SAVONIA

Carsinogen

Kauko-ohjattavan lentokoneen hallintaohjelmisto

Jani Taskinen

Opinnäytetyö

___. ___. _____ _____

Ammattikorkeakoulututkinto

SAVONIA-AMMATTIKORKEAKOULU **OPINNÄYTETYÖ**
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma Tietotekniikan koulutusohjelma	
Työn tekijä(t) Jani Taskinen	
Työn nimi Carsinogen	
Päiväys 28.5.2012	Sivumäärä/Liitteet 31/33
Ohjaaja(t) lehtori Sami Lahti, lehtori Jussi Koistinen	
Toimeksiantaja/Yhteistyökumppani(t) NCOW Technology Oy, toimitusjohtaja Mikko Jakonen	

Tiivistelmä

Project Carsinogen –projektin tavoitteena oli tuottaa Proof of Concept –mallikappale kauko-ohjattavan lennokin laitteisto- ja ohjelmistokokonaisuudesta ja osoittaa, että toteutus on kustannuksiltaan kaupallisesti hyödynnettävissä. Opinnäytetyönä pyrin toteuttamaan edellä mainitun kokonaisuuden vaatiman ohjelmiston.

Työ toteutettiin Microsoftin Visual Studio -ohjelmistokehitysympäristöllä, .NET Framework 4.0 -ohjelmistokomponenttikirjastoa ja C#-ohjelmointikieltä käyttäen.

Lopputuloksena syntynyt ohjelmisto on jaettu kahteen osaan: ensimmäinen osakokonaisuus pitää sisällään lentokoneeseen tulevan ohjelmiston, joka ohjaa siivekkeitä, tarkkailee sensoreita ja välittää mittadataa maa-asemalle.Toinen puolikas on maa-asemalla ajettava ohjelma, joka sisältää käyttöliittymän, logiikan lentokoneelta tulevan tiedon lukemiselle ja käsittelylle sekä tuen joystickilla tapahtuvalle ohjaukselle.

Työn tuloksena saatiin aikaan järjestelmä kauko-ohjattavan lentokoneen laitteiden hallintaan, joskaan aivan kaikkia suunniteltuja ominaisuuksia ei aikataulullisista syistä ehditty toteuttamaan.

Avainsanat
C#, .NET, DirectX, lentokone, kauko-ohjaus

SAVONIA UNIVERSITY OF APPLIED SCIENCES **THESIS**
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Jani Taskinen			
Title of Thesis Carsinogen			
Date	28 May 2012	Pages/Appendices	31/33
Supervisor(s) Mr Sami Lahti, Lecturer, Mr Jussi Koistinen, Lecturer			
Client Organisation/Partners NCOW Technology Ltd.			

Abstract

Project Carsinogen was an attempt to create a Proof of Concept of a unity of software and hardware for a remote controlled aircraft that could be commercialized later on. The purpose of this thesis was to produce the software needed.

The project was carried out using the Microsoft Visual Studio integrated development environment, .NET Framework 4.0 software framework and the C# programming language.

The software produced is divided into two parts: the first part of a whole consists of the software installed into the aircraft which controls the wanes, monitors the sensors and transmits data to the Ground Station. The second half includes the program ran in the Ground Station, including the user interface, logic to process the data received from the aircraft, and support for joystick control.

As a result, a system for controlling a remote controlled aircraft was created, although all of the planned features could not be actualized due to scheduling difficulties.

Keywords
C#, .NET, DirectX, aeroplane, remote control

ALKUSANAT

Tämä opinnätetyö tehtiin NCOW Technology Oy:lle talven 2011 ja kevään 2012 aikana. Työn ohjaajina toimivat lehtorit Sami Lahti ja Jussi Koistinen Savonia-ammattikorkeakoulusta. Haluan kiittää Mikko Jakosta ja Jari Taskista NCOW Technologylta ja työn ohjaajia koulun puolelta. Hyvä ystäväni Jarno Iivarinen ansaitsee kiitoksen mainioista vinkeistä.

Erityiskiitos kuuluu avopuolisolleni, joka jaksoi ja kesti kaikki ne pitkät tunnit, jotka työn tekeminen vaati.

Tampereella 28.5.2012

Jani Taskinen

SISÄLTÖ

LIITTEET

TERMIT JA LYHENTEET

.NET Framework	.NET Framework on Microsoftin kehittämä ohjelmistokomponenttikirjasto, jota Microsoftin Visual Studio.NET-ympäristössä kehitetyt ohjelmistot käyttävät.
Asiakas-palvelin-malli	Yksinkertainen ohjemointimalli, jolla maa-asema ja lentokone kommunikoivat ja jakavat tehtäviä keskenään
C#	C# on Microsoft-yhtiön .NET-konseptia vuonna 2000 julkaisema, jolla on pyritty yhdistämään C++:n tehokkuus ja Java-kielen helppokäyttöisyys.
CLR	Common Language Runtime, virtuaalikone, joka kääntää kehitysympäristössä tuotetun ohjelmakoodin binäärimuotoon
COB	Computer-On-Board, lentokoneeseen asennettu tietokone ja käyttöjärjestelmä
Debuggeri	Tietokoneohjelma, jota käytetään muiden tietokoneohjelmien ohjelmointivirheiden jäljittämiseen ja korjaamiseen.
DirectX	Kokoelma ohjelmointirajapintoja tietokoneohjelman ja laitteiston välille
Ecma International	Kansainvälinen, voittoa tuottamaton ICT-alan standardointijärjestö
ENGCTRL	Engine Control, moottorin tehon ohjaus
FCS	Flight Control System
GPS	Global Positioning System, maailmanlaajuinen satelliittipaikannusjärjestelmä
GS	Ground Station, maa-asema, josta lentokonetta ohjataan
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit, lentokoneen liikkeiden ja kiihtyvyyden seuranta eri antureista tulevan mittadatan perusteella
Intellisense	Microsoftin implementaatio mm. web-selaimissa, sähköpostisovelluksissa ja hakukoneissa käytetystä automaattitäydennyksestä
Joystick	Joystick on tietokoneen oheislaite, joka lähettää tiedon sauvan kallistuskulmasta tietokoneelle, jonka antaman tiedon perusteella lentokone kääntyy haluttuun suuntaan. Tämän lisäksi siinä on painonäppäimiä joita käytetään eri toimintojen

	laukaisemiseen.
LIGHTS	Valojen ohjaus
NMEA	National Marine Electronics Association, erinäisten suunnistuslaitteiden tiedonvälitysstandardeja määrittelevä järjestö
PDU	Power Distribution Unit, virrankulutuksen seuranta
SDK	Software Development Kit, pakkaus, johon on sisällytetty ohjelmistokehityksessä tarvittavia työkaluja ja rajapintoja
VCP	Virtual COM Port, ohjelma, joka mahdollistaa esim. USB-laitteen käytön sarjaportissa kiinni olevan laitteen tavoin.

1 JOHDANTO

Kauko-ohjattavalle lentokoneelle voi keksiä monia käyttötarkoituksia. Koneen lennättäminen voi olla yksinkertaista ajanvietettä palvelematta sen suurempaa tarkoitusta, tai kone voidaan valjastaa hyötykäyttöön lastaamalla sen kyytiin esimerkiksi laitteita, joilla saadaan vaikkapa hyödyllistä informaatiota ympäristöstä. Tässä vaiheessa koneen potentiaaliset käyttötarkoitukset ja -kohteet moninkertaistuvat ja ainoastaan mielikuvitus ja jossain vaiheessa fysiikan lait asettavat rajoitukset koneen käytölle.

Käyttökohteiden lisääntyessä voivat yhä useammat tahot kiinnostua koneen käytöstä. Koneen hyötykuormaa vaihtamalla melkeinpä kuka tahansa voi keksiä sille jotain käyttöä. Tässä vaiheessa kohderyhmät koneen kaupalliselle hyödyntämiselle voivat vaihdella yksittäisistä ihmisistä aina valtiollisiin toimijoihin asti.

Tämän opinnäytetyön tarkoituksena on toimittaa NCOW Technology Oy:lle mallikappale ohjelmistosta, jolla saataisiin kauko-ohjattava lentokone nousemaan ilmaan, lentämään ja loppujen lopuksi laskeutumaan hallitusti. Lentokone koottiin samaan aikaan, kun ohjelmistoa kirjoitettiin. Projektin perimmäisenä tavoitteena oli todistaa, että suhteellisen edullisista komponenteista ja yksinkertaisesta ohjelmistosta voidaan saada aikaan kustannustehokas ja toimiva kokonaisuus.

Opinnäytetyöraportti jakaantuu seuraaviin osiin: projektin ja siinä käytettyjen tekniikoiden yleiseen kuvaukseen, lopputuloksena syntyneen ohjelmiston tarkempaan tarkasteluun ja loppupäätelmiin projektin tuloksista.

2 KÄYTETYT TEKNIIKAT

2.1 .NET Framework

.NET Framework on Microsoftin kehittämä ohjelmistokomponenttikirjasto, joka tukee useita eri ohjelmointikieliä. Käytetyimpiä näistä varmastikin ovat C# ja VB.Net. .NET Framework koostuu kahdesta osasta: luokkakirjastoista ja ajoympäristöstä (CLR). (Microsoft, .NET Framework 4.)

Luokkakirjastot ovat tarjolla kaikille .NETiä hyödyntäville ohjelmointikielille. Ne sisältävät ja tarjoavat monipuoliset työkalut aina käyttöliittymäsuunnittelusta tietokantojen ja tiedostojen käsittelyyn, mikä osaltaan helpottaa ohjelmoijan tehtäviä. Ohjelmoijat tuottavat ohjelmistoja yhdistämällä omaa ohjelmakoodiaan .NETin ja mahdollisten muiden kirjastojen kanssa.

CLR tarjoaa ns. virtuaalikoneen, joka kääntää kehitysympäristön tuottaman esikäännetyn ohjelmakoodin binäärimuotoon, jota käyttöjärjestelmä voi lukea ja suorittaa. CLR tarjoaa myös palveluja, jotka vastaavat ohjelmiston tietoturvasta, muistin hallinnasta ja virheiden käsittelystä. (.NET Framework, Wikipedia.)

.NET Frameworkin uusin versio 4.0 on julkaistu 12. huhtikuuta 2010.

2.2 C#

C# on Microsoftin .NET-konseptia varten kehittämä ohjelmointikieli, joka julkaistiin kesäkuussa 2000. Kielen kehittämisen päätavoitteena oli luoda useanlaisiin ympäristöihin soveltuva yksinkertainen, moderni ja yleiskäyttöinen oliopohjainen ohjelmointikieli. (C#, Wikipedia.)

Ecma Internationalin standardoimana C#:lle on lueteltu mm. seuraavia tavoitteita:

- Kielen ja sen implementaatioiden pitää tarjota erinäisiä suunnitteluperiaatteita, kuten automatisoitu roskienkeruu ja alustamattomien muuttujien havaitseminen.
- Lähdekoodin siirrettävyys on erittäin tärkeää.
- Vaikka C#-sovellusten on tarkoitus olla taloudellisia muistinkäytön ja prosessoritehon kannalta, ei kielen ole tarkoitus kilpailla suorituskyvyssä esimerkiksi C:n kanssa. (Ecma International, C# Language Specification.)

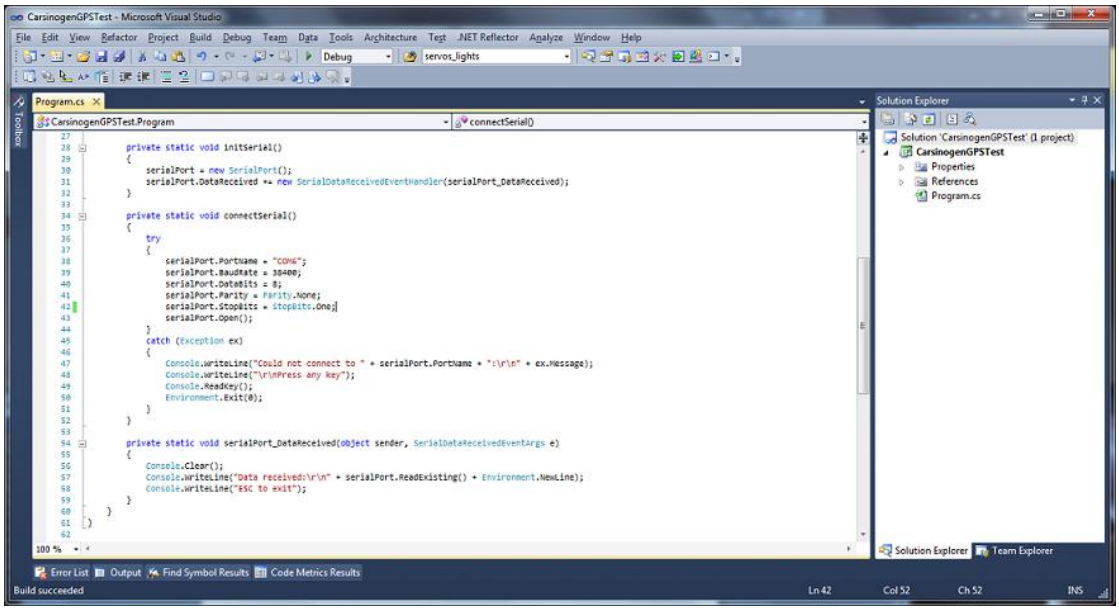
C#:n viimeisin versio 4.0 julkaistiin 12. huhtikuuta 2010.

