FFI-rapport 2015/01620

Design og implementasjon av FFI GCS – kontrollstasjon for ubemannede systemer

Espen Skjervold

FFI Forsvarets forskningsinstitutt

Design og implementasjon av FFI GCS – kontrollstasjon for ubemannede systemer

Espen Skjervold

Forsvarets forskningsinstitutt (FFI)

26. februar 2016

FFI-rapport 2015/01620

1359

P: ISBN 978-82-464-2690-7
E: ISBN 978-82-464-2691-4

Emneord

Kontrollsystemer

Ubemannede systemer

Autonome undervannsfartøyer

Eksperimentering

Godkjent av

Halvor Bjordal	Prosjektleder
Johnny Bardal	Avdelingssjef

Sammendrag

Formålet med denne rapporten er å dokumentere og gjøre kjent en egenutviklet kontrollstasjon for ubemannede systemer, tatt frem gjennom prosjektet 1359 «Teknologi for fremtidens UAS». Motivasjonen for å utvikle en egen kontrollstasjon er flerdelt: Eksisterende løsninger er ikke gode nok og gir ikke tilstrekkelig støtte til operatøren. Løsningene krever mye personell per farkost som dermed skalerer dårlig og medfører høye kostnader. Ved hjelp av økt automatisering kan operatøren avlastes og kapasitet frigis til andre oppgaver.

Eksisterende løsninger vil også måtte tilpasses for å integreres med sentrale systemer i Forsvarets informasjonsinfrastruktur. På sikt vil det være vesentlig at autonome kapasiteter kan behandles på en homogen måte sammen med andre kapasiteter og håndteres i for eksempel BMS. Ved å benytte MARIA kartmotor og Trackservice kan kontrollstasjonen enkelt integreres med forsvarssystemer som NorBMS og FACNAV.

Sist men ikke minst er kontrollstasjonen tenkt benyttet som en eksperimentplattform for ubemannede farkoster og vil støtte flere aktiviteter under FFIs strategiske satsning på dette området. Systemet tilbyr et plug-in-rammeverk og et programmeringsgrensesnitt (API) slik at man med liten innsats kan utvide systemet og kontrollere oppførselen til autonome farkoster og sensorer. Vi tror flere prosjektmiljøer ved FFI vil ha nytte av løsningen for eksperimentering med blant annet farkostoppførsel og farkostsamhandling samt sensorbehandling, og flere miljøer har allerede meldt sin interesse.

Gjennom en iterativ utviklingsprosess ledet av en «agile» (smidig) utviklingsmetodikk har gruppen tatt frem systemdesign og implementasjon. Fordi kontrollstasjonen skal benyttes som eksperimentplattform både for prosjekt 1359 og andre prosjektmiljøer, har det vært fokus på robust arkitektur og kodekvalitet for å sikre et system som kan utvides og vedlikeholdes.

Prosjektet har i dag en fungerende kontrollstasjon verifisert gjennom en serie tester hvor ubemannede flyvende farkoster er benyttet. Løsningen gir oss allerede nye kapasiteter i forhold til eksisterende systemer i form av evne til å kontrollere flere samtidige farkoster samt automatisk generering av ruter for akserekognosering/overvåkning.

English summary

This report seeks to document and announce the FFI GCS, a Ground Control System for unmanned vehicles designed and implemented at FFI through project 1359 “Technology for future UAS”. The motivation for creating a bespoke system was based on several factors; existing systems prove inadequate and lack the proper level of support for systems operators. They typically require several operators per UAV, which then scales poorly and incurs higher costs. By utilizing higher degrees of automation, the operator is freed up to perform other functions.

Existing systems would also need modifications in order to interoperate with systems central to the Armed Forces’ information infrastructure. In the future, autonomous units will be required to be managed in a homogenous fashion alongside other units from within systems such as the BMS. Utilizing the MARIA GIS system and Trackservice ensures easy integration with military systems such as NorBMS and FACNAV.

Finally, the GCS is planned used as a platform for experimentation with autonomous systems and will support multiple activities within FFI’s strategic investment in this area. The system provides an application programming interface (API) enabling users to extend the system and control the behavior of autonomous vehicles and sensors in an easy way. We believe that several projects at FFI could benefit from using the system for experimentation with autonomous vehicle behavior and collaboration as well as sensor processing.

The systems design and implementation was created through an iterative development process governed by an agile development methodology. Because the GCS is intended to be used by other projects at FFI as well as project 1359, focus has been on creating a robust systems architecture and high quality code in order to facilitate maintainability and extensibility. Today our project possesses a functioning GCS verified through a series of tests using autonomous aerial vehicles. The system is already providing us with additional capabilities over existing systems through the ability to control multiple, simultaneous vehicles, as well as automatic generation of flight plans for axis reconnaissance and surveillance.

Innhold

Forord

Undertegnede ønsker å rette en takk til Eirik Ulvik ansatt ved Teleplan Globe AS. Eirik har i svært stor grad bidratt med informasjon og støtte rundt bruk av MARIA GDK 2013. Denne komponenten har vært sentral for å realisere kontrollstasjonen, og Eirik har kunne bidra med mye «insider-tips» der dokumentasjonen har kommet til kort. Takk også til forhenværende forskningsleder Lorns Bakstad for inspirerende entusiasme og vidsynthet. Jørgen Nordmoen og Sondre Engebråten skal ha takk for utfyllende informasjon og bistand med MavLink og dennes implementasjon i Arduino Autopilot (APM).

1 Innledning

Gjennom diskusjoner innad i prosjekt 1359 ble det våren 2014 klart at det var behov for en egenutviklet kontrollstasjon for ubemannede systemer. I tillegg til å være menneske-maskin-grensesnittet mot ubemannede, autonome og semi-autonome farkoster ,ble det også tydelig at kontrollstasjonen ville bli en «hub» som binder sammen sub-systemer og tilstøtende systemer og et punkt for integrasjon mellom farkoster/sensorer og andre systemer. Selv om autonome farkoster på lengre sikt må bli i stand til å løse mange oppgaver uten kommunikasjon med bakkesystemer, vil det være behov for å disseminere telemetri, sensor-data og varslinger til andre systemer, farkoster og beslutningstagere mens operasjonen pågår.

Gjennom en iterativ utviklingsprosess ledet av en «agile» utviklingsmetodikk har gruppen tatt frem systemdesign og implementasjon. Fordi kontrollstasjonen skal benyttes som eksperimentplattform både for prosjekt 1359 og andre prosjektmiljøer, har det vært fokus på robust arkitektur og kodekvalitet for å sikre et utvidbart og vedlikeholdbart system.

Prosjektet har i dag en fungerende kontrollstasjon, kalt FFI GCS, som er verifisert gjennom en serie tester hvor ubemannede flyvende farkoster er benyttet. Løsningen gir oss allerede nye kapasiteter i forhold til eksisterende systemer i form av evne til å kontrollere flere samtidige farkoster samt automatisk generering av ruter for akserekognosering/overvåkning.

1.1 Motivasjon og bakgrunn

Beslutningen om å ta frem en egen kontrollstasjon for ubemannede farkoster ved FFI har bakgrunn i flere forhold:

- Behov for en kontrollstasjon som mer effektivt støtter operatører i deres oppdragsløsning. Dagens løsninger for å operere små og mellomstore UAS'er er mannskapsintensive. Operasjon av en liten UAV kan typisk kreve 2 til 3 mann per UAV for å betjene alle funksjonene. Dette medfører høye kostnader og vanskeliggjør skalering av løsningene. For å få ned mannskapsbehovet trengs økt automatisering, og prosjektet har identifisert en rekke funksjonsområder som egner seg godt for dette.
- Kompatibilitet med eksisterende systemer benyttet i det norske forsvaret. På sikt vil det være ønskelig å integrere kontrollsystemer for ubemannede systemer med eksisterende systemer som NorBMS, FACNAV, og lignende. Ved å benytte MARIA kartmotor fra Teleplan kan vi nyttiggjøre oss kart fra FMGT samt interoperere med MARIA-baserte systemer, blant annet ved å bruke Maria Trackservice. Dette fasiliterer for enkelt å kunne utveksle informasjon om "blue force tracking" og situasjonsbilde mellom systemene.
- Behov for kontrollstasjon som forsknings og eksperiment-plattform. En egenutviklet kontrollstasjon lar oss eksperimentere og få erfaring med sub-systemer og algoritmer for automatisk oppdragsløsning, ruteplanlegging, tilstandsovervåkning og autonom farkost-oppførsel. Moduler integrert med kontrollstasjonen får tilgang på telemetri tilhørende farkostene og aggregerte tilstandsdata fra kontrollstasjonen. De kan reagere på hendelser og

situasjoner og interagere med farkostene gjennom kontrollstasjonens programmeringsgrensesnitt (API). Ved å skrive "scripts" mot API'et i form av plugins gjør vi det mulig også for andre prosjektmiljøer å på en enkel måte prøve ut ny farkostoppførsel og sensorbehandling. Kontrollstasjonen er utviklet for å håndtere ulike typer autonome farkoster, slik som UAS, UGV, USV med flere.

