "push   r29                     \n\t"       \
                        "push   r30                     \n\t"       \
                        "push   r31                     \n\t"       \
                        "lds    r26, pxCurrentTCB       \n\t"       \
                        "lds    r27, pxCurrentTCB + 1   \n\t"       \
                        "in     r0, 0x3d                \n\t"       \
                        "st     x+, r0                  \n\t"       \
                        "in     r0, 0x3e                \n\t"       \
                        "st     x+, r0                  \n\t"       \
                    );

/*
 * Opposite to portSAVE_CONTEXT().  Interrupts will have been disabled during
 * the context save so we can write to the stack pointer.
 */

#define portRESTORE_CONTEXT()                                       \
            asm volatile ( "lds     r26, pxCurrentTCB       \n\t"       \
                        "lds    r27, pxCurrentTCB + 1   \n\t"       \
                        "ld     r28, x+                 \n\t"       \
                        "out    __SP_L__, r28           \n\t"       \
                        "ld     r29, x+                 \n\t"       \
                        "out    __SP_H__, r29           \n\t"       \
                        "pop    r31                     \n\t"       \
                        "pop    r30                     \n\t"       \
                        "pop    r29                     \n\t"       \
                        "pop    r28                     \n\t"       \
                        "pop    r27                     \n\t"       \
                        "pop    r26                     \n\t"       \
                        "pop    r25                     \n\t"       \
                        "pop    r24                     \n\t"       \
                        "pop    r23                     \n\t"       \
```

```
                "pop     r22                     \n\t"       \
                "pop     r21                     \n\t"       \
                "pop     r20                     \n\t"       \
                "pop     r19                     \n\t"       \
                "pop     r18                     \n\t"       \
                "pop     r17                     \n\t"       \
                "pop     r16                     \n\t"       \
                "pop     r15                     \n\t"       \
                "pop     r14                     \n\t"       \
                "pop     r13                     \n\t"       \
                "pop     r12                     \n\t"       \
                "pop     r11                     \n\t"       \
                "pop     r10                     \n\t"       \
                "pop     r9                      \n\t"       \
                "pop     r8                      \n\t"       \
                "pop     r7                      \n\t"       \
                "pop     r6                      \n\t"       \
                "pop     r5                      \n\t"       \
                "pop     r4                      \n\t"       \
                "pop     r3                      \n\t"       \
                "pop     r2                      \n\t"       \
                "pop     r1                      \n\t"       \
                "pop     r0                      \n\t"       \
                "out     __SREG__, r0            \n\t"       \
                "pop     r0                      \n\t"       \
        );

/*-----------------------------------------------------------*/

/*
 * Perform hardware setup to enable ticks from timer 1, compare match A.
 */
static void prvSetupTimerInterrupt( void );
/*-----------------------------------------------------------*/

/*
 * See header file for description.
 */
StackType_t *pxPortInitialiseStack( StackType_t *pxTopOfStack, TaskFunction_t
pxCode, void *pvParameters )
{
uint16_t usAddress;

            /* Place a few bytes of known values on the bottom of the stack.
            This is just useful for debugging. */

            *pxTopOfStack = 0x11;
            pxTopOfStack--;
            *pxTopOfStack = 0x22;
            pxTopOfStack--;
            *pxTopOfStack = 0x33;
            pxTopOfStack--;

            /* Simulate how the stack would look after a call to vPortYield()
            generated by the compiler. */

            /*lint -e950 -e611 -e923 Lint doesn't like this much - but nothing I
            can do about it. */

            /* The start of the task code will be popped off the stack last, so
            place it on first. */
            usAddress = ( uint16_t ) pxCode;
            *pxTopOfStack = ( StackType_t ) ( usAddress & ( uint16_t ) 0x00ff );
```

```
pxTopOfStack--;

usAddress >>= 8;
*pxTopOfStack = ( StackType_t ) ( usAddress & ( uint16_t ) 0x00ff );
pxTopOfStack--;