2.3 Visual Studio

Visual Studio on Microsoftin ohjelmistokehitysympäristö, jolla voi kehittää sovelluksia useita eri ohjelmointikieliä käyttäen. Visual Studiolla voi kehittää mm. Windows-, web- ja mobiilisovelluksia sekä siihen voidaan integroida myös kolmansien osapuolten lisäosia. (Microsoft Visual Studio, Wikipedia.)

Visual Studio 2010 sisältää Intellisenseä ja koodin refaktorointia tukevan koodieditorin (kuva 1). Joissain versioissa integroitu debuggeri toimii sekä lähdekoodi- että laitetasolla.

Muita sisäänrakennettuja työkaluja ovat mm. graafisten käyttöliittymien, web-sovellusten, luokkien ja tietokantaskeemojen suunnitteluun tarkoitetut apuohjelmat.



KUVA 1. GPS-testiohjelma Visual Studion koodieditorissa

Työssä käytettiin Visual Studion versiota 2010. Uusin versio, Visual Studio 2011, julkaistiin beta-versiona 29. helmikuuta 2012.

2.4 Laitteistotoimittajan tarjoamat ohjelmointirajapinnat

Laitteistotoimittaja Pololun SDK:t tarjoavat korkeamman tason rajapintoja laitteiston ohjaamiseen ja niiltä saatavan datan käsittelyyn. (Pololu Robotics and Electronics.) Pololun laitteistoja projektissa käytettiin siivekkeiden servo-ohjaimissa, kiihtyvyysantureissa ja gyroskoopeissa sekä GPS-paikannuslaitteessa.

Pololun moottori- ja servo-ohjaimet toimivat USB-porttien välityksellä. Valmistajan Internet-sivuilta löytyvän SDK:n avulla servojen ohjaaminen kävi esimerkkeihin tutustumisen jälkeen suhteellisen helposti.

Saman valmistajan kiihtyvyysanturit ja gyroskoopit puolestaan toimivat USB-adapterin välityksellä sarjaportin kautta. Tämän takia lähdekoodiin oli tutustuttava huomattavasti servo-ohjaimia tarkemmin, koska esimerkki sisälsi huomattavan paljon suoraan laitteistoa ohjaavia bittioperaatioita ja toiminnan ymmärtäminen vaati näin ollen reilusti enemmän työtä.

Pololun toimittama GPS-vastaanotin puolestaan on USB-laite, joka toimii virtuaalisen sarjaportin (VCP) kautta. Tämä mahdollistaa keskustelun laitteen kanssa millä tahansa taajuudella baudeina, jolloin laitteen välittämän tiedon päivitystaajuutta voidaan säätää suoraan laitteistotasolla suuremmin kirjoitettua ohjelmistoa muokkaamatta.

2.5 DirectX

DirectX on Microsoftin Windows-käyttöjärjestelmälle kehittämä, erityisesti pelien ja multimedian käsittelyyn tarkoitettu kokoelma ohjelmointirajapintoja tietokoneohjelman ja laitteiston välille. DirectX koostuu useasta erillisestä komponentista, joista työssä hyödynnettiin DirectInput-kirjastoa. DirectInput tarjoaa rajapinnat datan keräämiseen käyttäjältä, kuten tässä tapauksessa joystickin liikkeiden ja nappien painallusten havaitsemiseen. (DirectX, Wikipedia.)

DirectInput sisältää monia hyödyllisiä ominaisuuksia:

- Sovellus voi kerätä laitteelta dataa, vaikka se olisi taustalla eikä aktiivisena.
- DirectInput tarjoaa täyden tuen millaiselle laitteelle tahansa.
- Toimintakartoituksen avulla sovelluksille voi lähettää dataa ilman, että itse sovelluksen tarvitsee tietää, millainen laite on kyseessä.

Vaikka DirectInput onkin osa DirectX-kokoelmaa, ei sitä ole juurikaan päivitetty DirectX 8:n jälkeen. Tämä oli otettava huomioon lentokoneen ohjauksen suunnittelussa, koska em. versio DirectInputista vaatii toimiakseen .NET Framework 2.0:n, kun taas kirjoitettu ohjelmisto on tehty .NET Framework 4.0:lle. Taaksepäin yhteensopivuus kuitenkin toimii sulavasti, eikä tästä pienestä yksityiskohdasta aiheutunut loppujen lopuksi erityisiä ongelmia.

DirectX:n viimeisin valmis versio DirectX11 (versionumero 6.01.7601.17514) on julkaistu 16.2.2011.

3 YHTEYS LENTOKONEEN JA MAA-ASEMAN VÄLILLÄ

Alun perin datan siirtämiseen lentokoneen ja maa-aseman välillä oli kaksi vaihtoehtoa: USB Redirector ja ohjelmaan itse rakennettu asiakas-palvelin-malli. Vaihtoehtojen tutkimisen jälkeen asiakas-palvelin-mallin todettiin soveltuvan tarkoitukseen paremmin.

3.1 USB Redirector

USB Redirector on Incentives Pro -nimisen yrityksen tarjoama ohjelmisto, jolla voidaan ohjata USB-laitteita toisessa tietokoneessa TCP/IP-yhteyden yli samoin kuin laitteiden ollessa kiinni paikallisessa tietokoneessa.

Ratkaisun hyvinä puolina asiakas näki mm. seuraavat asiat:

- valmis ja testattu ohjelmisto
- pieni latenssi
- vähemmän ohjelmointia ja testausta ja näin ollen projektin mahdollisesti nopeampi valmistuminen

Haittoja USB Redirectorissa puolestaan olivat:

- Datan käsittely COB:n päässä: Koska USB-laite käyttäytyy, kuin se olisi kiinni paikallisessa koneessa, olisi tietojen tallentaminen voinut olla hankalampaa kuin asiakas-palvelin-mallilla.
- Kolmannen osapuolen suljetun lähdekoodin ohjelmisto: Toiminnan läpinäkyvyys heikkenee.
- Ohjelmiston lisenssit ja muuttuuko ohjelma tulevaisuudessa maksulliseksi.

3.2 Asiakas-palvelin-malli

Asiakas-palvelin-malli oli alun perinkin asiakkaan ensisijainen toive USB Redirectoriin liittyvien kysymysten vuoksi. Tämä toteutus valitsemalla saatiin räätälöityä juuri asiakkaan tarpeisiin soveltuvat rajapinnat datan välitykseen ja käsittelyyn ja ennen kaikkea kehittämällä tiedonsiirtojärjestelmä itse voitiin olla varmoja sen toiminnan eri vaiheiden ymmärtämisestä ja näin ollen jäljittämään ongelmat ja virhetilanteet vaivattomammin ja nopeammin.

Asiakas-palvelin-malli oli toivottu vaihtoehto myös siksi, että koneeseen voitaisiin ottaa yhteys useammalta maa-asemalta samaan aikaan. Tämä ominaisuus mahdollistaa koneen käyttömatkan moninkertaistamisen ohjauksen ollessa riippumaton yksittäisen WLAN-tukiaseman kantomatkan aiheuttamista rajoitteista.

Yhteys lentokoneen ja maa-aseman välillä on toteutettu yhdistellen synkronista ja asynkronista yhteyttä. Koneelle välitettävien komentojen täytyy luonnollisesti olla mahdollisimman pian perillä, kun taas vastausta voi joissain tapauksissa hetken odottaa.

Palvelimena ratkaisussa toimii lentokone. Maa-asema lähettää komentoja ja kyselyjä koneelle ja saa vastauksena kulloisenkin komponentin ajantasaisen tilan. Yhteydet palvelimeen käsitellään erillisissä säikeissä, jolloin tiedonsiirto ei häiritse ohjelman muuta toimintaa.

```
TcpClient client = this.tcpListener.AcceptTcpClient();
Connection connection = new Connection(this, client);
Thread item = new Thread(new ThreadStart(connection.StartConn));
item.Start();
```

Yllä olevasta koodista nähdään, kuinka serveri käsittelee saapuvat yhteyspyynnöt. Palvelimen hyväksyessä tulevat yhteyspyynnöt luodaan kutakin yhteyttä varten uusi instanssi *Connection*-luokasta, joka vastaa kulloinkin kyseessä olevan yhteyden hallinnasta. Kaikki nämä instanssit ajetaan omissa säikeissään, mikä mahdollistaa useamman yhteyden käsittelyn.

4 COMPUTER-ON-BOARD

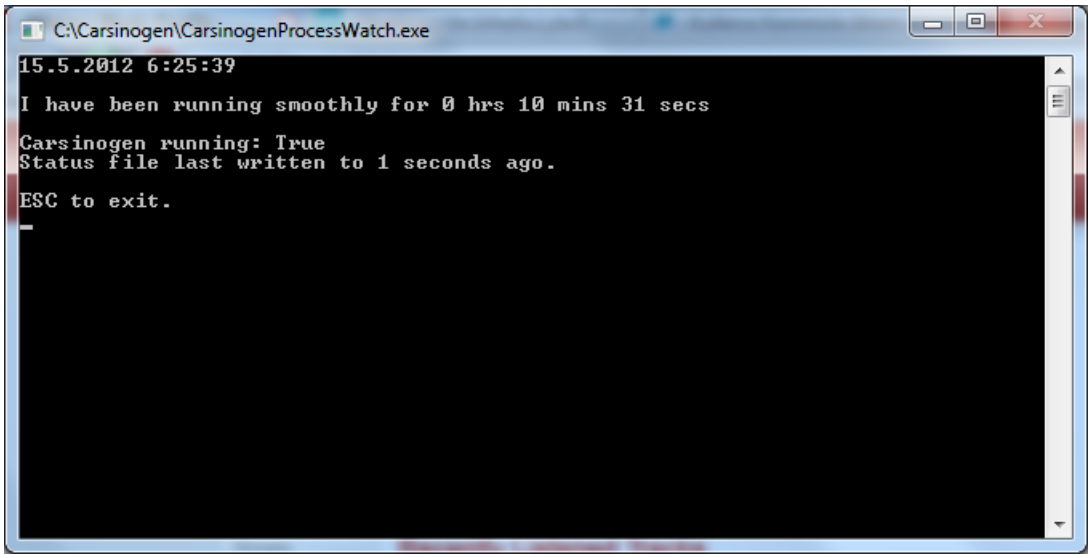
COB eli Computer-On-Board käsittää lentokoneeseen asennetun tietokoneen ja siihen asennetun ohjelmiston. Sekä COB että GS sisältävät periaatteessa täysin samat lentokoneen hallitsemiseen tarvittavat komponentit. Koneen hallintaan kuuluva käyttöliittymä sijaitsee maa-asemalla, joten ohjelmiston yksittäisiä osia käsitellään tarkemmin luvussa GROUND STATION.

Tässä luvussa käsitellään COB:n ominaispiirteitä sekä erilaisia tekniikoita ja erillisohjelmia, joilla on pyritty parantamaan virheistä palautumista, varmistamaan vastaanotettavan ja käsiteltävän datan yhteneväisyys todellisen tilanteen kanssa sekä äärimmäisissä hätätilanteissa tuomaan lentokone laskuvarjolla maahan.

4.1 Vahtikoira

Vahtikoiraksi nimetty komponentti on pieni itsenäinen ohjelma, joka toimii taustalla ja valvoo varsinaisen ohjelmaprosessin kuntoa ja toimintakykyä. Carsinogen päivittää tietyin väliajoin lokitiedostoa, jonka ainut tarkoitus on osoittaa, että ohjelmisto toimii kuten pitää. Vahtikoira puolestaan tarkkailee, koska kyseistä tiedostoa on viimeksi päivitetty ja saamiensa tulosten mukaan valitsee tilanteeseen sopivan toimintatavan:

- Mikäli ohjelmaprosessi on käynnissä ja tiedostoa päivitetään tasaisin väliajoin, on kaikki kunnossa eikä mitään toimenpiteitä tarvita.
- Mikäli ohjelmaprosessi lakkaa olemasta, vahtikoira käynnistää sen.
- Mikäli ohjelmaprosessi on käynnissä mutta tiedostoa ei tietyn ajan kuluessa päivitetä, on todennäköistä, ettei ohjelmisto toimi kuten pitäisi. Tällöin vahtikoira lopettaa ohjelmistoprosessin ja käynnistää sen uudelleen.



KUVA 2. Vahtikoira toiminnassa.

Kuva 2 esittelee vahtikoiran toiminnassa, silloin kun ohjelmisto toimii kuten pitääkin. Tilastoinnin ja laadunvalvonnan vuoksi ohjelma ilmoittaa kellonajan ja päivämäärän, kauanko se on ollut toiminnassa ja koska valvottavaa tiedostoa on viimeksi käsitelty. Näistä tiedoista voi olla hyötyä virhetilanteiden sattuessa.

4.2 Väliaikaiset tiedostot

Väliaikaiset tiedostot ovat periaatteessa lokitiedostoja, mutta niitä ylläpidetään ja tarkkaillaan ainoastaan ohjelmiston käytön aikana. Vahtikoiran toiminta perustuu siihen, että ohjelman toimiessa oikein päivittyy Vahtikoiran tarkkailema tiedosto tietyin väliajoin, muuten oletetaan jonkin olevan vialla.

Lennon aikana ylläpidetään myös tiedostoa servojen senhetkisestä asennosta mahdollisen yhteyskatkon varalta. Tiedosto luodaan, kun servoja liikutetaan käyttäjän toimesta ensimmäisen kerran, ja se poistetaan vain ja ainoastaan silloin, kun ohjelman ajo lopetetaan luonnollisesti käyttäjän toimesta.