Prosjektets undersøkelser har avdekket at ingen eksisterende kontrollstasjon tilbyr all funksjonaliteten vi trenger. Før beslutningen om å utvikle vår egen kontrollstasjon for ubemannede farkoster ble tatt ble det utredet hvorvidt det ville være hensiktsmessig å modifisere og utvide en allerede eksisterende løsning. En rekke åpen-kildekode-baserte kontrolltrollstasjoner eksisterer allerede, slik som Mission Planner, OpenPilot GCS, QGround Control, GCS Paparazzi med flere. Løsningene varierer en del i funksjonalitet og i forhold til hva slags kart-løsninger de støtter. Målt opp mot de ikke-funksjonelle kravene til løsningen (kap. 2.1) ble det tydelig at det mest hensiktsmessige ville være å designe og implementere vår egen kontrollstasjon. Tidligere erfaringer med større systemutviklingsprosjekter viser at det kan være raskere å komme i gang ved å utvide et eksisterende system, mens det over tid ofte blir svært ressurskrevende etter hvert som stadig mer funksjonalitet legges til og endres. Dette skyldes i hovedsak den underliggende systemarkitekturen, som nødvendigvis ble designet med et annet system i tankene. Det opprinnelige systemet ble gjerne utformet i henhold til andre krav og avgrensninger, og med en annen designfilosofi. Resultatet blir ofte at løsningen blir stadig dyrere å vedlikeholde og videreutvikle, og det melder seg omfattende behov for refaktorisering og kanskje også en fullstendig re-design av systemarkitektur og brukergrensesnitt.

FFIs behov for en eksperimentplattform for ubemannede farkoster stiller særlige krav til løsningen i forhold til utvidbarhet og mulighet for integrasjon. Kontrollstasjonen er designet med en plugin-arkitektur som muliggjør enkel utvidelse av eksisterende funksjonalitet uten behov for å rekompilere systemet.

2 Krav til løsningen

Før system-designet ble påbegynt ble flere i prosjektgruppen involvert i utarbeidelse av funksjonelle og ikke-funksjonelle krav. De funksjonelle kravene tok utgangspunkt i prosjektmedlemmers betydelige erfaring med eksisterende kontrollsystemer for UAS, og styrker, mangler og forbedringspotensial knyttet til disse. Kravene om Windows-kompatibilitet samt touch-basert GUI la føringer i forhold til teknologivalg, og det ble bestemt at .NET, C# og Windows Presentation Foundation (WPF) skulle benyttes.

2.1 Ikke-funksjonelle krav

En liste med ikke-funksjonelle krav til løsningen ble utarbeidet i fellesskap i prosjektet, og ble styrende for teknologivalg og systemarkitektur .

- Kunne kjøre på Windows-baserte PCer, laptops og tablets
- Skalerbart, touch-basert UI som kan betjenes i felt med fingre/stylus
- Fullstendig «self-contained», fungere uten tilgang til nettverk og internett
- Benytte MARIA kartmotor og kart fra FMGT
- Benytte MARIA Trackservice for enkel integrasjon med andre forsvarssystemer

2.2 Funksjonelle krav

En liste med funksjonelle krav til den første versjonen av kontrollstasjonen ble utarbeidet. Den fullstendige listen med funksjonelle krav er svært omfattende, og her følger en overordnet oversikt: Visualisering av multiple samtidige farkoster i kart, visualisering av telemetri, vise farkost-helse og status, laste opp flightplan med waypoint-lister til farkoster, take-off og flyving av flightplan, return to launch-funksjonalitet, generering av flightplan og tilordning av farkost, manuell planlegging av rute, automatisk generering av rute for akserekognosering/overvåkning, simulere flyving av rute, be en farkost gå til nytt punkt i kart.

3 Design

Det utvikles mye programvare på FFI, oftest ment for intern bruk og i form av prototyper, eksperimentell software og demonstratorløsninger. Mange av disse løsningene har relativt kort levetid da de har et spesifikt anvendelsesbehov begrenset i tid, f.eks som proof-of-concept løsninger, underlag for eksperimenter brukt for å komme frem til et forskingsresultat eller for å drive eksperimenter for en vitenskapelig artikkel. Andre løsninger støtter pågående, langsiktige aktiviteter og videreutvikles og vedlikeholdes over lang tid. FFI GCS faller i den siste kategorien, og er derfor utviklet med fokus på «patterns» og «best practices». Både utviklingsmetodikk, patterns og best practices er valgt for å understøtte utviklingen av en robust løsning som kan videreutvikles og vedlikeholdes over tid. Ved å la utviklingen av systemarkitekturen ledes av velprøvde designprinsipper minimeres sannsynligheten for at man tar uhensiktsmessige designvalg som vanskeliggjør videreutvikling på et senere tidspunkt.

3.1 Utviklingsmetodikk

Det er benyttet en iterativ utviklingsprosess og «agile» utviklingsmetodikk i arbeidet med FFI GCS. Med iterativ prosess menes at utviklingsaktivitetene foregår i sykluser, og at alle aktiviteter i noen grad gjennomføres i samtlige sykluser. Det betyr at man alt i første syklus gjennomfører kravspesifisering, design, utvikling og testing, selv om ikke alle kravene og hele designet er på plass enda. Det motsatte av en iterativ prosess er fossefallsmodellen, der man fryser og formelt lukker hver fase før man går videre til neste, og slik gjør seg ferdig med en fase uten å gjenoppta den aktuelle aktiviteten senere. I arbeidet med FFI GCS har vi lagt opp til korte iterasjoner bestående av identifisering av funksjonelle krav, tilpasning av applikasjonsarkitekturen, implementering av funksjonalitet og testing. Andre prosjektmedlemmer har blitt involvert blant annet for å avstemme

forventninger og funksjonalitet. Under utviklingen har det vært fokus på å ta frem en begrenset «tynn» løsning, altså en brukbar løsning med et minimum av funksjonalitet før man gir seg i kast med ytterligere funksjonalitet. Denne praksisen gjør at man tidlig har noe å vise frem til kunden (brukerne og beslutningstagere), og at man tidlig avdekker avvik mellom forventning og implementert funksjonalitet. Jo tidligere dette fanges opp desto mindre ressurskrevende vil endringene være.

Med en «agile» utviklingsmetodikk menes at prinsipper for Agile Software Development styrer systemutviklingsprosessen. Disse prinsippene innbefatter å omfavne endringer av kravene underveis, hyppige leveranser, involvering av brukerne, fremdrift målt på fungerende programvare og jevnlig vurdering og eventuell endring av løsningen. Det første prinsippet er bygget på erkjennelsen av at endringer av kravene alltid vil oppstå, og at det er billigere å håndtere dette tidlig enn sent. Ved ikke å tillate endringer i kravene ender man opp med et produkt som er ubrukelig for brukerne. Hyppige leveranser gir hyppige muligheter for å detektere avvik mellom forventning og implementasjon, og involvering av brukerne er kritisk for å ende opp med et nyttig produkt. I dette prosjektet er undertegnede både systemutvikler og bruker, men andre brukere har blitt involvert underveis.

