Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for teknisk kybernetikk

Jesper Wille Jensen
Jonas Hoel Klüver
Sander Løvdahl Nygård
Tormod Skorpe Skjolden

Terrengfølgende ruteplanlegging for drone

Terrain following path planning for drone

Bacheloroppgave i Elektroingeniør - Automatisering og robotikk
Veileder: Sigurd Gossé
Mai 2023

NTNU
Kunnskap for en bedre verden

MARITIME ROBOTICS

Jesper Wille Jensen
Jonas Hoel Klüver
Sander Løvdahl Nygård
Tormod Skorpe Skjolden

Terrengfølgende ruteplanlegging for drone

Terrain following path planning for drone

Bacheloroppgave i Elektroingeniør - Automatisering og robotikk
Veileder: Sigurd Gossé
Mai 2023

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for teknisk kybernetikk

NTNU
Kunnskap for en bedre verden

NTNU

Oppgavetittel: Terrengfølgende ruteplanlegging for drone. Terrain following path planning for drone.		
Forfattere: Jesper Wille Jensen Jonas Hoel Klüver Sander Løvdahl Nygård Tormod Skorpe Skjolden	**Prosjektnummer:**	E2311
	Innleveringsdato:	22.05.2023
	Gradering:	[x] åpen [ ] lukket
Studium:	Elektroingeniør – BIELEKTRO	
Studieretning:	Automatisering og robotikk	
Veileder internt: **Institutt:**	Sigurd Gossé Institutt for teknisk kybernetikk	
Oppdragsgiver: **Kontaktperson:**	Maritime Robotics, Trondheim Torbjørn Houge, torbjorn.houge@maritimerobotics.com	

Sammendrag:
Maritime Robotics bruker QGroundControl (QGC) til ruteplanlegging for sin VTOL-drone, Falk. QGC ekskluderer mulighet for terrengfølgende ruteplanlegging i store deler av Norge på grunn av manglende datagrunnlag. I tillegg består ruteplanlegging av repetitive oppgaver, og påvirkes av ytre faktorer som vær. Prosjektet tar tak i disse svakhetene med QGC. Terrenghøydedata innhentes fra Kartverkets API for at terrengfølgende ruteplanlegging kan utføres i hele Norge. I tillegg automatiseres repetitive brukeroppgaver; korrigering av terrengkolliderende rutesegmenter, ruten holder spesifisert høyde over terrenget, og korrigering av for bratt stigning i flyrutesegmenter. Til slutt er det lagt til et avansert estimat for flytid som regner med vind og høydeendring i ruten.

Maritime Robotics utilizes QGroundControl (QGC) for path planning for its VTOL aircraft, Falk. QGC excludes terrain-following path planning in large parts of Norway due to a lack of data. Additionally, path planning involves repetitive tasks and is influenced by external factors such as weather. The project addresses these weaknesses in QGC. Terrain elevation data is obtained from the Kartverket's API to enable terrain-following path planning throughout Norway. Automation of repetitive user tasks includes the correction of terrain-colliding path segments, maintaining a specified altitude above the terrain, and adjusting excessively steep climb segments in the flight path. Finally, an advanced flight time estimation has been implemented, which considers wind conditions and changes in altitude along the path.

Stikkord:	**Keywords:**
Programvareutvikling, drone, autonome systemer, databehandling	Software development, drone, autonomous systems, data processing

Sammendrag

Denne oppgaven har som hensikt å undersøke hvordan programmet QGroundcontrol (QGC) kan modifiseres for å effektivisere planleggingsprosessen til Maritime Robotics. QGroundControl er et program med åpen kildekode for ruteplanlegging av autonome droner. Rapporten adresserer problemer med QGC, hvordan disse løses, og resultatet.

Første problem er at QGC ikke henter høydedata for hele Norge inn til programmet. Andre problem er at det kan være tungvint å lage en rute som skal følge terrenget da det er flere aspekter som må tas hensyn til. For å løse disse problemene er det valgt å modifisere kildekoden og videreutvikle programmet ved å bruk av IDEen Qt Creator.

Kartvekets API for høydedata er blitt integrert. Dette sikrer høydedata for hele Norge av høy kvalitet. Samtidig er alle tidligere funksjoner ivaretatt ved å la brukeren kunne velge mellom den originale datadistributøren og Kartverket.