Tiedosto sisältää tiedon kunkin servon asennosta viimeisimmän vastaanotetun komennon jälkeen. Jos yhteys mistä syystä tahansa katkeaa, tarkistaa ohjelmisto uudelleenyhdistettäessä, onko tiedosto olemassa, ja jos on, välitetään tiedoston sisältämät arvot maa-asemalle. Näin voidaan varmistua datan samasta sisällöstä sekä maassa että ilmassa, ja käyttäjä voi olla varma siitä, että tieto on varmasti ajantasaista.

4.3 Paniikkikytkin

Siltä varalta, että kaikki menee täysin pieleen ja/tai yhteys koneeseen katkeaa siten, ettei sitä saada tietyssä ajassa palautettua, on ohjelmistoon sisällytetty ns. ”paniikkikytkin”. Tämä komponentti toimii yhteistyössä kohdan Vahtikoira-komponentin kanssa. Vahtikoiran tarkkailessa itse ohjelmistoprosessin kuntoa ja toiminnallisuutta, on paniikkikytkin kiinnostunut verkkoyhteyden toiminnasta.

Mikäli kone lennossa ollessaan joutuu yhteydettömään tilaan, eikä yhteyttä maa-asemaan saada tietyssä ajassa palautettua, laukaisee kytkin varotoimenpiteenä laskuvarjon ja sen jälkeen kytkee laitteiston pois päältä. Tarpeen vaatiessa voidaan varjo laukaista myös maasta käsin.

4.4 Lokitiedostot

Computer-On-Board ylläpitää kahta lokitiedostoa, joista toiseen kirjataan ohjelmiston ja laitteiston tila ja tapahtumat niiden toiminnasta, kun taas toinen keskittyy verkkoliikenteen tapahtumiin ja kirjaa ylös serverin käynnistykset, sammutukset ja vastaanotetut komennot.

5 GROUND STATION

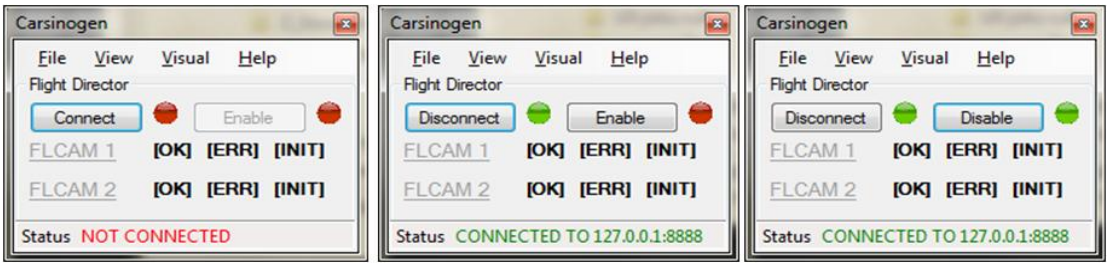
Ground Station eli maa-asema käsittää fyysisenä kokonaisuutena joystickin sekä tietokoneen jolla ohjelmiston maa-aseman puoleista osiota käytetään. Ohjelma on verrattaen kevyt, eikä mitään erillistä asentamista tarvita, joten ohjelmaa voi ajaa vaikkapa siirrettävältä massamuistilta, kuten muistitikulta. Koska kaikki data ja komennot siirtyvät TCP/IP-yhteyden yli, on langaton verkkokortti koneessa pakollinen. Muutoin ohjelman käyttämiseen ei laitteiston osalta ole sen kummempia vaatimuksia.

Ohjelmisto vaatii toimiakseen .NET Frameworkin version 4.0 sekä DirectX:n vaatimat kirjastot. Ohjelmaa ensi kertaa ajettaessa käyttöjärjestelmä ilmoittaa, mikäli jotain näistä puuttuu ja tarjoutuu asentamaan ne.

Tietokoneohjelmana GS pitää sisällään graafisen käyttöliittymän lentokoneen hallintaan ja siihen asennettujen anturien ja sensorien välittämän datan tarkkailuun ja käsittelyyn, sekä taustalogiikan käyttäjän komentojen välittämiseen lentokoneelle.

5.1 Pääikkuna

Pääikkuna on käyttökokemuksen ja käyttäjäystävällisyyden nimissä pidetty yksinkertaisena, ja sitä käytetäänkin pääasiassa yhteyden muodostamiseen koneeseen ja koneessa olevan laitteiston käynnistämiseen. Näiden perustoimintojen lisäksi pääikkuna pitää sisällään käyttäjäkokemukseen vaikuttavia asiakkaan toivomia ominaisuuksia, kuten ikkunoiden paikan tallentamisen.

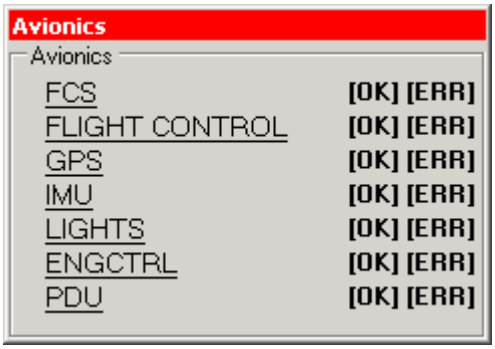


KUVA 3. Pääikkunan eri vaiheet muodostettaessa yhteys lentokoneeseen.

Kuva 3 esittää havainnollistetun esimerkin yhteyden muodostamisesta lentokoneeseen. Alkutilanteessa indikaattoreina toimivat valot ja tilapalkissa oleva teksti ovat punaisia, kertoen käyttäjälle sekä verkko- että laitteistoyhteyden olevan poissa käytöstä. Verkkoyhteyden avautuessa muuttuu sitä indikoiva valo vihreäksi, kertoen käyttäjälle nopeasti yhteyden olevan auki. Niin ikään tilapalkin teksti vaihtuu vihreäksi kertoen käyttäjälle lentokoneen IP-osoitteen ja käytössä olevan portin. Kolmannessa vaiheessa käynnistetään varsinainen lentokoneeseen sijoitettu laitteisto. Laitteiston käynnistyttyä muuttuu viimeinenkin indikaattori vihreäksi kertoen käyttäjälle että kone on valmis ohjattavaksi.

5.2 Avionics

Avionics on yksinkertainen käyttökokemusta helpottava komponentti, joka tarjoaa pikaisen pääsyn lentokoneen hallinta- ja monitorointijärjestelmiin.

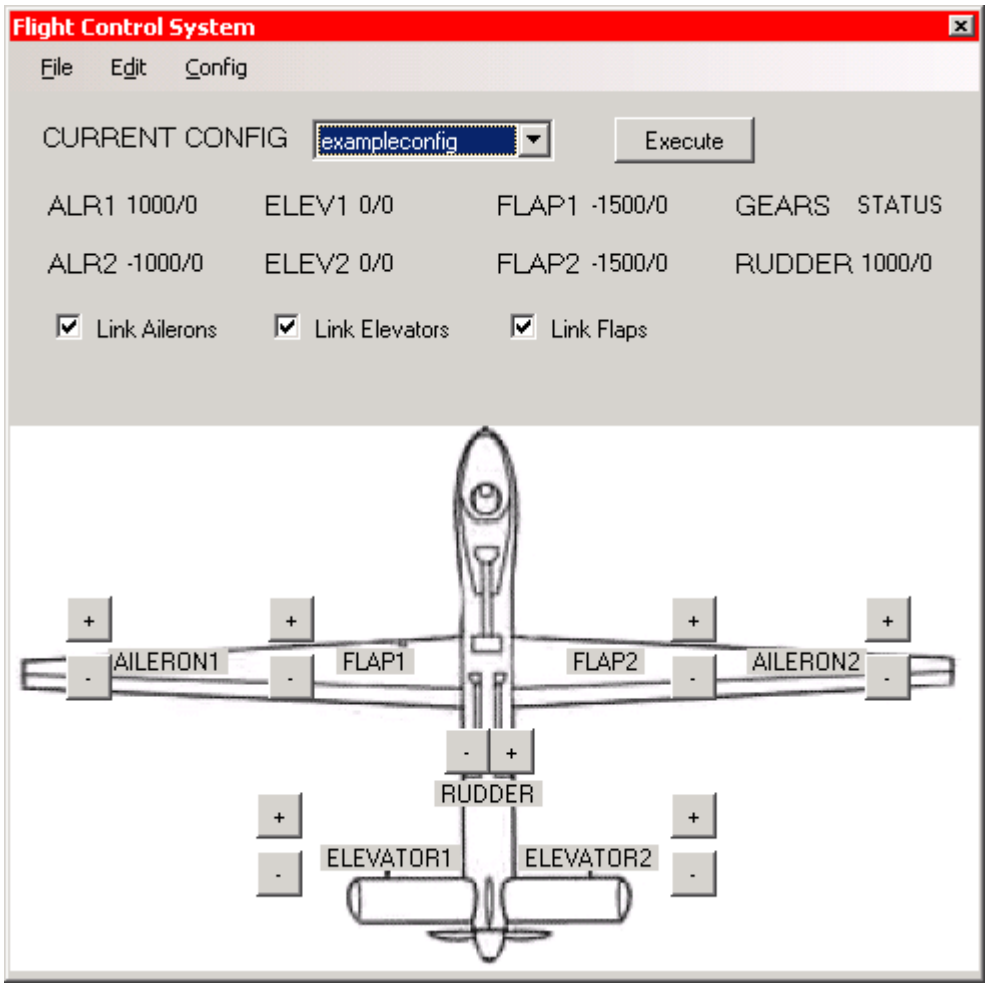


KUVA 4. Avionics-ikkuna

Kuva 4 on kuvakaappaus Avionics-ikkunasta. Kutakin kohdetta klikkaamalla aukeaa kyseessä oleva ikkuna, joka tarjoaa näkymän kyseisen komponentin tietoihin. Joissain tapauksissa prosessoritehon ja verkkokapasiteetin säästämiseksi tarvittava järjestelmä käynnistetään vasta ikkunan avautuessa, kriittisemmät komponentit käynnistyvät heti kun laitteisto käynnistetään.

5.2.1 FCS

Flight Control System on ikkuna, josta käyttäjä näkee reaaliaikaisesti kutakin siivekettä ohjaavan servon asennon. Tästä ikkunasta käyttäjä voi ladata ja tallentaa haluamiaan esiasetuksia, joilla voidaan kompensoida erilaisia olosuhteita, kuten tuulta, ja näin saada lentokone käyttäytymään rauhallisesti ja hallitusti.



KUVA 5. Flight Control System

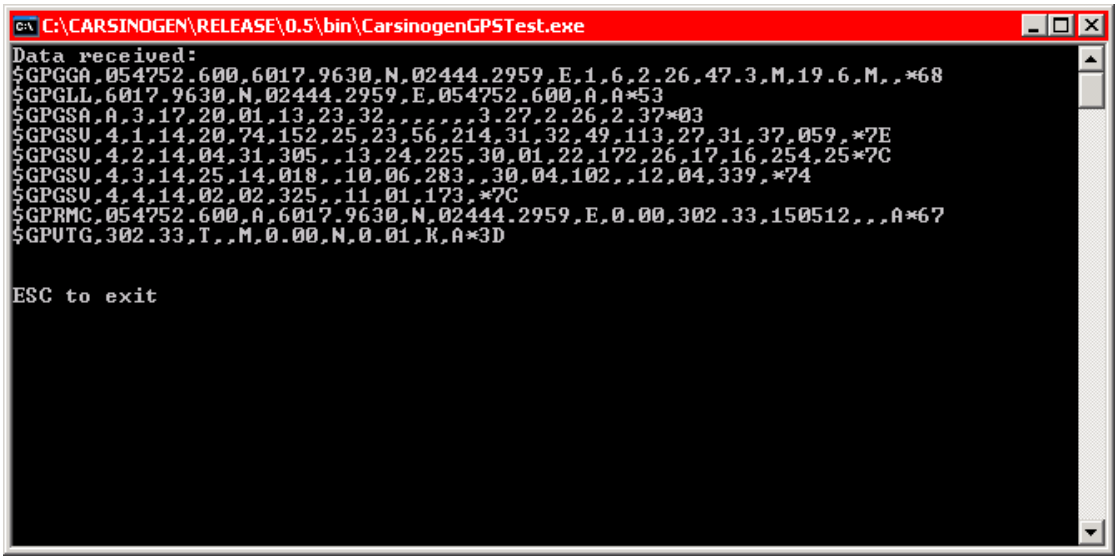
Kuvassa (kuva 5) nähdään FCS:n ikkuna. Kutakin siivekettä klikkaamalla avautuu näkyviin liukusäädin, jota käyttäen servojen asentoa voidaan muuttaa. Hienosäätöä varten käyttöliittymään on lisätty ’+’- ja ’-’-näppäimet. Eri konfiguraatioiden nopea lataaminen onnistuu kuvassa näkyvän pudotusvalikon kautta. Konfiguraatioiden tallentaminen tapahtuu Config-valikon kautta, jonka välityksellä voi luonnollisesti myös ladata asetuksia.

Vaikka koneen ohjaaja voikin ohjata servojen asentoja maasta käsin joko FCS:n tai joystickin välityksellä, varsinainen muutos tapahtuu lentokoneen päässä.