3.2 Patterns og Best Practices

Best practices er et sett med retningslinjer som representerer lærdom gjort av det kollektive utviklermiljøet, og retningslinjene er ment å styre utviklere unna fallgruver og peke de i retning av velprøvde, smarte designvalg. Ikke alle regler er allmenngyldige i enhver situasjon, og noen regler kan være motstridende. Det er allikevel nyttig å kjenne til mange av dem, da man som utvikler ofte kommer opp i problemstillinger som ligner på kjente problemstillinger eller faller inn i en kategori av problemstillinger hvor gitte hensyn er fordelaktige å ta. Ved å lese om og jobbe med best practices blir man som utvikler også mer reflektert rundt vanlige problemstillinger, og utvikler en «mageølelse» for hva som kan være viktige hensyn å ta og hvilke valg som kan være hensiktsmessige.

Patterns inngår i best practices, og er spesifikke design-maler eller mønstre som har vist seg hensiktsmessige for utvikling av programvare. Det finnes patterns for svært mange aspekter av programvaren, f.eks. lagdeling og organisering av programvaren (systemarkitekturen), hvilke klasser som har ansvar for hvilke oppgaver, hvilke objekter som skal kunne referere til hverandre, hvilke objekter som skal kunne opprette hvilke andre objekter, generalisering av grensesnitt, innkapsling av sub-systemer og lignende. Ulike patterns kan også være motstridende, og man må som utvikler velge det som synes mest hensiktsmessig for den aktuelle løsningen og i vekselvirkning med de andre benyttede patterns. I arbeidet med FFI GCS er det gjennomgående benyttet patterns og prinsipper for objektorientert systemutvikling tilhørende et sett som heter General Responsibility Assignment Software Patterns (GRASP).

3.3 Systemarkitektur

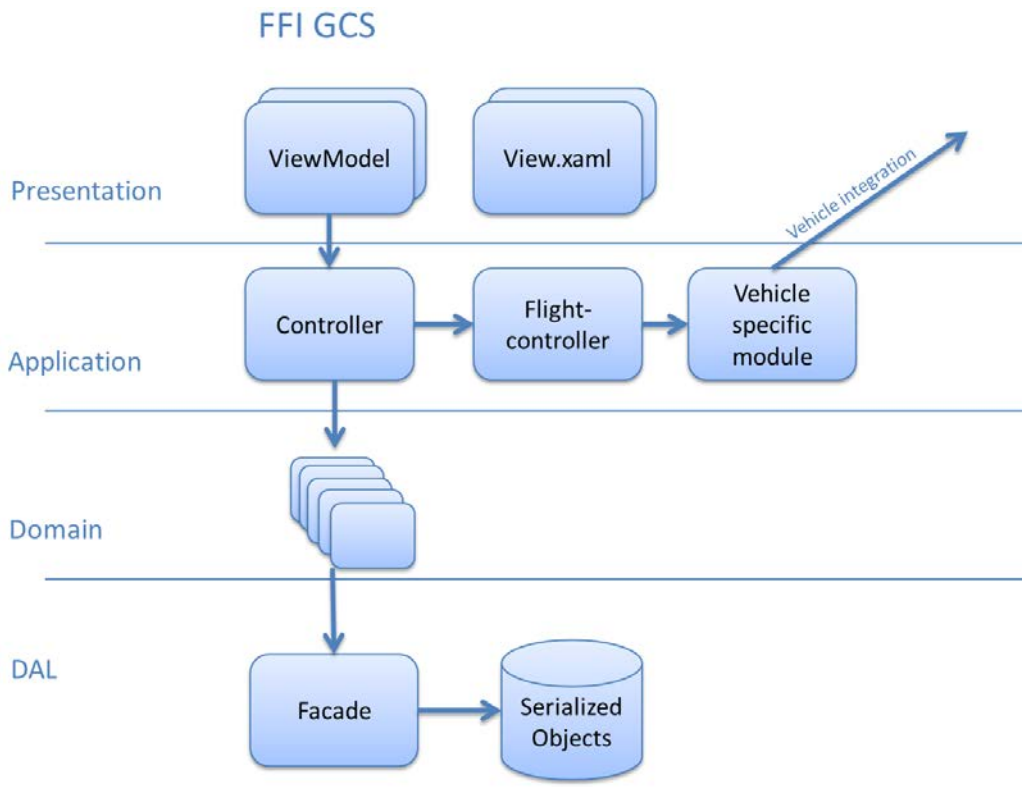
GRASP-pattern'et «Application Layers» har vært styrende for utviklingen av systemarkitekturen. Dette innebærer en oppdeling av systemarkitekturen i ulike lag, hvor 3-lags arkitektur er mest utbredt. Vi har valgt en 4-lags arkitektur, med lagene «presentation», «application», «domain» og «DAL» (data access layer). Det ekstra «application»-laget fasiliterer for integrasjon med andre systemer, og

FFI GCS integrerer med hardware og software i form av autonome farkoster, og autopilotene om bord i disse. Hensiktene med en lagdelt systemarkitektur er flere: Lagdelingen hjelper å ivareta designprinsippet om «high cohesion, low coupling», hvilket innebærer at hvert lag kun gjør det de har ansvar for, og at koblingen mellom lagene holdes minimal. Sistnevnte fasiliterer for enklere vedlikehold og refaktorisering av løsningen. Layers-pattern'et sier også at referanser mellom objekter i de ulike lagene skal gå nedover i lagshierarkiet, altså at objekter i presentasjonslaget kan referere objekter i applikasjonslaget, og at objekter i applikasjonslaget kan referere objekter i domenelaget, men ikke motsatt. Dette bidrar også til «low coupling», og gjør at man unngår «spagetti-kode» der alle deler av applikasjonen er sammenkoplet med alle andre. Layers-pattern'et anbefaler å unngå referanser som hopper over et eller flere lag (f.eks. direkte fra presentasjonslaget til domenelaget), men dette er ikke en absolutt regel. Andre fordeler med lagdeling er skalering og ytelse, hvor f.eks datalaget (DAL) kan flyttes til egen hardware for å møte krav om ytelse.

Et annet arkitektur-pattern benyttet for FFI GCS er «MVVM» (Model View ViewModel), en avart av Martin Fowlers «presentation model» design pattern. Dette er støttet av Microsofts WPF, og fasiliterer for separasjon av presentasjons-«markup» og presentasjonslogikk. Dette muliggjør igjen automatisert testing av GUI'et i tillegg til at det er enklere å endre og vedlikeholde GUI. Det gjør også at GUI-logikk og GUI-design kan utvikles separat, og sistnevnte kan settes bort til en dedikert designer.

Løsningen har per nå et beskjedent lagringsbehov i den forstand at det er små datamengder som lagres i form av Missons og Flightplans. Som lagringsløsning er det derfor valgt objektpersistens i form av serialiserte binære data. Da dette gjøres gjennom generell kode og ikke krever spesifikk lagringslogikk for hver enkelt operasjon eller use case muliggjør det rask utvikling og lite vedlikehold. Mens denne tilnærmingen er godt egnet per i dag kan det i fremtiden dukke opp behov for lagring av mer data, og man kan bli nødt til å integrere løsningen mot en relasjonsdatabase, document store eller en annen lagringsteknologi. Dette er forberedt gjennom bruken av «facade»-pattern'et, som omhandler innkapsling av et helt sub-system gjennom et enhetlig grensesnitt. Innkapslingen bidrar til lav grad av kobling mellom lagringslogikken og resten av systemet ved at ingen deler av systemet tillates å referere til kode som befinner seg bakenfor fasaden. Dette gjør at endringer i lagringslogikken ikke berører resten av systemet overhodet, men er begrenset til koden i DAL-laget. Et bytte til f.eks. en relasjonsdatabase vil således kun kreve endringer i DAL-laget.

Figur 3.1 viser den overordnede applikasjonsarkitekturen for FFI GCS. Figuren viser de ulike lagene, samt viktige klasser. Klassen FlightController har ansvaret for integrasjonen mot farkoster, via system-spesifikke adaptere (vehicle specific modules).



Figur 3.1 Applikasjonsarkitektur for FFI GCS.