/* Next simulate the stack as if after a call to portSAVE_CONTEXT().
portSAVE_CONTEXT places the flags on the stack immediately after r0
to ensure the interrupts get disabled as soon as possible, and so
ensuring the stack use is minimal should a context switch interrupt
occur. */
*pxTopOfStack = ( StackType_t ) 0x00; /* R0 */
pxTopOfStack--;
*pxTopOfStack = portFLAGS_INT_ENABLED;
pxTopOfStack--;


/* Now the remaining registers.   The compiler expects R1 to be 0.
*/
*pxTopOfStack = ( StackType_t ) 0x00; /* R1 */
pxTopOfStack--;
*pxTopOfStack = ( StackType_t ) 0x02; /* R2 */
pxTopOfStack--;
*pxTopOfStack = ( StackType_t ) 0x03; /* R3 */
pxTopOfStack--;
*pxTopOfStack = ( StackType_t ) 0x04; /* R4 */
pxTopOfStack--;
*pxTopOfStack = ( StackType_t ) 0x05; /* R5 */
pxTopOfStack--;
*pxTopOfStack = ( StackType_t ) 0x06; /* R6 */
pxTopOfStack--;
*pxTopOfStack = ( StackType_t ) 0x07; /* R7 */
pxTopOfStack--;
*pxTopOfStack = ( StackType_t ) 0x08; /* R8 */
pxTopOfStack--;
*pxTopOfStack = ( StackType_t ) 0x09; /* R9 */
pxTopOfStack--;
*pxTopOfStack = ( StackType_t ) 0x10; /* R10 */
pxTopOfStack--;
*pxTopOfStack = ( StackType_t ) 0x11; /* R11 */
pxTopOfStack--;
*pxTopOfStack = ( StackType_t ) 0x12; /* R12 */
pxTopOfStack--;
*pxTopOfStack = ( StackType_t ) 0x13; /* R13 */
pxTopOfStack--;
*pxTopOfStack = ( StackType_t ) 0x14; /* R14 */
pxTopOfStack--;
*pxTopOfStack = ( StackType_t ) 0x15; /* R15 */
pxTopOfStack--;
*pxTopOfStack = ( StackType_t ) 0x16; /* R16 */
pxTopOfStack--;
*pxTopOfStack = ( StackType_t ) 0x17; /* R17 */
pxTopOfStack--;
*pxTopOfStack = ( StackType_t ) 0x18; /* R18 */
pxTopOfStack--;
*pxTopOfStack = ( StackType_t ) 0x19; /* R19 */
pxTopOfStack--;
*pxTopOfStack = ( StackType_t ) 0x20; /* R20 */
pxTopOfStack--;
*pxTopOfStack = ( StackType_t ) 0x21; /* R21 */
pxTopOfStack--;
*pxTopOfStack = ( StackType_t ) 0x22; /* R22 */
pxTopOfStack--;
```

```
		*pxTopOfStack = ( StackType_t ) 0x23; /* R23 */
		pxTopOfStack--;

		/* Place the parameter on the stack in the expected location. */
		usAddress = ( uint16_t ) pvParameters;
		*pxTopOfStack = ( StackType_t ) ( usAddress & ( uint16_t ) 0x00ff );
		pxTopOfStack--;

		usAddress >>= 8;
		*pxTopOfStack = ( StackType_t ) ( usAddress & ( uint16_t ) 0x00ff );
		pxTopOfStack--;

		*pxTopOfStack = ( StackType_t ) 0x26; /* R26 X */
		pxTopOfStack--;
		*pxTopOfStack = ( StackType_t ) 0x27; /* R27 */
		pxTopOfStack--;
		*pxTopOfStack = ( StackType_t ) 0x28; /* R28 Y */
		pxTopOfStack--;
		*pxTopOfStack = ( StackType_t ) 0x29; /* R29 */
		pxTopOfStack--;
		*pxTopOfStack = ( StackType_t ) 0x30; /* R30 Z */
		pxTopOfStack--;
		*pxTopOfStack = ( StackType_t ) 0x031;            /* R31 */
		pxTopOfStack--;

		/*lint +e950 +e611 +e923 */

		return pxTopOfStack;
}
/*-----------------------------------------------------------*/