5.2.2 GPS

GPS eli Global Positioning System on maailmanlaajuinen paikannusjärjestelmä, jonka avulla lentokoneen sijainti, nopeus ja suunta voidaan määrittää hyvinkin tarkasti. Tämän lisäksi GPS:ltä saadaan koko järjestelmälle sekä lentoonlähdölle ja laskeutumiselle tarkat ajat.

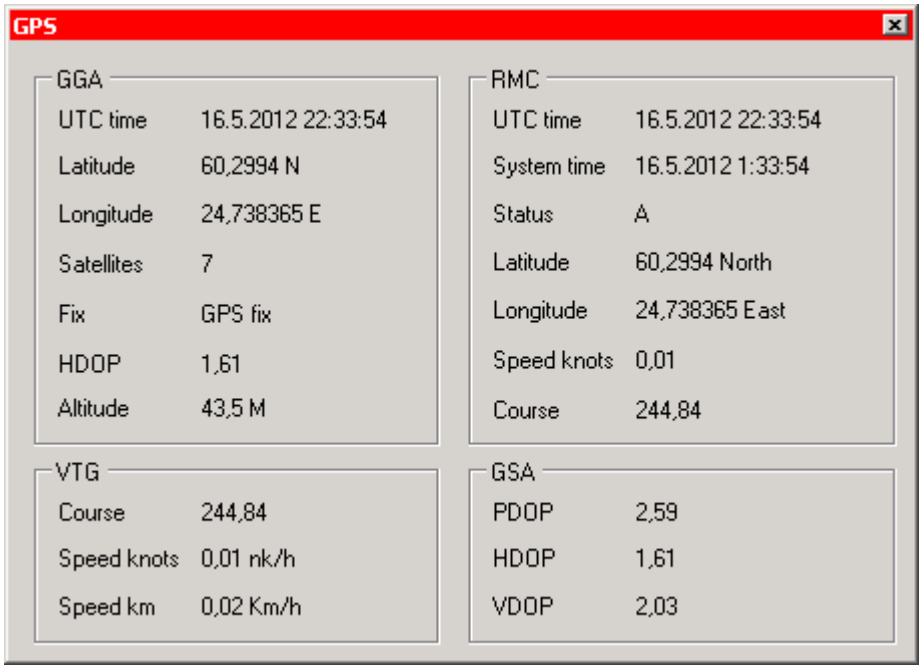
Lentokoneeseen sijoitettu GPS-vastaanotin välittää dataa NMEA 0183 –protokollan mukaisesti. GPS:n näyttämä data on ainoastaan mittadataa, eikä käyttäjä voi siihen suoraan vaikuttaa.



KUVA 6. GPS-vastaanottimen raakadataa

Yllä (kuva 6) on kuvakaappaus testiohjelman tulostamasta GPS-vastaanottimen syöttämästä raakadatasta, josta kulloinkin tarpeelliset tiedot luetaan ja näytetään niille kuuluvissa paikoissa.

Kuva 7 puolestaan näyttää raakadatasta poimittua eriteltyä dataa. Näiden ja IMUn tarjoamien paikka- ja asentotietojen avulla koneen ohjaajan pitäisi pystyä pitämään kone kurssissa, oikealla korkeudella ja muutenkin hallinnassa.

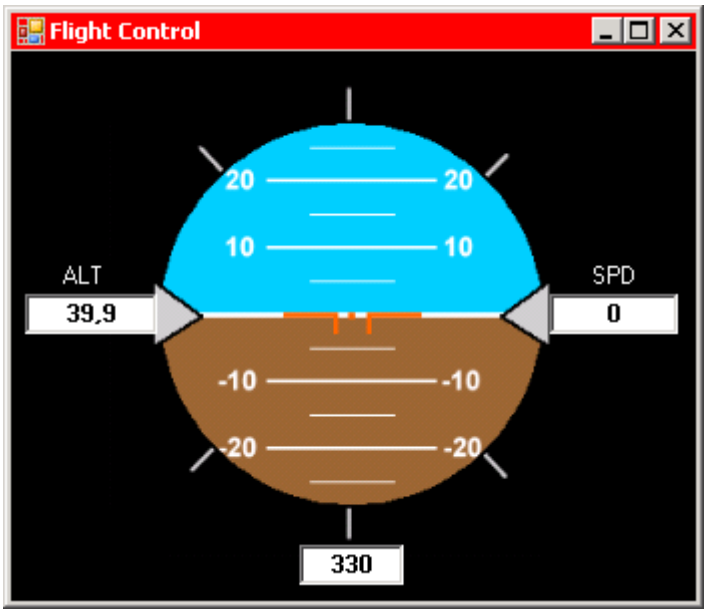


KUVA 7. GPS-vastaanottimen eriteltyä dataa

5.2.3 IMU

Inertial Monitoring Unit näyttää käyttäjälle reaaliaikaista dataa koneen asennosta, kiihtyvyyksistä, nopeudesta ja suunnasta. Ikkuna sisältää myös graafisen asentoindikaattorin, joka helpottaa ja nopeuttaa koneen ohjaamista ja mahdollisten tarvittavien korjausliikkeiden tekemistä.

IMU saa kiihtyvyys- ja asentodatansa lentokoneessa sijaitsevilta gyroskoopeilta ja kiihtyvyysantureilta. Nopeus-, suuntima- ja korkeustiedot puolestaan tulevat GPS-vastaanottimelta. Tämä data on reaaliaikaista tietoa sensoreilta, eikä koneen ohjaaja voi siihen vaikuttaa.

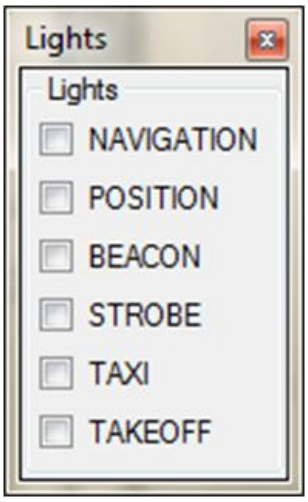


KUVA 8. Asentoindikaattori

Yllä (kuva 8) on asentoindikaattorin näyttämä kuva paikoillaan olevasta koneesta. Korkeus (kuvassa ALT) on korkeus merenpinnasta, nopeus (SPD) nopeus kilometreinä tunnissa maanpinnan suhteen ja alhaalla suunta, johon koneen keula osoittaa. Koneen ollessa tasaisella alustalla on keinohorisonttikin neutraalissa asennossa.

5.2.4 LIGHTS

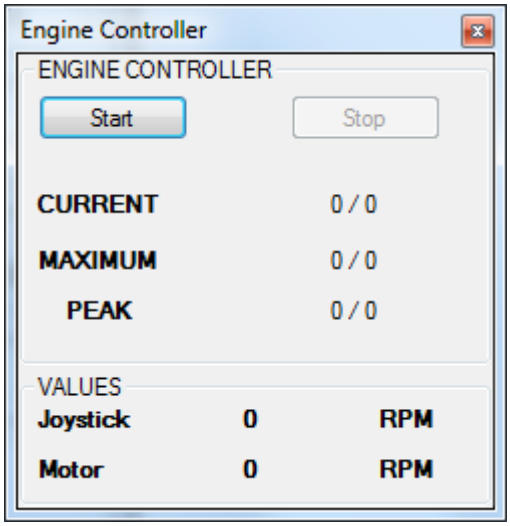
LIGHTS on ikkuna lentokoneessa olevien valojen statuksen tarkistamiseen ja niiden ohjaamiseen (kuva 9). Käyttäjä voi hallita valoja joko tämän ikkunan tai joystickin avulla. Käyttöliittymäikkunasta kuitenkin näkee nopeasti, mitkä valot ovat toiminnassa ja mitkä eivät. Kunkin valon kytkeminen aloittaa ennalta ohjelmoidun sekvenssin, joka vilkuttaa kutakin valoa kansainvälisten asetusten määräämällä tavalla, jolloin muut taivaalla liikkujat huomaavat koneen ja voivat päätellä sen suunnan ja aikeet.



KUVA 9. Valojen ohjaus.

5.2.5 ENGCTRL

ENGCTRL sisältää dataa moottorin tilasta ja sen kierrosnopeudesta. Lisäksi ikkuna näyttää joystickin moottoria ohjaavan säätimen asennon, jolloin nopeasti nähdään onko tiedonsiirto koneen ja maa-aseman välillä ajantasaista (kuva 10).



KUVA 10. Moottorin ohjaus.

Koska moottorin testaaminen olisi luonnollisesti vaatinut paljon tilaa, ei sitä työtä tehdessä ollut asennettu koneeseen.

Seuraavat koodiesimerkit suoritetaan COB:ssä maa-aseman käyttäjän toimenpiteiden mukaisesti.

Start-nappia painettaessa suoritetaan alla oleva koodi. Koska tiedetään, ettei koneeseen ole kytketty kuin yksi moottori, valitaan kytkettyjen laitteiden listasta ensimmäinen. Instanssi otetaan talteen paikalliseen muuttujaan, jota käyttäen moottoria voidaan ohjata.

```
Smc connectToDevice()
{
    List<DeviceListItem> connectedDevices = Smc.getConnectedDevices();

    foreach(DeviceListItem dli in connectedDevices)
    {
        return new Smc(dli);
    }
    throw new Exception("Could not find device.“);
}
```

Stop-nappia painettaessa suoritetaan puolestaan seuraavanlainen koodi. *_device* on muuttuja, joka sisältää laitteen instanssin aiemmasta esimerkistä.

```
private void btnStop_Click(object sender, EventArgs e)
{
    try
    {
        _device.stop();
    }
    catch (Exception exception)
    {
        showException(exception);
    }
}
```

Varsinainen moottorin nopeuden säätäminen tapahtuu alla olevan koodin mukaisesti.

```
private void adjustMotorSpeed(int val)
{
    try
    {
        _device.setSpeed(val);
    }
    catch (Exception exception)
    {
        showException(exception);
    }
}
```

Funktiot *stop()* ja *setSpeed()* ovat Pololun SDK:n tarjoamia rajapintoja laitteiston ohjaamiseen. Kuten yllä olevista esimerkeistä nähdään, voi kehitystyö välillä olla hyvinkin suoraviivaista.

5.2.6 PDU

PDU eli Power Distribution Unit on komponentti lentokoneen virrankulutuksen tarkkailuun. Koneessa on akkuja, joiden lataustaso tiedetään. Tämän tiedon ja virrankulutuksen perusteella ohjelmisto laskee arvioidun jäljellä olevan lentoajan, varoittaa virrankulutuksen yllättävistä piikeistä ja tiedottaa niistä käyttäjälle.

PDU:n esittämä data sisältää mittaustuloksia ja niiden perusteella tehtyjä laskelmia, eikä koneen ohjaaja voi niihin näin ollen vaikuttaa.

5.3 Lokitiedostot

Kuten COB, myös GS ylläpitää kahta lokitiedostoa: toinen laitteistoa ja toinen verkkoliikennettä tarkkaillen. COB:stä poiketen maa-aseman laitteistoloki on paljon rajoittuneempi, koska maa-asemalla ei ole suoraa yhteyttä varsinaiseen laitteistoon. Myös verkkoliikennettä tarkkaileva loki keskittyy maa-aseman eli asiakkaan tapahtumiin ja tallentaa palvelimen vastaukset annettuihin komentoihin.

Yhdessä näitä lokeja tarkastelemalla saa hyvän käsityksen siitä, mitä milloinkin on tapahtunut tai mitä olisi pitänyt tapahtua. Näin mahdolliset virhetilanteet ja/tai ei-toivottu toiminnallisuus voidaan huomata ja asiaan puuttua tai vaihtoehtoisesti jatkokehityksessä ottaa huomioon poikkeustilanteet, joihin aiemmin ei ehkä olla osattu varautua.

6 MAHDOLLISET LISÄOMINAISUUDET

Aikataulullisista syistä aivan kaikkia suunniteltuja ominaisuuksia ei ohjelmistoon ehditty toteuttamaan, vaan työssä jouduttiin keskittymään tärkeimpien hallintalaitteiden ja -mekanismien toimintakuntoon saattamiseen.

6.1 Kartta- ja navigointijärjestelmä

Nykyisin eri palveluntarjoajilta on saatavilla useita eri vaihtoehtoja karttasovelluksen lisäämiseksi nyt toteutettuun ohjelmistoon. Tällaisen karttasovelluksen avulla voitaisin reaaliaikaisesti seurata koneen sijaintia kartalla, automaatiota ohjelmistoon lisäämällä määritellä reittipisteitä, joiden kautta koneen halutaan lentävän sekä kunkin lennon jälkeen tarkastella kuljettua reittiä graafisesti sähköisen kartan avulla.

6.2 Autopilotti

Ajatus autopilotista liittyy osittain luvun 6.1 ajatukseen ennalta määrätystä navigoinnista, mutta pitää sisällään muutakin. Autopilotin avulla kone voitaisiin asettaa lentämään suoraan, ympyrää tai muuta ennalta määriteltyä kuviota ilman karttapisteiden erillistä määrittelyä. Tämä voisi olla hyödyllinen ominaisuus esimerkiksi silloin, kun yhteys syystä tai toisesta katkeaa tai kun koneen ohjaaja ei pysty konetta hetkeen ohjaamaan.

7 POHDINTA

Työn tekeminen oli monella tapaa varsin opettavainen kokemus. Toisinaan työn eteneminen oli hyvinkin suoraviivaista, kuten luvun 5.2.5 koodiesimerkeistä käy ilmi, kun taas toisinaan työssä edettiin ainoastaan yrityksen ja erehdyksen kautta. Hyvän matkaa työn alkuvaiheista menikin tunnustellen ja tutkien laitteiston ja sen rajapintojen toimintaa, mikä myöhemmissä vaiheissa saattoi asettaa omia rajoitteitaan tehtyjen valintojen vuoksi.