3.4 Plugin-arkitektur

Plugin-arkitekturen er basert på Managed Extensibility Framework (MEF), en teknologi introdusert i .NET 4.5. MEF er et rammeverk for enkelt å lage utvidbare applikasjoner gjennom plugins, og støtter automatisk deteksjon av plugin-assemblies (.dll-filer) uten å behøve konfigurasjon. Ved å skrive plugins for FFI GCS kan systemets funksjonalitet utvides uten at det behøver å rekompileres. For å skrive en plugin behøver man heller ikke kildekoden eller den kompilerte kontrollstasjonen, men kun et bibliotek vi har kalt GcsPluginsCommon. Dette biblioteket inneholder de nødvendige kontraktene i form av to abstrakte klasser: PluginBase og GcsBase. Når man utvikler en plugin for FFI GCS lar man den arve PluginBase. For å interagere med kontrollstasjonen kaller man metoder på GcsBase, f.eks FlyTo(..), SetMode(..), SubscribeForTelemetry(..) etc.

Plugins kan også abonnere på «events» fra kontrollstasjonen. Disse er typisk assosiert med en autonom farkost og beskriver tilstanden til farkosten, slik som «Armed», «Launched», «Waypoint_Reached», «Flightplan_Completed», «Landed» etc. Ved å abonnere på telemetri-data (f.eks. posisjon og retning) og events fra ulike farkoster kan man gjøre farkostene i stand til å samhandle. Dette kan innebære at de posisjonerer seg i forhold til hverandre, unngår hverandre (avoidance), følger hverandre eller samarbeider om å løse oppgaver. Utvikleren av plugins står fritt til å velge om hun ønsker å styre samtlige farkoster fra én plugin, eller om hun ønsker én plugin per farkost. Plugins kan kommunisere med andre plugins ved å sende hverandre egendefinerte meldinger.

For eksperiment og demonstrasjons-formål skrev prosjektet en plugin som gjør en farkost i stand til å følge en annen. Lederfarkosten («RavnMaster») fløy en pre-programmert rute i form av en flightplan

med «waypoints» (3D-koordinater). Følgefarkosten («RavnSlave») abonnerte på telemetridata og events fra RavnMaster, og planla således sin rute i sanntid basert på bevegelsene til RavnMaster. For å minimere sjansen for uhell holdt RavnSlave seg 5 meter høyere enn RavnMaster, og med en minimumshøyde på 10 meter (for å unngå potensiell kollisjon med bakken). RavnSlave ble tatt opp i luften først, og startet sin autonome ferd automatisk når RavnMaster startet flyvningen av sin pre-programmerte rute, og genererte et «Launched»-event. Først når RavnMaster hadde fløyet sin rute og genererte et «Flightplan_Completed»-event sluttet RavnSlave å følge RavnMaster, og initierte en «Return To Launch»-sekvens. Hele plugin'en opptar 43 kodelinjer, og er gjengitt i Figur 3.2.

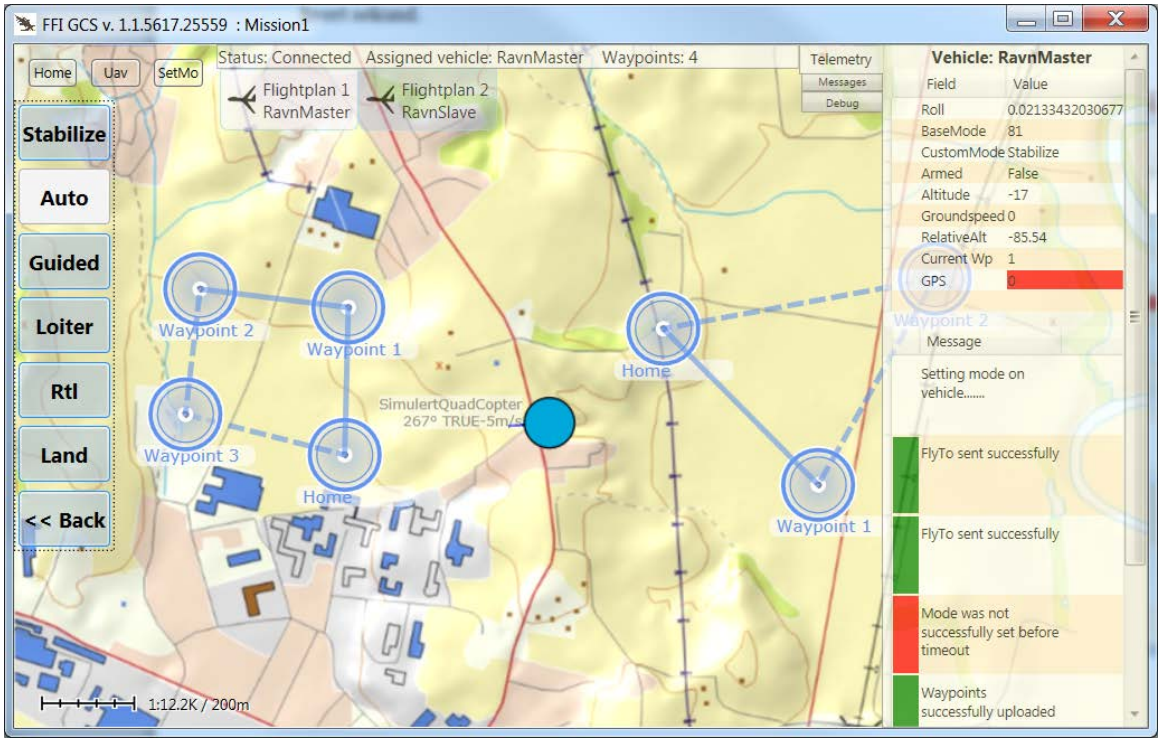
```
[Export(typeof(PluginBase))]
public class Plugin1 : PluginBase
{
    private GcsBase _gcs;
    private bool _isRavnMasterFlying, _isRavnSlaveFlying;
    private const string RavnMaster = "RavnMaster";
    private const string RavnSlave = "RavnSlave";

    public override void Load(GcsBase gcs)
    {
        _gcs = gcs;
        _gcs.SubcribeForTelemetryForVehicle(this, RavnSlave, RavnMaster);     //start receiving telemetry for myself and ravn master
        _gcs.AddReceiveTelemetryAction(this, OnReceiveTelemetry);    //bootstrap telemetry receiving
        _gcs.AddEventAction(OnEvent);                        // bootstrap event receiving
        _gcs.Log("Plugin for RavnSlave has loaded");
    }

    public void OnReceiveTelemetry(string vehicle, IDictionary<string, string> telemetryMap)
    {

        if (_isRavnMasterFlying && _isRavnSlaveFlying && vehicle.Equals(RavnMaster)) //both vehicles are flying, and telemetry from ravn master
        {
            try
            {
                var lat = Double.Parse(telemetryMap["lat"]);
                var lon = Double.Parse(telemetryMap["lon"]);
                var alt = Double.Parse(telemetryMap["RelativeAlt"]);
                if (alt < 10) alt = 10; //for safety

                _gcs.FlyTo(RavnSlave, lat, lon, (int)alt + 5);  // fly ravn slave to the reported position of RavnMaster, but 5 meters higher
            }
            catch (Exception) {return; }
        }
    }

    public void OnEvent(string vehicle, string type) //e.g. arm, launch, land, reach waypoint etc
    {
        if (vehicle.Equals(RavnMaster) && type.Equals(GcsBase.Event.Launched.ToString())) //ravn master has launched
            _isRavnMasterFlying = true;
        else if (vehicle.Equals(RavnSlave) && type.Equals(GcsBase.Event.Launched.ToString())) //ravn slave has launched
            _isRavnSlaveFlying = true;
        else if (vehicle.Equals(RavnMaster) && type.Equals(GcsBase.Event.FlightplanCompleted.ToString()))   //flightplan completed
        {
            _isRavnMasterFlying = false; //stop following RavnMaster
            _isRavnSlaveFlying = false; // reset
            _gcs.SetMode(RavnSlave, "RTL"); //Finished, ravn slave return to launch
        }

    }
}
```

Figur 3.2 Plugin for å la en autonom farkost følge en annen.