For å lette planleggingsjobben for brukeren er det laget tre funksjoner for effektivisering, ”Auto-avoid”, ”Smooth” og ”Keep dist”. ”Auto-avoid” bruker terrengprofilen laget av høydedata til å detektere kollisjon med terreng, og genererer en rute som unngår kollisjon ved å endre høyder og legge til waypoints. ”Smooth” ser på stigningen i ruten og jevner ut etter gitte begrensninger. ”Keep dist” tilpasser segmentene som ligger for nært terrenget.

Det er også implementert en estimering av flytid med hensyn til vind. Ved å hente inn vindstyrke og retning langs ruten, estimeres en mer realistisk flytid. I tillegg blir bruker om det er for mye vind til å fly.

Resultatet er et program som automatisk henter inn høydedata for hele Norge. Med nye funksjoner som korter ned tiden brukt på å planlegge ruter til autonome droner.

Innhold

Figurer

Tabeller

Kodelister

Akronymer

API application programming interface.

DWD Deutschen Wetterdienstes.

ICON Icosahedral Nonhydrostatic Model.

IDE integrated development environment.

MR Maritime Robotics.

MSVC Microsoft Visual Studio Code.

QGC QGroundControl.

SDLC software development lifecycle.

UA User-Agent.

UAV unmanned aerial vehicle.

USV unmanned surface vehicle.

VTOL Vertical Take-Off and Landing.

Ordliste

Backend Refererer til programapplikasjonens grunnkode. Koden som kjøres i bakgrunnen for å støtte brukergrensesnittet sies å være i bakenden av programmet.

Branch En strukturkomponent til Github. En branch blir en selvstendig kopi av koden den ble laget av.

Build Et build er en ferdig kompilert, kjørbar versjon av et program.

CAD Computer Assisted Design. Data programvare som brukes til konstruksjon og teknisk tegning.

Frontend Refererer til programapplikasjonens brukergrensesnitt og rammeverket rundt dette. Det kan ligge logikk i frontend, men i hovedsak kun i situasjoner hvor det øker programeffektivitet.

GitHub Github er en tjeneste som tilbyr servere hvor brukere kan oppbevare kode, versjonskontrollere, og utvikle software sammen med andre. Github baserer seg på versjonskontrollen fra Git.

Header Filtype i C++ som inneholder kodefilens deklarasjoner.

Heuristikk Er en fremgangsmåte som brukes til å forenkle et problem. Heuristikk kan være nyttig uten å ha basis i teori.

Identifiers er en fellesbetegnelse for funksjoner, variabler, klasser, enum, typer, og kostanter.

Klassemedlem Kodefunksjon som tilhører en klasse.

Kompilator gjør om kildekode til maskinkode.

Locationforecast API fra Norges Metrologiske institutt for værmeldinger.

NaN Står for Not a Number. Ofte brukt som feilverdi i programvare dersom en funksjon returnerer ugyldig data.

parametrisk kontinuitet Begrep for glattheten av en funksjon.

Plug-and-play Betegnelse for teknologi som enkelt kan kobles opp til uten store mengder kalibrering.

plugin Plugins er programutvidelser som enkelt legger til eller redigerer eksisterende kildekode..

Pull Handling i Github. Brukes for å trekke endringer fra en branch inn til seg selv.

Push Handling i Github. Brukes for å dytte endringer fra sin branch til en annen.

RAM (Random Access Memory) er en minneenhet som holder viktig informasjon lagret i korttid. All data lastet inn til enhver tid ligger i RAM. Prosesser som krever mange innlastede filer samtidig krever store mengder minne i RAM.

Repositorium Oppbevaringssted for programvare.

REST API Domene som innehar nyttedata som kan brukes i programvareutvikling for visualisering og utregning.

Runtime Kode som kjøres betegnes som å være i runtime. Objekter og variabler vil i runtime ikke være tilgjengelige fra sine variabelnavn, men heller sine minneadresser. For at kode skal være i runtime må det være kompilert og kjørende.

Tile En tile brukes for en mengde data som beskriver et rektangulært område. Det kan være et kartbilde eller høydedata.