Huolellisen suunnittelun merkitys korostui työtä tehdessä useasti: tämänkin kokoista ohjelmistoa rakennettaessa täytyy muutoksia tehdessä muistaa ja osata ottaa huomioon kaikki muutoksista aiheutuvat kerrannaisvaikutukset sekä miettiä, kuinka ne tulevaisuudessa rajoittavat tai vaikeuttavat jatkokehitystä.

Kolmansien osapuolten tuottamien ominaisuuksien sijoittaminen oman ohjelmakoodin seuraksi oli mielenkiintoista. Sen sijaan että tarjolla olisi ollut valmiit luokkakirjastot ja rajapinnat, jouduin itse toteuttamaan nämä siten, että lopputulos toimi siististi.

Ohjelmiston testaaminen tarjosi niin ikään omat haasteensa. Kokonaisuuden jakautuessa kahteen erilliseen osaan oli virhetilanteiden havaitseminen ja niiden paikallistaminen välillä varsin haastavaa. Joissain tapauksissa virheet olivat niin pieniä ja huomaamattomia, ettei niistä testatessa olisi ollut mitään haittaa, mutta lentokoneen ilmassa ollessa olisi näillä tapahtumilla voinut olla tuhoisia seurauksia. Testivetoinen kehitys eli testien kirjoittaminen etukäteen voisi hyvinkin olla ratkaisu vastaavan kokoisten tai suurempien ohjelmistojen kirjoittamiseen. Virheiden varhaisen löytymisen lisäksi varsinaista suunnittelua ja toiminnallisuuden tarkkaa dokumentointia tulisi varmastikin tehtyä enemmän ja huolellisemmin.

Suurin haaste työn tekemisessä oli ehdottomasti maa-aseman ja lentokoneen välisen keskustelun toteuttaminen. Data piti saada liikkumaan niin reaaliaikaisesti kuin mahdollista ja maa-asemalla lentokonetta ohjaavan henkilön piti pystyä olemaan varma, että näytöllä näkyvät tiedot ovat ajantasaisia ja vastaavat todellisuutta. Tässä mielessä tavallisen, suoraan lentokoneesta käsin ohjattavan järjestelmän toteuttaminen olisi voinut olla helpompi toteuttaa.

Jälkiviisaana olisi helppo sanoa, että jos aloittaisin työn nyt uudestaan, suunnittelisin kaiken etukäteen ja toteuttaisin ohjelmiston paljon määrätietoisemmin. Ajatuksen tasolla tämä onkin totta, mutta todellisuudessa laitteiston ohjelmointirajapintojen ja toiminnan opetteleminen vei sen verran aikaa, että kaiken etukäteen suunnitteleminen olisi ollut todella hankalaa, ellei täysin mahdotonta.

LÄHTEET

.NET Framework, Wikipedia [verkkosivu]. [viitattu 16.5.2012]. Saatavissa: http://fi.wikipedia.org/wiki/.NET_Framework

C#, Wikipedia [verkkosivu]. [viitattu 16.5.2012]. Saatavissa: http://fi.wikipedia.org/wiki/C_sharp

DirectX, Wikipedia [verkkosivu]. [viitattu 18.4.2012]. Saatavissa: http://en.wikipedia.org/wiki/DirectX.

Ecma International. *C# Language Specification*. [viitattu 18.4.2012]. Saatavissa: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf.

Microsoft. *.NET Framework 4* [verkkosivu]. [viitattu 18.4.2012]. Saatavissa: http://msdn.microsoft.com/en-us/library/w0x726c2.aspx.

Microsoft Visual Studio, Wikipedia [verkkosivu]. [viitattu 14.5.2012]. Saatavissa: http://en.wikipedia.org/wiki/Microsoft_Visual_Studio.

Pololu Robotics and Electronics. [verkkosivu]. [viitattu 18.4.2012]. Saatavissa: http://www.pololu.com/.

Carsinogen

Project plan

version 3.0, 27.5.2012

Jani Taskinen

VERSION	DESCRIPTION	DATE
0.1	***Document created and updated***	***9.2.2012***
0.2	***Document updated***	***10.2.2012***
0.3	***Document updated***	***12.2.2012***
0.4	***UI updated*** ***Functionality & error checking in case of disconnection updated*** ***Obsolete functionality from COB removed*** ***Document updated***	***15.2.2012***
0.5	***Functionality & thread safety @COB & @GS improved***	***17.2.2012***
0.6	***Document name changed*** ***Document updated***	***24.2.2012***
1.0	***Document approved***	***7.3.2012***
1.1	***Watchdog application added to monitor Carsinogen process*** ***IMU integrated to COB*** ***Pictures of UI added*** ***Document updated***	***16.3.2012***
1.2	***Joystick control functional*** ***Software reliability and recovery enhanced*** ***Second thesis supervisor added*** ***Timeline adjusted***	***30.3.2012***
2.0	***Document approved***	***2.4.2012***
2.1	***Document updated***	***25.5.2012***
3.0	***Document approved***	***27.5.2012***

CONTENTS

1. INTRODUCTION

1.1. Purpose of Plan

This document provides a definition of the Project Carsinogen, including the goals and objectives of the project from the aspect of the software development involved. As the project also acts as the signatory's thesis, aspects and timelines relevant are also covered.

1.2. The Product

Project Carsinogen is a proof of concept of a cost-effective and competitive unity providing software, user interface and equipment of a remote controlled aircraft possibly to be commercialized later on. The scope of potential clients of the outcome ranges from private customers to governmental players. This document covers the definition, goals and objectives concerning the software.

1.3. Administration of the Document

This document will be reviewed and updated on a somewhat weekly basis by those working on the project.

1.4. Definitions and Abbreviations

In this document, term customer refers to NCOW Technology Ltd and its representatives, whereas developer refers to Mr. Jani Taskinen. Term thesis supervisor refers to Mr. Sami Lahti.
Other definitions and abbreviations concerning the software are documented in Software Requirement Specification.

2. PROJECT ORGANIZATION

2.1. Project Phasing

The project will be carried out non-stop. Documentation, development and testing will be performed throughout the project.

2.2. Persons in Charge

Responsible for the software development is the developer. The customer is responsible for providing hardware and proper testing equipment. Mr. Sami Lahti and Mr. Jussi Koistinen from Savonia UAS act as supervisors for the thesis part of the project.

Table 1. Persons involved

Name	Role	E-mail	Phone number
Mikko Jakonen	**Customer**	**mikkoj@secproof.com**	**040 5720 475**
Jani Taskinen	**Developer**	**jani.taskinen@edu.savonia.fi**	**045 1141 811**
Sami Lahti	**Thesis supervisor**	**sami.lahti@savonia.fi**	**044 785 6337**
Jussi Koistinen	**Thesis supervisor**	**jussi.koistinen@savonia.fi**	**044 785 5512**

3. PROJECT MANAGEMENT

3.1. Goals and Prioritization

The project is to be completed in the course of spring 2012. By the time of completion, both software and the documentation will be finalized. Thesis report shall be written once the project is finished.

3.2. Assumptions, Constraints and Boundary Conditions

The completion of the software for the project is dependent on the prevailing state of the hardware. As a boundary condition, the project must be carried through in the course of spring 2012 (Table 3. Milestones).

3.3. Risk Assessment

Persons involved in the software development are committed to complete the project over the spring 2012. Any adversities will be reckoned with consultation between the developer and the customer to mitigate any detriment to the software development. The software will be tested throughout the project by both the developer and the customer.
Table 2 lists possible risks hindering and/or postponing development of the software, actions to be taken to mitigate the problem, and probability of the scenario.

Table 2. Risk assessment

Risk	Remedial measures to be taken	Probability (1 – 5)
Unrealistic schedule	Ignore inessential and/or futile features	3
Erroneous functionality or futile features	Active dialogue between the developer and the customer	4

3.4. Supervision and Counseling

Persons involved in the project will plan and review the development of the project on somewhat weekly basis. Updates to the software will be reviewed by the customer, and any flaws, enhancement propositions and other feedback will be reported to the developer. All updates and changes to the software and/or documents will be reported to the thesis supervisor to be reviewed and approved.

4. TECHNOLOGY

4.1. Methods and Tools

The development of the software will be carried out using Visual Studio 2010 IDE. The software code will be written in C# programming language using .NET Framework 4.0. Additional software libraries and higher-level hardware interfaces will be provided by hardware vendor Pololu. The software is designed to be used in Windows OS environment.

4.2. Documentation

Entire documentation for the software will be found in the Project Plan and the Software Requirement Specification. Thesis report to be written later on will cover the thesis part of the project.

4.3. Quality Control

The quality of the software is ensured by testing, which will be carried out throughout the project by both the developer and the customer.

5. TIMELINE AND MILESTONES

Table 3 describes the timeline and major milestones of the project. These can be used to monitor the progress of the project to some extent. The dates given can be considered advisory though, as they are subject to change due to e.g. priority changes. Thesis seminar date, however, is to be taken as a boundary condition date to finalize the project, since it is not dependent of persons involved in the project.

Table 3. Milestones

Task	Due date	Date complete
Client-server connection operational Flight control System operational Power distribution unit operational Lights controlling operational Servo controlling operational	***2012***	***2/2012***
UI final version	***4/2012***	***5/2012***
IMU monitoring complete	***2.3.2012***	***16.3.2012***
PDU monitoring complete	***4/2012***	***4/2012***
GPS monitoring complete	***4/2012***	***5/2012***
Engine control complete	***4/2012***	***5/2012***
Joystick control complete	***10.4.2012***	***5/2012***
Software complete	***20.4.2012***	***5/2012***
Documentation complete	***20.4.2012***	***5/2012***
Thesis report finished	***20.4.2012***	***5/2012***
Thesis seminar	***27.4.2012***	***27.4.2012***

Carsinogen

Software Requirements Specification

version 4.0, 28.5.2012

Jani Taskinen

Version history

VERSION	DESCRIPTION	DATE
0.1	*Servo controlling operational* *Flight Control System operational* *Client-server connection operational* *Power Distribution Unit operational* *Lights Controlling operational* *Inertial Measurement Unit in testing*	*Fall 2011 – February 2012*
0.2	*Document created*	*9.2.2012*
0.3	*Document updated*	*10.2.2012*
0.3	*Document updated*	*12.2.2012*
0.4	*UI updated* *Functionality & error checking in case of disconnection updated* *Obsolete functionality from COB removed* *Document chapter 5 updated*	*15.2.2012*
0.5	*Functionality & thread safety @COB & @GS improved*	*17.2.2012*
0.6	*Document name changed* *Document updated*	*24.2.2012*
1.0	*Document approved*	*7.3.2012*
1.1	*Watchdog application added to monitor Carsinogen process* *IMU integrated to COB* *Pictures of UI added* *Document updated*	*16.3.2012*
1.2	*Joystick control functional* *Software reliability and recovery enhanced*	*30.3.2012*
1.3	*Document updated*	*11.4.2012*
2.0	*Document approved*	*18.4.2012*
2.1	*Document updated*	*25.5.2012*
3.0	*Document approved*	*27.5.2012*
3.1	*Document updated*	*28.5.2012*
4.0	*Document approved*	*28.5.2012*

1 INTRODUCTION

1.1 Purpose

This document defines the functionality and features of the software of Project Carsinogen. Document also includes general description of the software, including definition of the software's purpose and definitions and abbreviations associated.

1.2 The Product

Project Carsinogen is a proof of concept of a cost-effective and competitive unity, built from commercial off-the-shelf parts, providing software, user interface and hardware equipment of a remote controlled aircraft. This document covers the software development aspect of the project.

1.3 Definitions and Abbreviations

In this document, software affiliated with and used to control the hardware included in Project Carsinogen is simply referred as software. In some cases, both Main window and Avionics window may be referred only as Main Window.
Table 1 Explains abbreviations and acronyms used in this document.

Table 1. Abbreviations

COB	Computer On Board
ENGCTRL	Engine Controlling
FCL	Flight Control
FCS	Flight Control System
GPS	Global Positioning System
GS	Ground Station
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
OS	Operating System
PDU	Power Distribution Unit
TCP/IP	Transmission Control Protocol/Internet Protocol
UI	User Interface
WLAN	Wireless Local Area Network

1.4 Overview to the Document

First part of the document (Introduction) comprises of short description of the product. It also lists the definitions and abbreviations found in the document.

Second part of the document consists of brief technical overview of the software, including used technologies, summary of the functionality of the software, and general restrictions and constraints.

Third part summarizes data handling and storage.

Fourth part offers insight to the software's functionality and features, including use cases and descriptions of the functions involved.

Fifth part describes user interface in more detail, including screenshots of various windows, and explanations what happens under the hood in each case.

2 OVERVIEW

2.1 Programming and Usage Environment and Technology

The software for Project Carsinogen is developed using Visual Studio 2010 IDE, with code written in C# programming language with .NET Framework 4.0. Additional libraries providing higher-level interfaces from hardware vendor Pololu are also used. Parts of C# Avionic Instrument Controls as well as NMEA 0183 2.0 Sentence parser are used under the Code Project Open License. The software is designed to be used in Windows OS environment.