4 Implementasjon og funksjonalitet

Da FFI GCS er basert på .NET og WPF og benytter et touch-basert GUI kan den kjøres på windows-PCer og tablets i felt og opereres med finger/stylus. GUI'ets hovedkomponent er en Maria kartmotor-komponent, og denne dekker hele skjermen. Øvrige komponenter er plassert oppå kartkomponenten og gjort semi-transparente for at operatøren skal ha størst mulig synlig kartflate. GUI'et er laget dynamisk slik at det skalerer hensiktsmessig etter tilgjengelig skjermstørrelse. Når skjermstørrelsen minsker bevarer komponentene i GUI'et sin form og størrelse for at knapper skal kunne betjenes og tekst leses. Komponentene tar således opp en større prosentandel av størrelsen på en liten skjerm enn på en stor skjerm. På en større skjerm vil en større kartflate være tilgjengelig for brukeren til enhver tid. Systemet er utformet slik at en rekke bruker-initierte oppgaver kjører i egne tråder for å unngå at brukergrensesnittet oppleves ikke-responsivt mens oppgaven utføres. Enkelte funksjoner kan imidlertid være utilgjengelige (knappene er grået ut) mens oppgaver utføres for å unngå at systemet kan settes i ugyldige tilstander.

Figur viser GUI'et slik det ser ut når man er oppkoblet mot én eller flere farkoster. Feltet øverst på skjermen viser gjeldende flightplan og tilknyttet farkost (Flightplan 1 og RavnMaster), samt oppkoblingsstastus for farkosten (Status: Connected). Sistnevnte betyr at kommunikasjon er opprettet og fungerende mellom kontrollstasjonen og farkosten. Feltet øverst til høyre på skjermen viser valgt farkost (RavnMaster) samt sanntids-telemetridata fra farkosten. Her kan det avleses hvor mange satellitter GPS-modulen ser, hvilken modus farkosten står i, om farkosten er armert eller ikke samt høyde relativt til launch-punkt. For lesbarhet oppdateres telemetridataene i GUI'et ikke oftere enn hvert sekund selv om enkelte verdier kan mottas så ofte som 10 ganger per sekund. Verdier som er utenfor definerte områder vil flagges i GUI'et med rød eller oransje farge, eks: Dersom farkosten ser færre enn 5 GPS-satellitter flagges det oransje, og dersom farkosten ser 3 eller færre flagges det rødt.



Figur 4.1 Touch-basert GUI, FFI GCS

Nederst i feltet til høyre vises systemmeldinger til operatøren. Når operatøren initierer en operasjon, f.eks. opplasting av flightplan til farkost eller en «Set Mode»-kommando, vil kommandoen først vises som initiert, og fremdrift vil vises som en prikkete linje. Dersom kommandoen lykkes utført innen en gitt tid erstattes meldingen av en grønn melding som indikerer at operasjonen var vellykket. I motsatt fall vises en rød feilmelding. Meldingene blir liggende under hverandre i en liste slik at operatøren til enhver tid har en historikk på tidligere utførte kommandoer.

På venstre side av skjermen ser man i Figur en «drill down»-meny, altså en hierarkisk meny bestående av flere. Brukeren presenteres først for rot-nivået (Home), og ved å klikke på et menyvalg driller brukeren seg ned i menyhierarkiet, og de synlige menyvalgene erstattes av menyvalgene tilhørende undermenyen. Øverst på skjermen, over menyen, vises «breadcrumbs», en sti som viser hvor i menyhierarkiet man befinner seg. Operatøren kan også klikke i stien for å komme tilbake til høyereliggende nivåer.

Menykomponenten er laget som en frittstående WPF-komponent (i MVVM-utførelse), og konfigureres for bruk i applikasjonens konfigurasjonsfil (app.config). Her legges en konfigurasjonsseksjon med navnet «LeftMenu» som angir den hierarkiske menystrukturen, navnet på menyvalg, hvor mange menyvalg som kan vises på samme side og lignende. Et utdrag av konfigurasjonen vises i Figur 3.2.

```
<LeftMenu>
  <add key="MaxNoOfButtons" value="10" />
  <add key="ShowBreadCrumbs" value="True" />
  <add key="ToggleMethods" value="Home_New_Waypoint,Home_New_UAV,Home_Flightplan_SelectRoad,Waypoint_Move,Home

  <add key="Home" value ="Home_Mission,Home_Flightplan,Home_New,Home_Simulate,Home_InitUav,Home_Logging,Home_U
  <add key ="Home_New" value="Home_New_Waypoint,Home_New_UAV" />
  <add key ="Home_Flightplan" value="Home_Flightplan_New,Home_Flightplan_Edit,Home_Flightplan_Clear,Home_Fligh
  <add key ="Home_Uav" value="Home_Uav_Arm,Home_Uav_SetMode,Home_Uav_FlyTo,Home_Uav_UploadWP" />
  <add key ="Home_Uav_SetMode" value="Home_Uav_SetMode_Stabilize,Home_Uav_SetMode_Auto,Home_Uav_SetMode_Guided
  <add key ="Home_Mission" value="Home_Mission_New,Home_Mission_Load,Home_Mission_Edit,Home_Mission_Close" />
</LeftMenu>
```

Figur 3.2. Et utdrag av konfigurasjonen for meny-komponenten.

Konfigurasjonen lar oss angi «ToggleMethods», menyvalg som vises som aktivert i form av rød tekst på knappen når den klikkes på. Dette benyttes for å informere operatøren om at operasjonen er påbegynt men ikke ferdigstilt. Et eksempel er «Fly To»-knappen: Når denne klikkes på blir den rød, og et påfølgende klikk i kartet sender den gjeldende farkosten til punktet i kartet. Knappen får så tilbake sin ordinære farge, og flyTo-kommandoen er utført (selv om farkosten enda ikke har ankommet punktet i kartet). Noen ToggleMethods forblir røde etter å ha blitt klikket på inntil de blir klikket på én gang til.

4.1 Vehicle Specific Module

Vechicle Specific Modules (VSMs) er adaptere som håndterer kommunikasjonen mellom FFI GCS og én spesifikk type farkost-autopilot eller sensor-hardware. For UAV'er har en slik autopilot i oppgave å holde farkosten flyvende i lufta ved å bestemme kraften til de ulike motorene og styre farkosten mot definerte waypoints. Prosjektet har per nå laget VSM'er for Ardupilot, Arducopter og en egenutviklet autopilot for UGV'en OLAV (Offroad Light Autonomous Vehicle). De to første er Arduino-baserte autopiloter som kontrollerer henholdsvis fly og multikopter, mens den siste er tatt frem på prosjekt 1371 – «Ubemannede kjøretøyer for Forsvaret» og kommuniserer med UGV. De arduino-baserte autopilotene er basert på både åpen programvare og åpen hardware, og er i utstrakt bruk på tvers av farkosttyper i prosjektet. I fremtiden vil vi lage VSM for nye autopiloter etter behov, og sannsynlige kandidater er Pixhawk og PD-100's autopilot. I tillegg til prosjekt 1371 kan det bli aktuelt å bistå andre prosjektmiljøer på FFI med å lage VSM for andre typer farkoster.

Kompleksiteten til en VSM vil variere noe etter hvilke operasjoner som er støttet av autopiloten, samt hva slags kommunikasjonsprotokoll som kreves. Den implementerte VSM'en for Arducopter er på ca 400 kodelinjer, og benytter MAVLINK-protokollen over trådløs seriellport. VSM har ansvaret for følgende oppgaver:

- Motta kommando fra brukeren/systemet, og generere riktig type melding som sendes til farkosten over radio.
- Overvåke responsmeldinger fra farkosten og eventuelt forsøke kommandoer utført på nytt. Rapportere resultat til GUI/bruker.
- Lytte til og motta telemetri-meldinger fra farkost, dekode disse og hente ut relevante verdier og rapportere disse videre i systemet. Eks: posisjon og retning.