UI På norsk, kalt brukergrensesnitt, er fellesbetegnelsen for alle plattformer brukere kan operere for å interagere med en prosess. Ofte er dette brukt om grafiske bru-

kergrensesnitt (GUI/UI), hvor plattformen er digital og prosessen er en program-applikasjon.

Waypoint er punkter som settes inn i QGroundControl for å bestemme hvor dronen skal fly. Waypoints er nummerert og dronen vil forsøke å fly mellom disse i den nummererte rekkefølgen.

Yr Digital værtjeneste fra Norges Metrologiske Institutt.

Kapittel 1

Introduksjon

1.1 Kontekst

Maritime Robotics (MR) jobber med utvikling av ubemannede luftfartøy, et felt som krever presis data og nøye ruteplanlegging for å kunne gjennomføre flyvninger. Til dette bruker de bakkekontrolprogrammet QGroundControl (QGC). QGC mangler enkelte funksjonaliteter for å gjøre ruteplanlegging effektivt i variert terreng, da programmet i liten grad automatiserer repetitive oppgaver, som ofte forekommer ved lengre ruter. I tillegg har ikke QGC dekning for høydedata i store deler av Norge.

1.2 Oppdragsgiver

MR ble stiftet i 2005 med hovedkontor i Trondheim. De spesialiserer seg innen design og produksjon av ubemannede sjø- og luftfartøy, og er verdensledende i sitt fagfelt. MR har jobbet nært med NTNUs bachelorprogram, og har gjennom årene vært med å fremme et miljø hvor unge nyutdannede ingeniører kan bidra til å videreutvikle dagens marinindustri. Hovedsakelig er de kjent for sin OTTER og MARINER. To ulike unmanned surface vehicle (USV) med modulære Plug-and-play sensorplattformer som innhenter data over og under vann. I tillegg utvikler de andre autonome systemer, slik som FALK, en fixedwing unmanned aerial vehicle (UAV). Å levere produkter som overgår kunders forventning, å fremme trygge og rettferdige arbeidsmiljø, og å hele tiden forbedre me-

todene deres er noen av MRs kjerneprinsipper (Maritime Robotics, 2023).

1.3 Mål

Målet med oppgaven er å utvikle moduler til QGC. Modulene har som hensikt å reduserer arbeidsmengden for operatør. Dette innebærer å implementere tre individuelle moduler. Modul 1 er å implementere høydedata fra Kartverket. Modul 2 skal automatisk korrigere en planlagt rute med hensyn til terrenghøyden. Modul 3 skal estimere flytid med hensyn til vind for en planlagt rute. Gjennom prosjektet jobbes det for å opprettholde god prosjektstruktur og skape et progressivt og trivelig arbeidsmiljø både internt i gruppen og med eksterne veiledere.

1.4 Problemstilling

Kan norsk høydedata integreres i ruteplanlegging for MRs UAV for å oppnå optimal flyhøyde? Videre, kan utvikling av ekstra funksjonalitet effektivisere ruteplanlegging i QGroundControl, med fokus på å fly nært terrenget?

1.5 Prosjektets begrensninger

Det finnes en rekke faktorer som utgjør begrensninger for oppgaven. Primært, betraktes tid som den største begrensningen. Prosjektarbeidet gjennomføres over en periode på 14 uker. I løpet av den tiden gjennomføres også et forprosjekt og en rekke tester. Den gjenværende tiden gir rammene for et stramt tidsbudsjett som i stor grad setter grenser for hvor omfattende arbeidet kan være. Sekundært, er det kunnskapsmessige begrensninger. Samtlige gruppemedlemmer har begrenset erfaring med softwaredesign i C++, som vil kunne hindre både fremgangen og omfanget til prosjektarbeidet. Siste begrensning er at gruppen ikke har dronen FALK til disposisjon, og er derfor ikke istand til å verifisere resultater med testing på en fysisk modell. Disse begrensningene vil kunne påvirke det totale resultatet i stor grad. Det er derfor viktig at de tas med i betraktning

under planleggingsfasen slik at målene som settes er oppnåelige selv med opprinnelig kunnskapsgrunnlag.