2.2 Functionality

The software comprises of two parts, Ground Station (GS) and Computer-On-Board (COB). GS offers User Interface (UI) to the end-user to control and monitor the behavior of the aircraft and the hardware installed in it. COB controls and monitors the actual hardware installed to the aircraft, according to the actions of the user on GS. Communication between the GS and the COB is realized through a client-server solution over TCP/IP network.

This document focuses mainly on the functionality of GS, as the functionality is pretty much designed to be as if the hardware was operated on a local machine. On relevant parts, should the COB's behavior differ from the GS's, the case will be addressed.

2.3 General Restrictions and Constraints

The software requires Microsoft Windows 98 or later OS to be executed. The software itself is quite light-weight, so it causes no major consideration concerning performance on modern machines.

.NET Framework 4.0, as well as hardware vendor Pololu's USB drivers are required to be installed on the target machine, as the software utilizes .NET Framework's class libraries, and the servos and other hardware on the aircraft are controlled by the drivers provided by Pololu.

Communication between GS and COB is realized through a client-server solution, working over TCP/IP, so the computer Carsinogen is to be used on has to be provided with a TCP/IP-capable network medium.

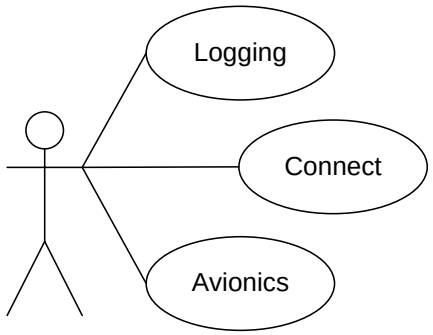
3 DATA STORAGE AND HANDLING

The software uses local disk space mainly for configuration loading and saving and for log writing. Disk space used is therefore quite minimal. As the software requires no installation whatsoever, it is possible to use the software even from removable drives, making the application directly transportable.

4 FUNCTIONALITY AND USAGE

4.1 Main Window

Error! Reference source not found. describes the main functions of the GS, which are addressed more in-depth further in the document.



Use Case 1. Main functions of the GS

4.1.1 Logging

At the moment, the software keeps 4 different logs, each saved to their own file. These log files are COB side log of the actions and responses concerning the hardware, COB side log of the network operations, GS side log of interactions concerning the hardware and finally, GS side log of the network. Though the GS doesn't directly get involved in controlling the hardware, but is merely a UI to make the operations user friendly, it keeps track of the operations initiated by the user, and should any malfunction occur, comparing the data from the GS and the COB hardware logs may help to track where the problem first arose.

4.1.1.1 Watchdog application

To ensure that the software runs smoothly, and to mitigate any problems due to crashes or unwanted behavior during flight, the software is accompanied by a "watchdog" application, which monitors the health of the Carsinogen process in the COB. In practice, the software updates a log file at certain intervals just to tell the guardian application it is still alive and functioning correctly. In case the file doesn't get updated regularly, or Carsinogen process isn't found, the watchdog does one of the following:

- If the Carsinogen process ceases to exist, the application tries to restart the process, thus enabling the GS to reconnect to the COB and operating the aircraft may resume.
- If the file doesn't get updated, but the Carsinogen process exists, it is most likely the program is unable to function correctly. The application then kills the Carsinogen process, after which it is restarted and normal functionality hopefully restored.

4.1.1.2 Temporary log file(s)

During operations, the COB writes a temporary file to store the latest positions for servos in case the client disconnects during flight. Once reconnected after unexpected connection failure, the COB checks if this data file is found and passes it to the GS, thus keeping the data synchronized on both sides.

This file is created when the servos' positions are changed for the first time, and deleted (only) in case the software exits normally.

4.1.2 Connect

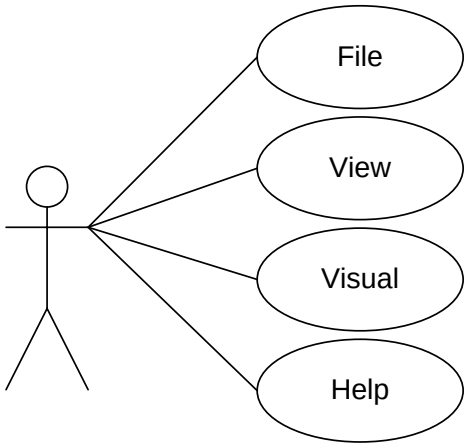
After clicking 'Connect', the software tries to establish a connection to the COB according to the configuration saved in client.

Table 2. Connect

Name:	Connect
Executor:	User
Prerequisites:	COB-side program of software is running
Description:	The software establishes connection to the COB
Exceptions:	COB is not running, network malfunction or hardware failure. Error message is shown.
Outcome:	Establishes connection to COB enabling further operations

4.1.3 Main Window Toolbar menu

From the toolbar menu, user can select various options.



Use Case 2. Main Window Menus

4.1.3.1 File

From the ‘File’ drop-down dialogue, user can exit the program.

4.1.3.2 View

The ‘View’ dialogue features different view windows to be selected.

4.1.3.3 Visual

‘Visual’ dialogue contains tools to enhance user experience, such as invert window colors and save positions for windows

4.1.3.4 Help

‘Help’ dialogue offers information and support concerning the software.

4.2 Main Window, Connected to COB

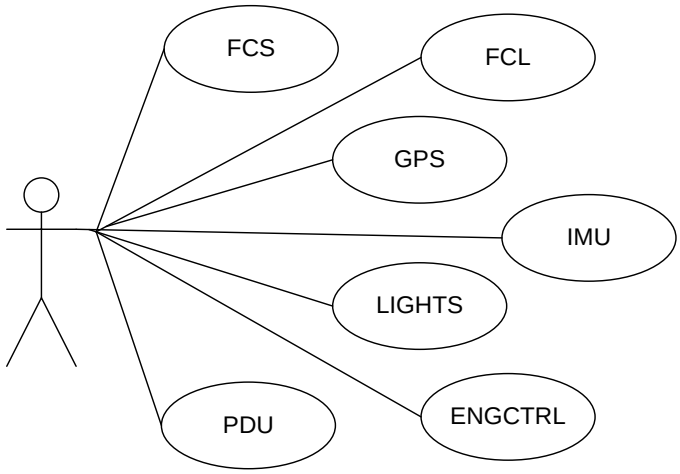
After establishing a connection to COB, thus ensuring the connection works, user can connect the COB to the actual hardware on the aircraft. Table 3 describes the process to enable the hardware and assume control of the aircraft.

Table 3. Enable Hardware

Name:	Enable
Executor:	User
Prerequisites:	Connection to COB has been established and is working
Description:	COB tries to enable installed hardware, enabling controlling and monitoring of the aircraft
Exceptions:	Hardware cannot be enabled. Error message follows. Most likely due to hardware or network malfunction; software cannot recover.
Outcome:	Hardware on the aircraft is enabled and controllable by GS.

4.3 Avionics

Prerequisites to accessing Avionics window are a working connection to COB and COB's access to the hardware on the aircraft. Once these are established, user can access all the control and monitoring components of GS.



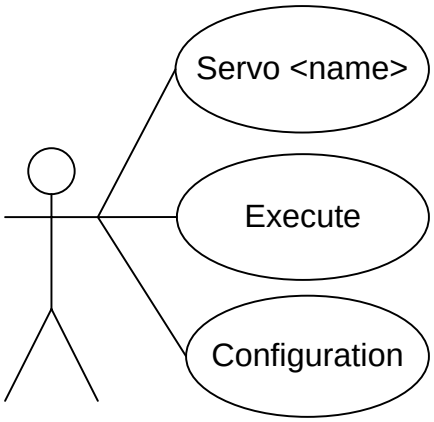
Use Case 3. Avionics Window

Table 4. Avionics

Name:	Avionics
Executor:	User
Prerequisites:	Connection to COB has been established and hardware on board is enabled.
Description:	User clicks on component desired to be viewed.
Exceptions:	Hardware fails to respond or is otherwise unreachable. Due to hardware of network malfunction, software cannot recover.
Outcome:	Component window opens.

4.3.1 FCS

Flight Control System provides means to controlling presets for the servos controlling the aircraft while in-flight.



Use Case 4. FCS

Table 5. Servo <name>

Name:	Servo <name>
Executor:	User
Prerequisites:	Connection to COB has been established and hardware is enabled.
Description:	User adjusts servos.
Exceptions:	None
Outcome:	User sees numerical outcome of adjusting servos' positions. No actual adjustment is done before 'Execute' is executed.

Table 6. Execute

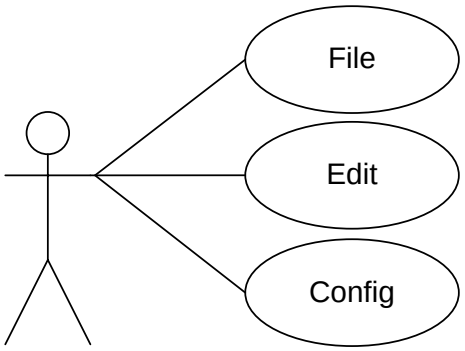
Name:	Execute
Executor:	User
Prerequisites:	Connection to COB has been established and is working.
Description:	User executes servo preset tuning.
Exceptions:	Hardware fails to respond or network malfunction. Error message follows.
Outcome:	Servos are adjusted due to user's input.

Table 7. Configuration

Name:	Configuration
Executor:	User
Prerequisites:	Connection to COB has been established and is working.
Description:	User can save current configuration as a preset or load a preset from a configuration file.
Exceptions:	Configuration file cannot be read or disk cannot be written to. Error message ensues.
Outcome:	User sees the results of loaded preset in servos' values or current configuration is saved to a file. No actual adjustment is

	made before ‘Execute’ is executed.

From the FCS Toolbar Menu, user can select various options to work with.



Use Case 5. FCS Toolbar Menu

4.3.1.1 File

From the ’File’ dialogue, user can exit FCS.

4.3.1.2 Edit

In the ’Edit’ dialogue is an option to reset all the servos to ‘zero’ position.

4.3.1.3 Config

’Config’ dialogue offers options to save and load different configurations.

4.3.2 GPS

GPS shows information about aircraft’s current position, speed and heading.

4.3.3 IMU

Inertial Monitoring Unit shows data from accelerometers, gyros and provides information about the attitude of the aircraft, as well as acceleration information.

4.3.4 ENGCTRL

Engine Control window is used for controlling the engine.

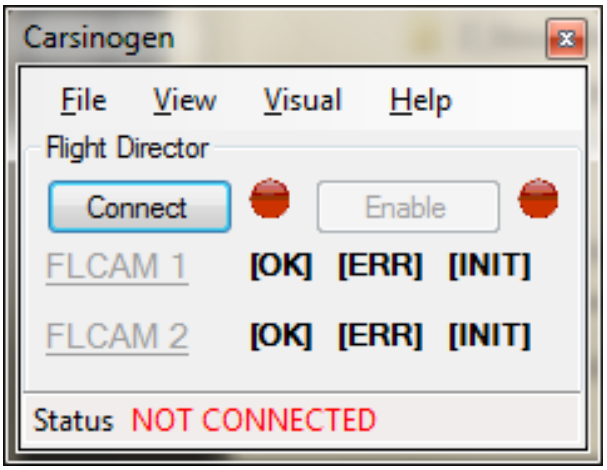
4.3.5 PDU

Power Distribution Unit monitor shows data concerning current power consumption, remaining power in the batteries, estimated flight time left etc. PDU is therefore strictly a monitor window.

5 USER INTERFACE

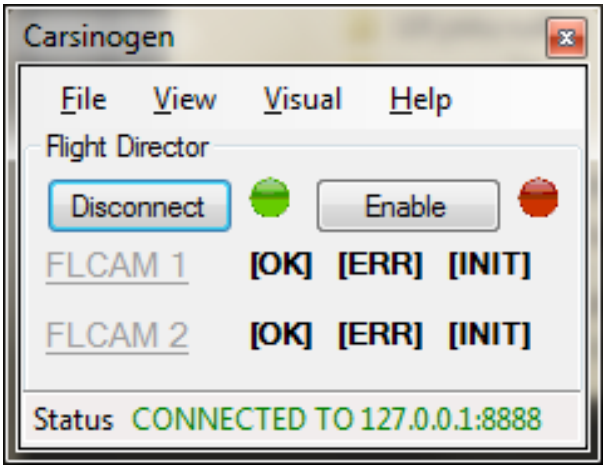
5.1 Main Window

The main window is used mainly for administering the network connection to COB, COB's access to the hardware installed in the aircraft and customizing essential UI features. For operational safety and user experience clarity, most of the functionality is disabled until the connection to both COB and aircraft's hardware are established. As well, while connection to COB can be established at any time, enabling the actual hardware on-board is separated from the network connection to ensure unintentional powering on to avoid damage to the hardware.

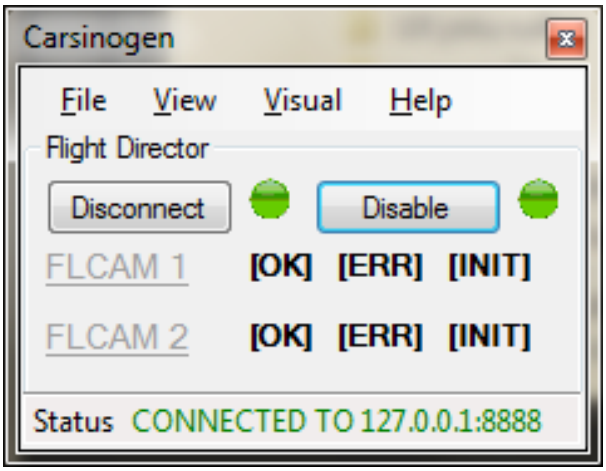


Picture 1. Initial main window

As seen in picture 1, starting the program offers little options as what to do. Picture 2 and Picture 3 depict the steps to establish connection to both COB and hardware, in order to assume control of the aircraft.