- Overvåke spesifikke telemetri-verdier, og generere events når disse endres i samsvar med gitte regler. Eks: Når Current Waypoint endres til nr. 2 (betyr at farkosten er ferdig med auto-takeoff), generér et Launched-event.

```
TelemetryManager.GetInstance().AddFieldChangeEvent(Uav, "Current Wp",
    (oldValue, newValue) => newValue.Equals("2"), GcsBase.Event.Launched);
```

på hvordan man fra VSM kan be systemet overvåke telemetrivariabler, og generere et event når betingelsene er tilfredsstilte. Her er det benyttet prinsipper fra funksjonell programmering ved at det brukes lambda-uttrykk og høyere-ordens-funksjoner.

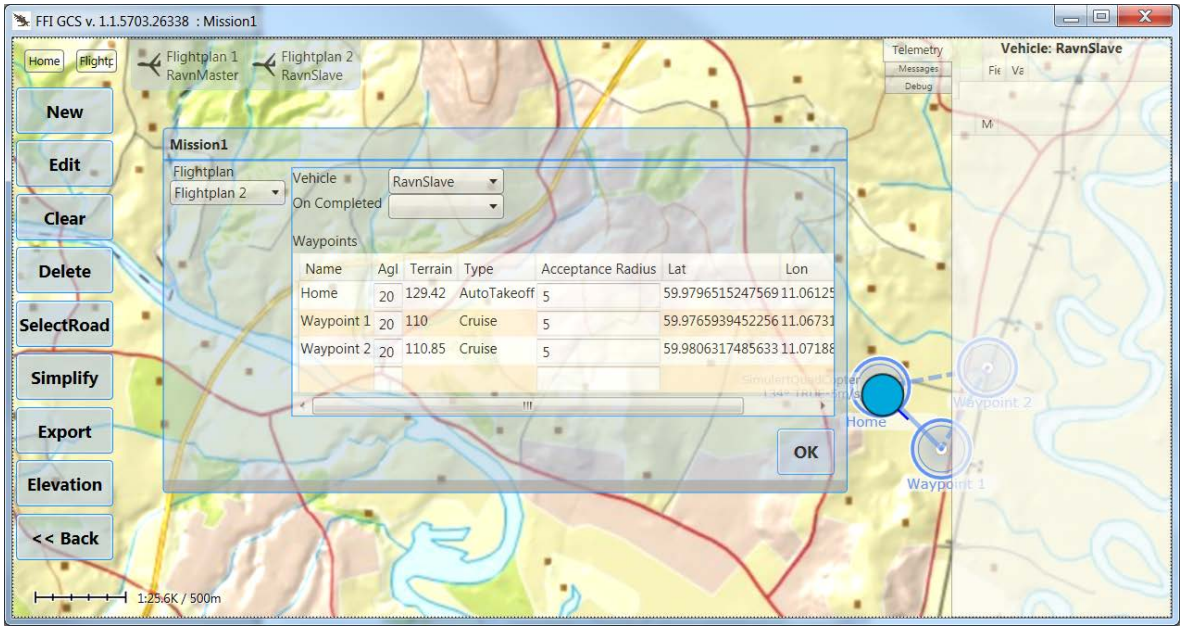

```
TelemetryManager.GetInstance().AddFieldChangeEvent(Uav, "Current Wp",
    (oldValue, newValue) => newValue.Equals("2"), GcsBase.Event.Launched);
```


4.2 Funksjonalitet

Gjeldende versjon av kontrollstasjonen innehar all funksjonalitet nødvendig for å operere flere samtidige farkoster på autonom og semi-autonom måte. I tillegg til basisfunksjonalitet er det implementert funksjonalitet for å støtte og avhjelpe operatøren i hennes arbeid med planlegging og utføring av oppdrag.

4.2.1 Basisfunksjonalitet

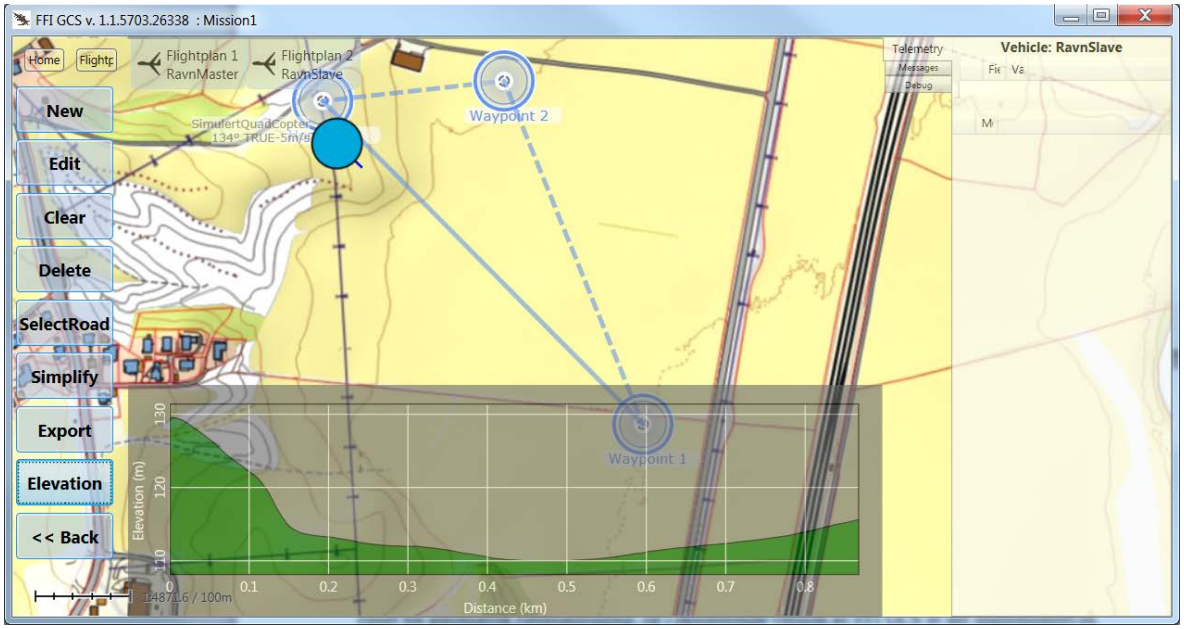
Når man planlegger ruter og flyr oppdrag i FFI GCS jobber man alltid med entiteter samlet under et «Mission». Under et Mission har man én eller flere «Flightplans». En Flightplan inneholder en oppdragsbeskrivelse som kan forstås og utføres av farkostens autopilot, slik som auto-takeoff, fly til en bestemt høyde, fly en rute bestående av waypoints, holde seg i luften i et bestemt tidsrom, og landing. Denne oppdragsbeskrivelsen lages ved å betjene funksjoner i GUI'et, slik som å utplassere waypoints i kartet. Når man klikker i kartet (med mus, stylus eller finger på pekeskjerm) og oppretter et nytt waypoint gis det automatisk en default-høyde over bakken som er definert i konfigurasjonsfilen. Denne høyden samt andre detaljer ved waypoint'ene kan avleses og endres under menyen Flightplan -> Edit, se Figur 4.3.



Figur 4.3 Menyen Flightplan -> Edit.

En Flightplan kan kun ha én farkost tilknyttet om gangen. For å fly flere samtidige farkoster må det opprettes flere Flightplans under samme Mission. Flightplan'er kan utføres i parallell eller i sekvens.

Terrenghøyde hentes ut fra MARIA, og benyttes ved flyving av farkoster. Når operatøren spesifiserer høyden på Waypoints angis dette som Above Ground Level (AGL) i meter. Dette gjør det mer intuitivt for operatøren å planlegge ruten enn om høyde over havet eller høyde relativt til Home-punktet benyttes. Høydeprofilen for terrenget langs hele ruten uttrykt som meter over havet er tilgjengelig for operatøren gjennom menyen Flightplan -> Elevation, se Figur 4.4.