BaseType_t xPortStartScheduler( void )
{
		/* Setup the hardware to generate the tick. */
		prvSetupTimerInterrupt();

		/* Restore the context of the first task that is going to run. */
		portRESTORE_CONTEXT();

		/* Simulate a function call end as generated by the compiler.  We
		will now jump to the start of the task the context of which we have
		just restored. */
		asm volatile ( "ret" );

		/* Should not get here. */
		return pdTRUE;
}
/*-----------------------------------------------------------*/

void vPortEndScheduler( void )
{
		/* It is unlikely that the AVR port will get stopped.  If required
		simply disable the tick interrupt here. */
}
/*-----------------------------------------------------------*/

/*
 * Manual context switch.  The first thing we do is save the registers so we
 * can use a naked attribute.
 */
void vPortYield( void ) __attribute__ ( ( naked ) );
void vPortYield( void )
{
```

```
            portSAVE_CONTEXT();
            vTaskSwitchContext();
            portRESTORE_CONTEXT();

            asm volatile ( "ret" );
}
/*-----------------------------------------------------------*/

/*
 * Context switch function used by the tick.  This must be identical to
 * vPortYield() from the call to vTaskSwitchContext() onwards.  The only
 * difference from vPortYield() is the tick count is incremented as the
 * call comes from the tick ISR.
 */
void vPortYieldFromTick( void ) __attribute__ ( ( naked ) );
void vPortYieldFromTick( void )
{
            portSAVE_CONTEXT();
            if( xTaskIncrementTick() != pdFALSE )
            {
                        vTaskSwitchContext();
            }
            portRESTORE_CONTEXT();

            asm volatile ( "ret" );
}
/*-----------------------------------------------------------*/

/*
 * Setup timer 1 compare match A to generate a tick interrupt.
 */
static void prvSetupTimerInterrupt( void )
{
uint32_t ulCompareMatch;
uint8_t ucHighByte, ucLowByte;

            /* Using 16bit timer 1 to generate the tick.  Correct fuses must be
            selected for the configCPU_CLOCK_HZ clock. */

            ulCompareMatch = configCPU_CLOCK_HZ / configTICK_RATE_HZ;

            /* We only have 16 bits so have to scale to get our required tick
            rate. */
            ulCompareMatch /= portCLOCK_PRESCALER;

            /* Adjust for correct value. */
            ulCompareMatch -= ( uint32_t ) 1;

            /* Setup compare match value for compare match A.  Interrupts are
            disabled before this is called so we need not worry here. */
            ucLowByte = ( uint8_t ) ( ulCompareMatch & ( uint32_t ) 0xff );
            ulCompareMatch >>= 8;
            ucHighByte = ( uint8_t ) ( ulCompareMatch & ( uint32_t ) 0xff );
            OCR1AH = ucHighByte;
            OCR1AL = ucLowByte;

            /* Setup clock source and compare match behaviour. */
            ucLowByte = portCLEAR_COUNTER_ON_MATCH | portPRESCALE_8;
            TCCR1B = ucLowByte;
            TCCR1A = 0x00;

            /* Enable the interrupt - this is okay as interrupt are currently
            globally disabled. */
```

```
		ucLowByte = TIMSK1;
		ucLowByte |= portCOMPARE_MATCH_A_INTERRUPT_ENABLE;
		TIMSK1 = ucLowByte;
}
/*-----------------------------------------------------------*/

#if configUSE_PREEMPTION == 1

		/*
		 * Tick ISR for preemptive scheduler.  We can use a naked attribute
		 * as the context is saved at the start of vPortYieldFromTick().
		 * The tick count is incremented after the context is saved. */
		ISR(TIMER1_COMPA_vect,ISR_NAKED)
		{
			vPortYieldFromTick();
			asm volatile ( "reti" );
		}
#else

		/*
		* Tick ISR for the cooperative scheduler.  All this does is incre
		* ment the tick count.  We don't need to switch context, this can
		* only be done by manual calls to taskYIELD();
		*/
		ISR(TIMER1_COMPA_vect)
		{
			xTaskIncrementTick();
		}
#endif
```

Liite 2. Schematic.SchDoc 1 (1)