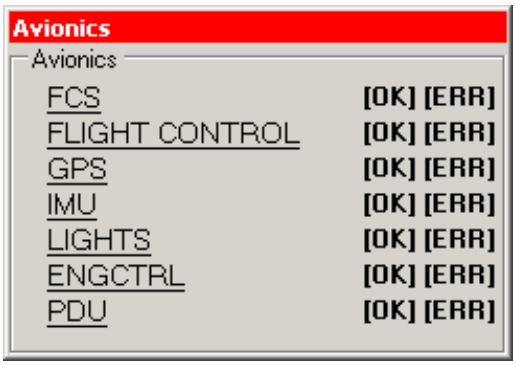
Picture 2. Connection to COB established



Picture 3. Connection to both COB and actual hardware established

5.2 Avionics

Avionics window is a simple window to simplify the UI and to offer quick access to all of the GS's (and hardware in general) controls. For practical and safety reasons, this window cannot be closed unless the whole program exits.

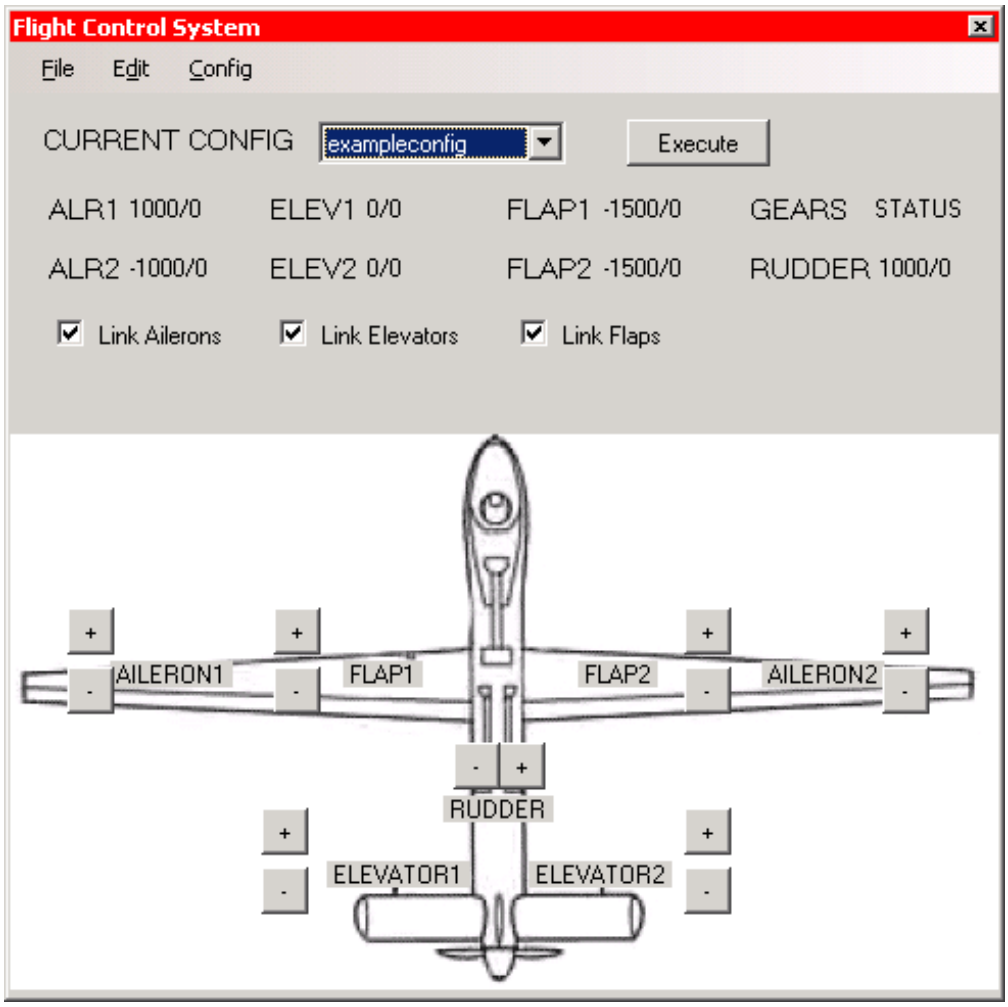


Picture 4. Avionics window

Picture 4 shows the condition where connection to COB is established and the hardware on board is powered on. GPS shows disabled due to the development stage of the software.

5.2.1 FCS

FCS Window is used to adjust servos' preset values to compensate different operating conditions, such as wind, in order to ensure aircraft's optimal responsiveness and maneuverability. Different configurations can be saved to and loaded from disk through this window.

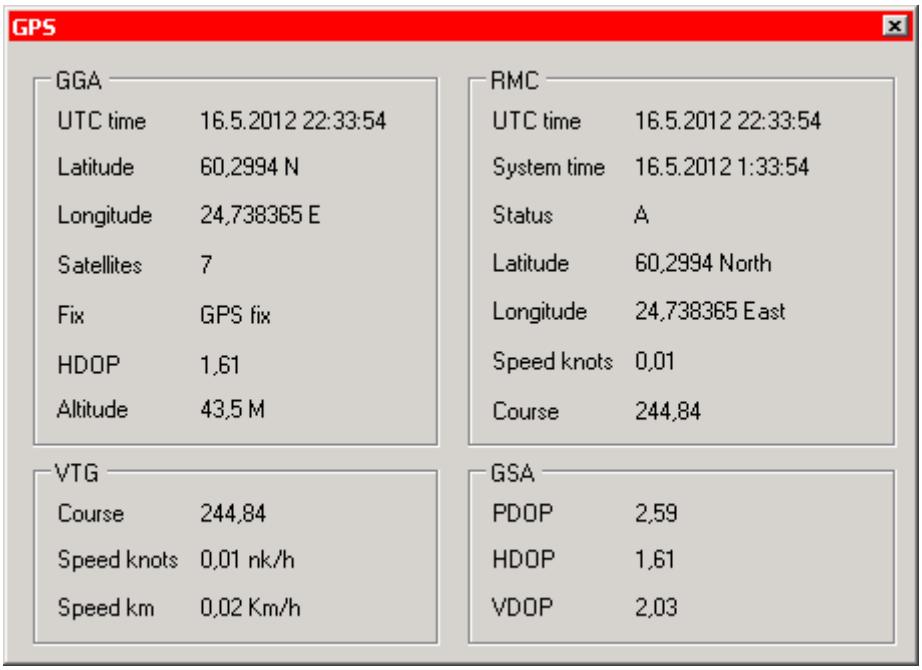


Picture 5. FCS window

Picture 5 shows the FCS window. Values next to abbreviations are ratios to ‘zero’ point of the servos. For each vane, clicking on the name opens up a track bar to easily adjust their respective positions. The plus and minus sign buttons can be used as well. ‘Not Connected’ status is to be replaced with something more appropriate as the project progresses. ‘Current Config’ dropdown box is for quickly loading presets from a file. Upon loading the window, the software automatically retrieves all existing configurations and shows them in the dropdown box.

5.2.2 GPS

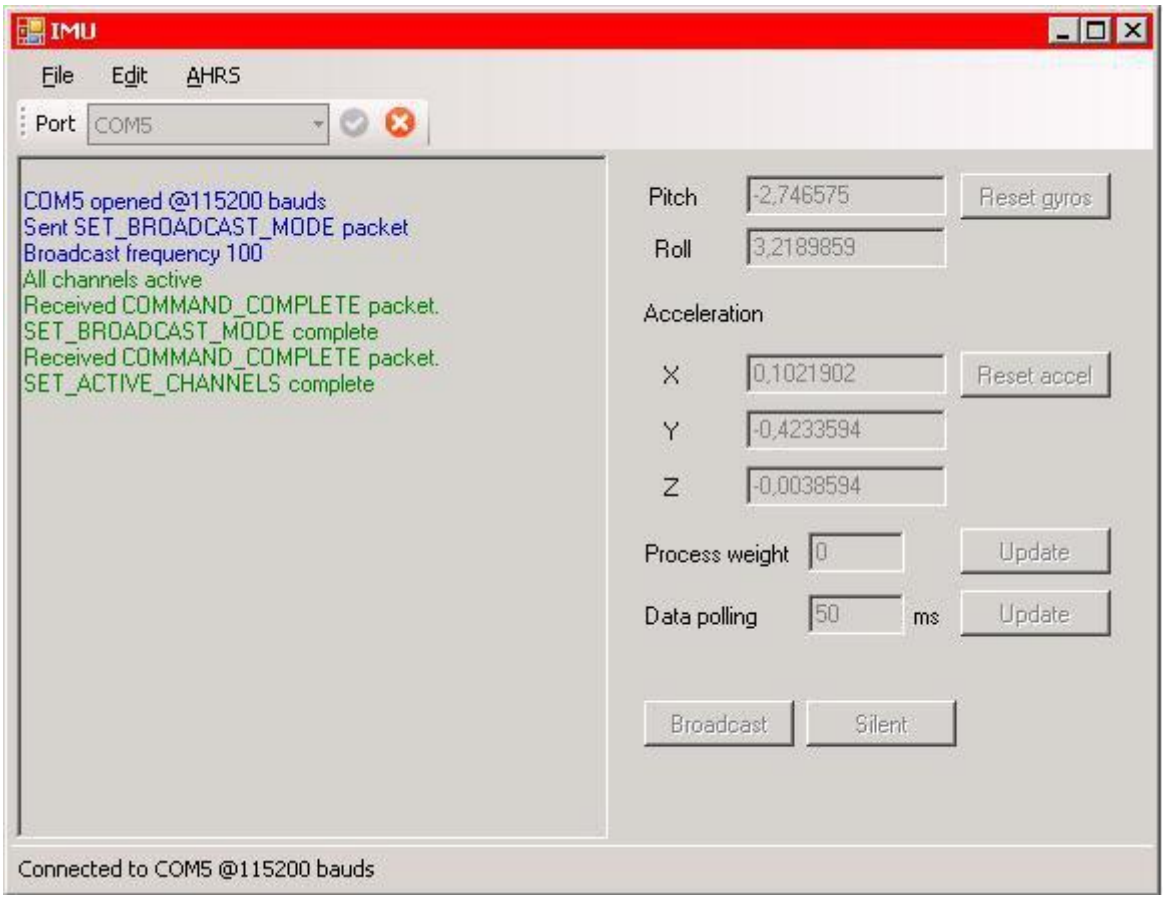
The GPS window shows data from the aircraft's GPS unit, indicating the aircraft's speed, heading and position coordinates. This is strictly a monitor window.



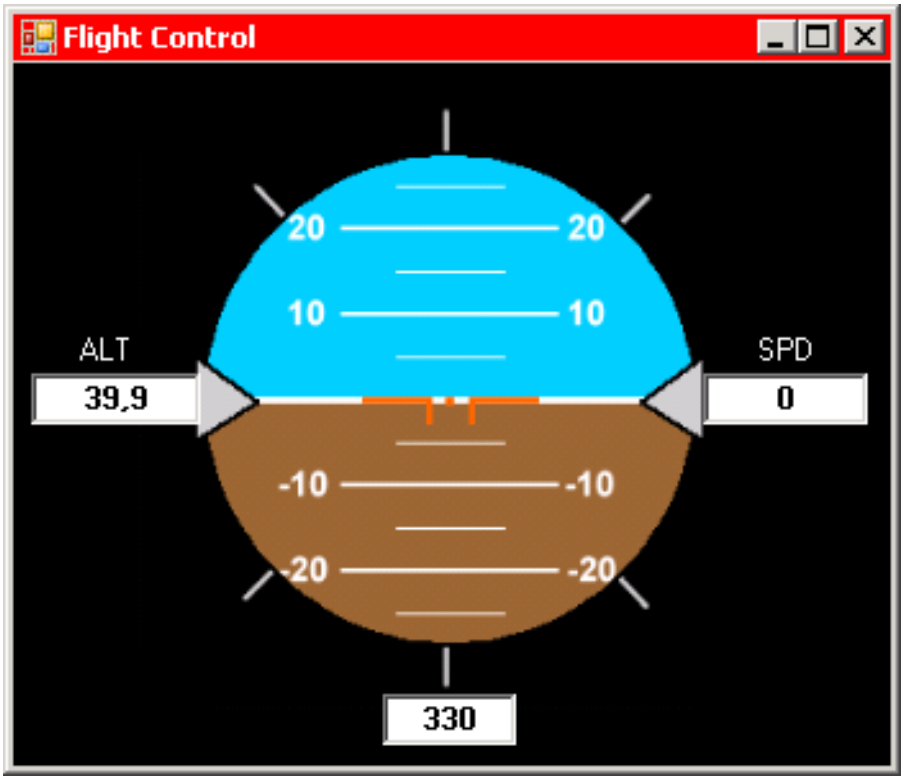
Picture 6. Data from GPS

5.2.3 IMU

Inertial Measurement Unit Window shows data from accelerometers and gyroscopes installed to the aircraft. IMU works in conjunction with an attitude indicator, so user can have a clear perception of the aircraft's situation.