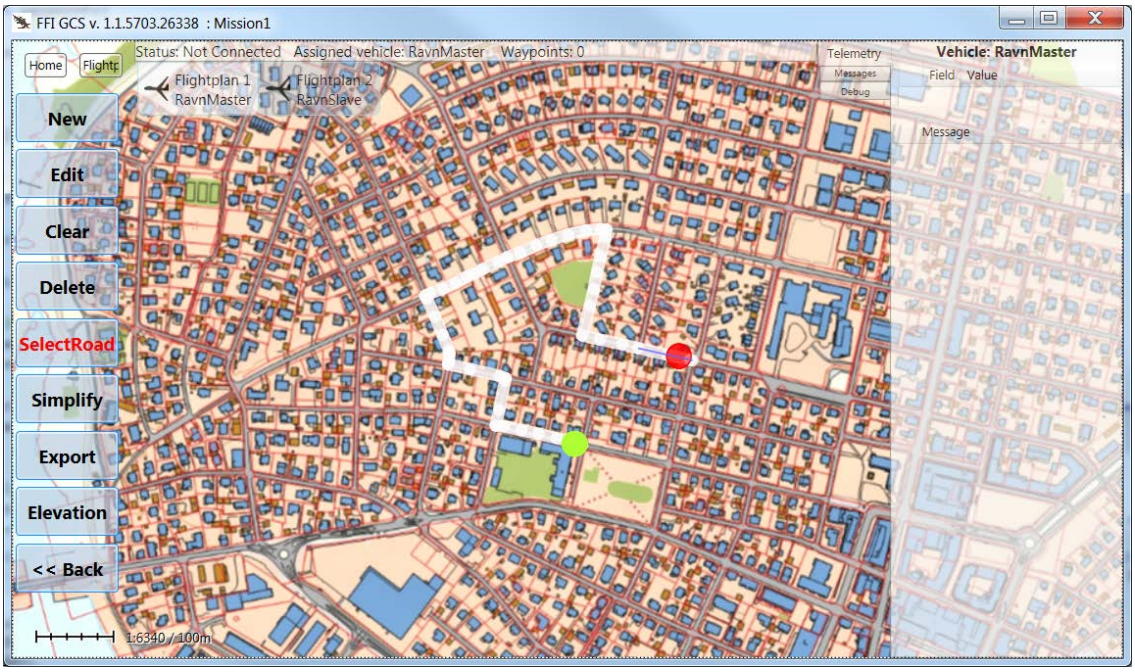


Figur 4.4. Høydekurven for terrenget langs ruten. Y-aksen viser terreng-høyde og X-aksen viser avstand.

Når én eller flere Flightplan'er er generert og populert med waypoints lastes de opp til de respektive farkostenes autopiloter. Dette gjøres under menyen UAV -> Upload Waypoints. Under opplastingen bekreftes hvert enkelt waypoint av autopiloten, og når samtlige er bekreftet vises opplastingen som vellykket i GUI. Under UAV-menyen finnes også funksjoner for å armere farkosten, sette farkosten i ulike modi samt utføre «Fly-To» kommando slik at UAV'en avbryter eventuell waypoint-rute og flyr til den posisjonen i kartet man klikker på. Waypoint-ruten kan deretter gjenopptas ved å endre farkostens modus til «auto».

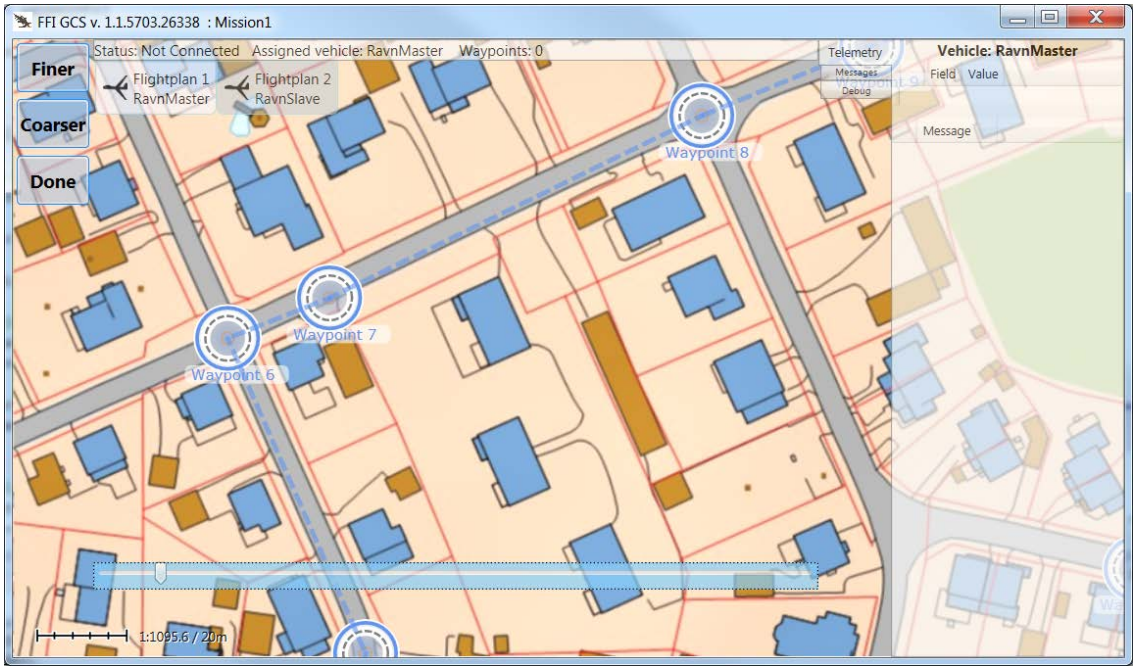
4.2.2 Automatisk generering av rute fra vei

Akseoppklaring og overvåkning er en sentral eksperiment-aktivitet i prosjekt 1359, og egner seg godt for bruk av UAS. Man ønsker typisk å oppklare eller overvåke en vei, og kan benytte én eller flere UAV'er til å avbilde strekningen med en gitt gjenvisitasjonstid. Billedmaterialet kan så benyttes til analyse, til å generere 3D-modeller eller ortografiske kart, eller til endringsdeteksjon ved gjenvisitt. Selv om ruten kan planlegges ved manuelt å opprette waypoints langs strekningen blir dette fort tidkrevende ved store avstander. Det kan også forekomme forhold ved strekningen, farkosten eller avbildingsutstyret (kamera, kamerafeste, «Gimball») som gjør at det finnes en optimal høyde og posisjonering i forhold til veien for å oppnå best mulig resultat. Disse faktorene gjør oppgaven godt egnet for automatisk ruteplanlegging, og i eksisterende versjon av FFI GCS er det implementert en funksjon som tar utgangspunkt i en angitt veistrekning og genererer en rute basert på denne, se Figur 4.5



Figur 4.5 Automatisk generering av rute for oppklaring/overvåkning av vei

I gjeldende versjon støttes operatøren ved at ruten genereres automatisk basert på angitt veistrekning og oppløsning på waypoint'ene. Sistnevnte angis ved at operatøren angir en toleranse for antall meters avvik fra midtlinjen i veien, og gjøres med knapper eller «slider» i samme grensesnitt, se Figur 4.6.