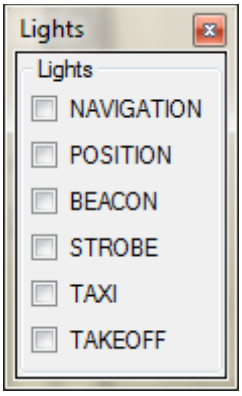
Picture 7. COB side debugging & initiating window for calibration prior takeoff



Picture 8. Attitude window displaying data from IMU

5.2.4 Lights

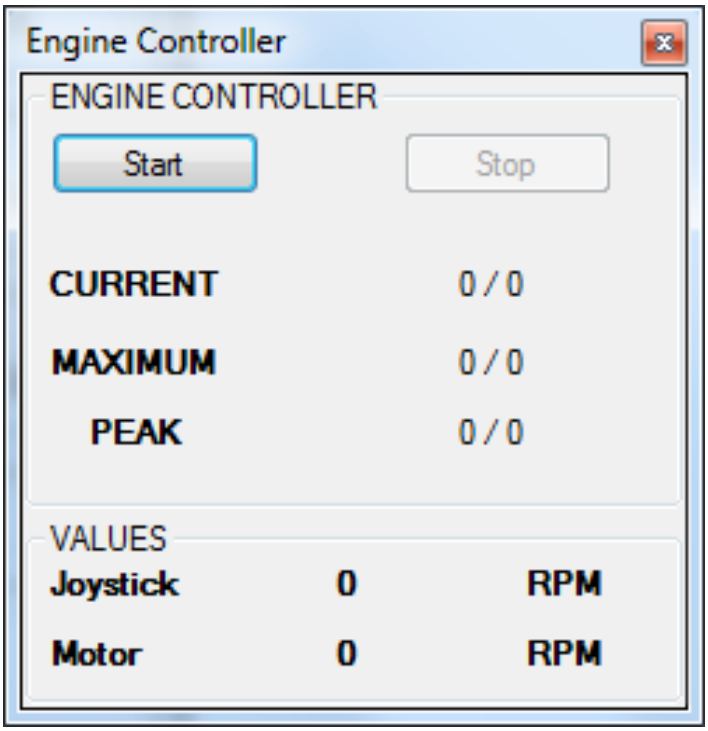
Lights window is an easy-to-use interface to control the lights on board the aircraft. Activating each light starts a predefined sequence flashing the lights according to international aeronautical regulations.



Picture 9. Lights window

5.2.5 ENGCTRL

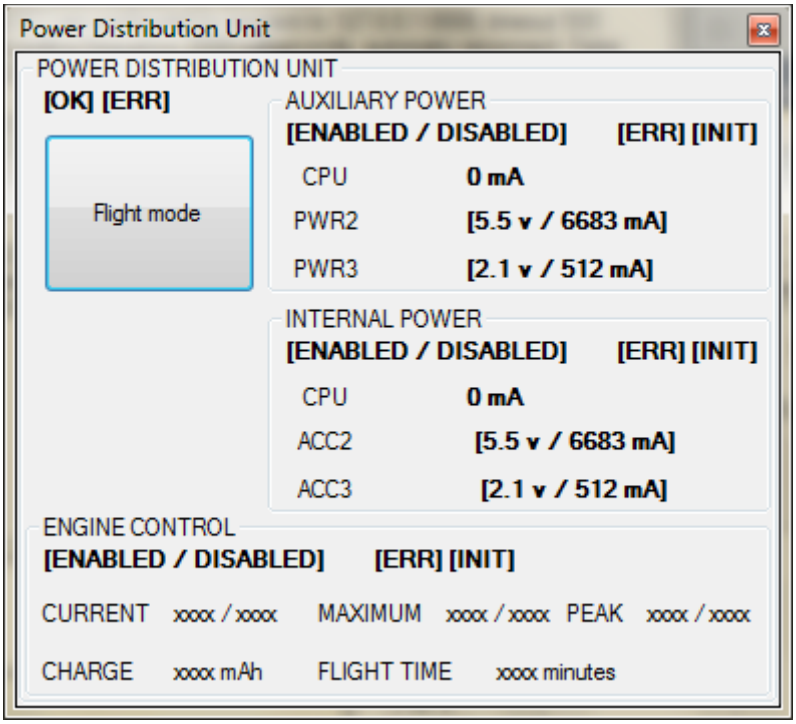
Engine Controller window is used to control the aircraft's engine. Using the UI window or joystick, user can start or stop the motor as well as adjust its speed.



Picture 10. Engine Controller

5.2.6 PDU

Power Distribution Unit window shows data from installed batteries, ampere meters and offers information of battery charge, power consumption and estimated flight time left. PDU has no controls, but is strictly a monitor to data from the aircraft instead.

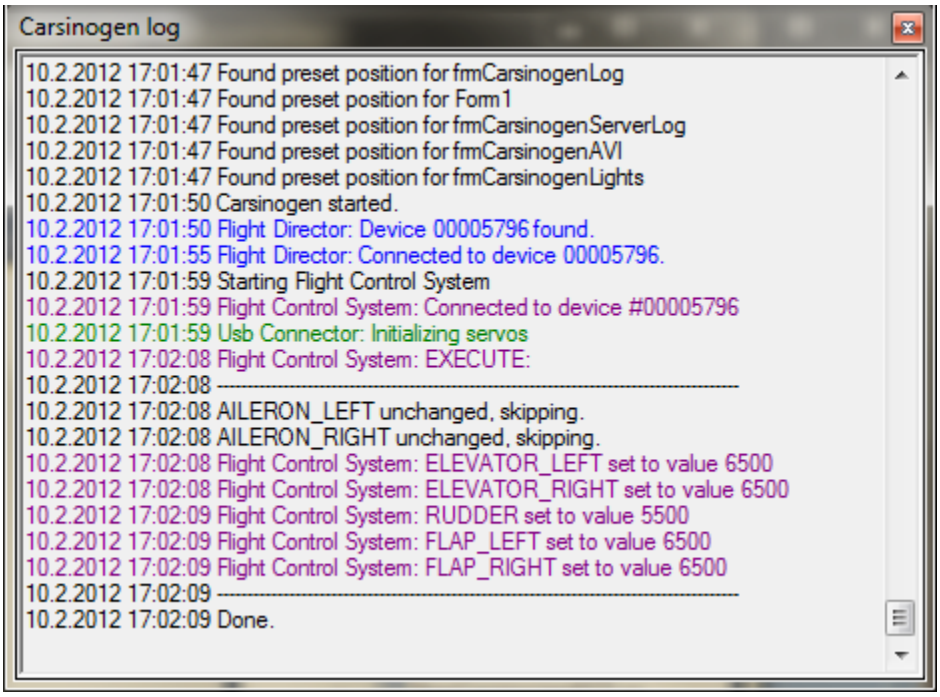


Picture 11. PDU window

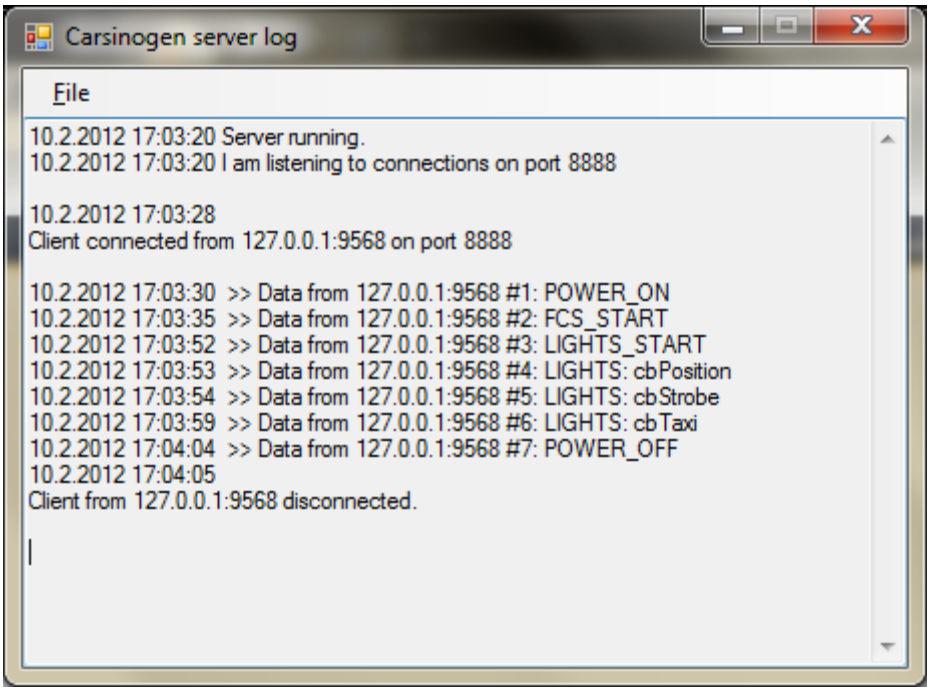
5.3 Logging

Log windows for the software are mostly for keeping the user informed of changes, actions and operations taking place during the use of the software. The GS client log differs here, for its log window also has options for client configuration definition. Client configuration settings are the rules by which client tries to connect to the server on the COB. Naturally, the COB configuration settings must be similar for the connection to be able to be established.

In pictures Picture 12 and Picture 13 are shown the COB side logs written.

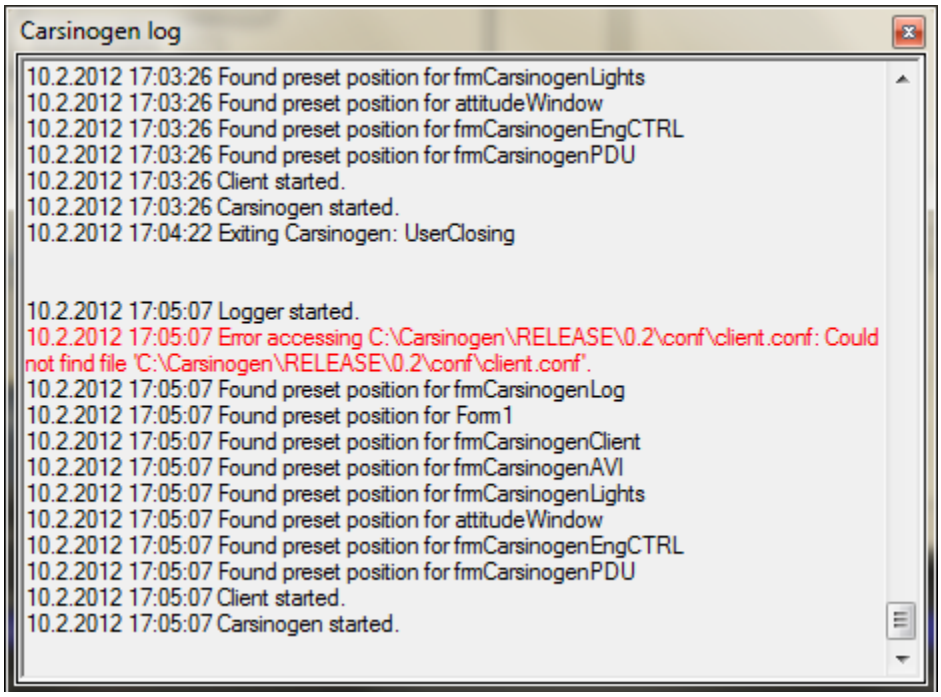


Picture 12. COB side hardware logging

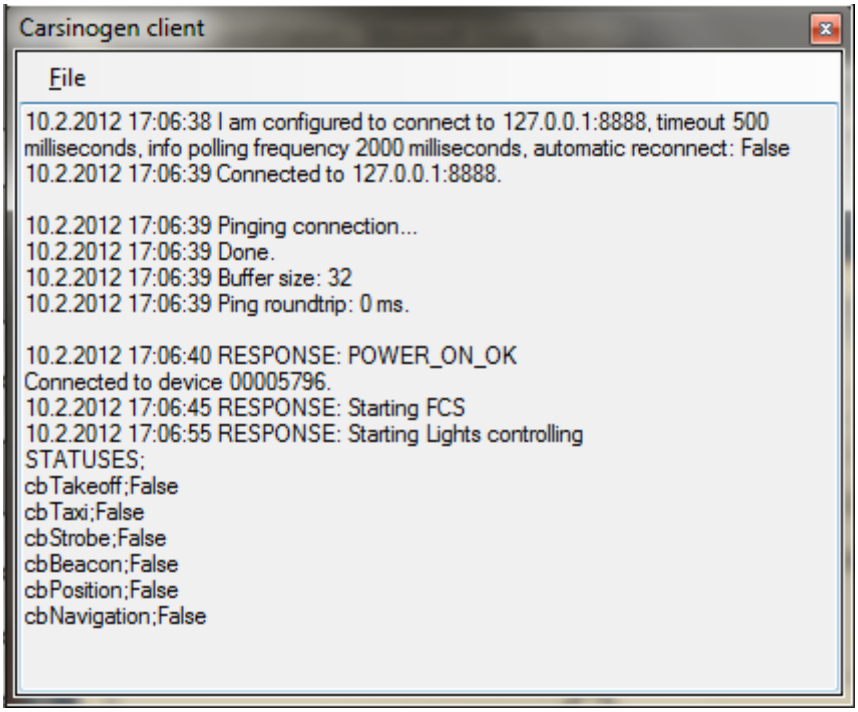


Picture 13. COB side network logging

Pictures Picture 14 and Picture 15 show examples of the logs the GS writes.



Picture 14. GS side hardware logging



Picture 15. GS side network logging with debugging output


SAVONIA