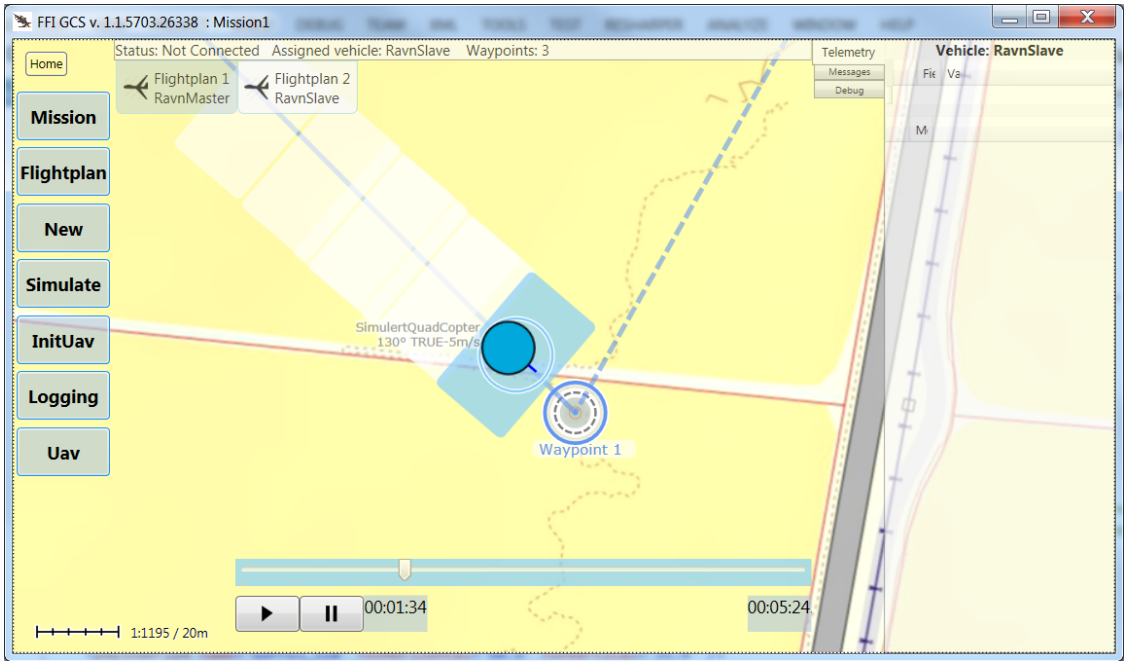


Figur 4.6 Oppløsningen på waypoint'ene angis ved å dra i slideren, eller ved bruk av knappene

I fremtidige versjoner av denne funksjonaliteten vil oppløsningen på waypoint'ene automatisk bestemmes basert på de forhold som har innvirkning på optimal avbildning.

4.2.3 Simulering av rute

Som en støtte til operatøren tilbys en enkel innebygget simulator. Denne lar operatøren lage/generere en rute samt velge farkost og avbildningsutstyr, for så å simulere flyvning av ruten. Ruten kan flys i sanntid eller i ønsket hastighet, og operatøren kan også «spole» fremover og bakover for å se når farkosten vil være ved gitte punkter på ruten. Simulatoren hensyntar farkostens flyveegenskaper slik som cruise-hastighet, klatre-rate og svingradius. Den kalkulerer og visualiserer også avbildings-sensorens bakke-avtrykk som en funksjon av sensorens field-of-view (FOV) samt farkostens posisjon og høyde over bakken. Dette benyttes som en støtte til operatøren for å verifisere at relevante områder avbildes/dekkes. Figur 4.7 viser simulert flyvning av en rute med en spesifikk farkost og avbildings-sensor. I dette eksempelet benyttes et kamera med FOV 122x94 meter når farkosten flyr 20 meter over bakken. Det blå rektangelet under farkosten viser sensor-avtrykket i sanntid mens den hvite halen viser hva sensoren tidligere har dekket. Disse verdiene konfigureres i systemets konfigurasjonsfil, «App.config». Figur 4.8 viser relevante seksjoner i konfigurasjonsfilen for overnevnte verdier.



Figur 4.7 *Simulering av rute fløyet med en spesifikk farkost og avbildningsutstyr. Avbildningsutstyrets dekningsområde vises som et blått rektangel.*

```
<UavTypes>
  <uavType name="SkyWalkerX8" cruiseVelocity="22.0" climbRate="3.0" turnRadius="100" />  <!-- Units of m/s, m/s and m, respectively -->
  <uavType name="BlackAdder" cruiseVelocity="22.0" climbRate="3.0" turnRadius="50" />  <!-- Units of m/s, m/s and m, respectively -->
  <uavType name="Quadcopter" cruiseVelocity="5.0" climbRate="2.2" turnRadius="5" />
</UavTypes>

<PayloadTypes>
  <payloadType name="GoPro3_Wide" fovHorizontal="122.6" fovVertical="94.4" />
  <payloadType name="GoPro3_Sim" fovHorizontal="60.6" fovVertical="35.4" />
</PayloadTypes>

<AutoPilotTypes>
  <autoPilotType name="ArduPilot2"/>
  <autoPilotType name="Simulator" />
</AutoPilotTypes>

<UAVs>
  <uav name="RavnMaster" uavType="Quadcopter" comPort="12" baudrate="115200" autoPilotType="ArduPilot2" payloadType="GoPro3_Wide" />
  <uav name="RavnSlave" uavType="Quadcopter" comPort="12" baudrate="115200" autoPilotType="ArduPilot2" payloadType="GoPro3_Wide" />
  <uav name="Ravn3" uavType="BlackAdder" comPort="12" baudrate="115200" autoPilotType="ArduPilot2" payloadType="GoPro3_Wide" />
  <uav name="BlackAdder2" uavType="BlackAdder" comPort="12" baudrate="115200" autoPilotType="ArduPilot2" payloadType="GoPro3_Wide" />
  <uav name="SimulertQuadCopter"  uavType="Quadcopter" comPort="dummy" baudrate="57600" autoPilotType="Simulator" payloadType="GoPro3_Wide" />
</UAVs>
```

Figur 4.8 *Konfigurasjon av farkost-typer, payload/sensor-typer, autopilot-typer og konkrete UAV'er.*

5 Oppsummering

Gjennom diskusjoner innad i prosjekt 1359 ble det våren 2014 klart at det var behov for en egenutviklet kontrollstasjon for ubemannede systemer ved FFI. Beslutningen har bakgrunn i at eksisterende løsninger ikke er tilstrekkelige, at kompatibilitet med forsvarssystemer som NorBMS og FACNAV vil være ønskelig, samt et behov for en forsknings og eksperiment-plattform. Vi har kalt

kontrollstasjonen FFI GCS, og den foreligger i dag i en versjon (v.1.1.5) som er verifisert gjennom en serie med praktiske tester. Testene er avviklet ved en modellflystripe på Kløfta, og inkluderer flyving med samtidige heterogene farkoster, både fly og multikopter. Den er også benyttet av UGV-prosjektet til å følge og overvåke UGV i kart i sanntid. I løpet av høsten 2015 vil UGV også kunne styres/kontrolleres fra kontrollstasjonen. Løsningen gir oss allerede nye kapasiteter i forhold til eksisterende systemer i form av evne til å kontrollere flere samtidige farkoster samt automatisk generering av ruter for akserekognosering/overvåkning. Den gir oss også mulighet til å eksperimentere og få erfaring med sub-systemer og algoritmer for automatisk oppdragsløsning, ruteplanlegging, tilstandsovervåkning og autonom farkost-oppførsel. Det integrerte plugin-rammeverket gir både oss og andre prosjektmiljøer mulighet til enkelt å utvide kontrollstasjonens funksjonalitet. Ved å skrive scripts mot det utviklede programmeringsgrensesnittet (API'et) kan man enkelt prøve ut ny farkostoppførsel og sensorbehandling. Dette kan benyttes for å gjøre autonome farkoster i stand til å samhandle, og kan innebære at de posisjonerer seg i forhold til hverandre, unngår hverandre (avoidance), følger hverandre eller samarbeider om å løse oppgaver.

Vi ser FFI GCS vil kunne støtte flere aktiviteter under FFIs strategiske satsning på autonome farkoster, og det er således lagt vekt på utviklingsprosess og kodekvalitet. Både utviklingsmetodikk, patterns og best practices er valgt for å understøtte utviklingen av en robust løsning som kan videreutvikles og vedlikeholdes over tid. Ved å la utviklingen av systemarkitekturen ledes av velprøvde designprinsipper minimeres sannsynligheten for at man tar uhensiktsmessige designvalg som vanskeliggjør videreutvikling på et senere tidspunkt.

Prosjektet har per nå laget tilpasning (VSM) for Ardupilot og Arducopter, to Arduino-baserte autopiloter som kontrollerer henholdsvis fly og multikopter. Det er også laget VSM for å kommunisere med autonomt bakkekjøretøy for prosjekt 1371. I fremtiden vil vi lage VSM for nye autopiloter etter behov, og sannsynlige kandidater er Pixhawk og PD-100's autopilot. Det kan også bli aktuelt å bistå andre prosjektmiljøer på FFI med å lage VSM for andre typer farkoster. Vi håper og tror FFI GCS kan bli en ressurs også for andre prosjektmiljøer på FFI.