1.6 Oppbygging av rapporten

Rapportens hovedelementer er introduksjon, åtte hovedkapitler og en konklusjon. Hovedkapitlene dekker hele utviklingsprosessen. Hensikten med mange hovedkapittel er at mottaker enkelt skal kunne finne frem til informasjon av interesse. Likevel er kapitlene i rapporten strukturert i kronologisk rekkefølge, tilpasset for mottaker som ønsker å reprodusere resultatene.

Kapittel 1: Introduksjon

Kapittel 1 dekker grunnleggende detaljer ved oppgaven, bakgrunn til gruppen bakgrunn til oppdragsgiver, problemstilling, begrensninger ved gjennomføring og målet med prosjektarbeidet. Mottaker skal i dette kapittelet få all nødvendig kunnskap for å kunne sette oppgaven i riktig kontekst.

Kapittel 2: Prosjektbeskrivelse

Kapittel 2 presiserer rammene rundt prosjektoppgaven. Her diskuteres bakgrunnen til oppgaven samt hvilke ressurser gruppen har tilgjengelig og hvilke avgrensninger som settes for prosjektarbeidet. Hensikten med kapittelet er å tydeliggjøre oppgavens formål og omfang.

Kapittel 3: Metodikk

Kapittel 3 går over gruppens arbeidsprosess. Her diskuteres overordnede aspekter ved arbeidet knyttet til planlegging og prosjektledelse. Metodikken legger frem detaljer rundt forprosjektet, fasilitering og gruppens valg av utstyr.

Kapittel 4: Programvareoppsett

Kapittel 4 gir innblikk i hvordan man setter opp programvare aktuelt til prosjektarbeidet. Eksempelvis inkluderes det stegvis gjennomgang av oppsettet til Git, GitHub og Qt Creator samt nødvendige komponenter.

Kapittel 5: Programvarearkitektur

Kapittel 5 gjør rede for QGC sin indre programmeringsstruktur, og dekker blant annet relasjoner og samarbeid mellom av klasser. Kapittelet dekker også grunnleggende egenskaper til anvendte kodespråk, og gir innblikk i hvordan sentrale systemer i koden fungerer, og hvilke oppgaver de har.

Kapittel 6: Modul 1 | Innhenting av norsk høydedata

Kapittel 6 legger frem arbeidet gjort rundt modul 1, den første av tre hoveddeler av prosjektarbeidet. Det presenteres relevant teori rundt innhenting av norsk høydedata, samt valgt fremgangsmåte med resultater. Videre, gås det over designvalg rundt pluginarkitektur for implementering av tilleggskode. Avslutningsvis, diskuteres fordeler og ulemper ved fremgangsmåte og valgt løsning.

Kapittel 7: Modul 2 | Autokorrigering av kollisjonskurs

Kapittel 7 dekker arbeidet gjort rundt modul 2. Kapittelet forklarer designprosessen til automatiske operatørverktøy i QGC, og gir grundig innblikk i hvordan funksjonalitetene fungerer. Videre presenteres resultatene og hvorvidt de oppfyller krav og andre ønsker fra oppdragsgiver. Til slutt diskuteres ulike løsninger samt nyanser i alternative design som etter hvert ble valgt vekk.

Kapittel 8: MODUL 3 | Estimer flytid med hensyn til vind

Kapittel 8 dekker modul 3, som omhandler estimering av flytid gitt rute og værdata, og er sist av de tre modulene. Innledningsvis, presenteres relevant teori, før det legges frem øvrige designvalg og overordnet arbeidsprosess rundt deloppgaven. Videre kartlegges

resultatene av arbeidet med diskusjon rundt fordeler og ulemper ved måten oppgaven ble løst.

Kapittel 9: Testing

Kapittel 9 legger frem hvordan arbeidets kvalitet er overholdt gjennom tester. Det beskrives hvordan testene ble designet for å best møte både oppdragsgivers og gruppens behov, og samtidig gjennomføre testene på et grundig vis slik at det kan overholdes en ønsket standard på arbeidet. Kapittelet avsluttes med diskusjon rundt design av testene, samt andre mulige ordninger som ville vært mer hensiktsmessige for effektiv og god testing.

Kapittel 10: Resultat

Kapittel 10 sammenstiller og presenterer samtlige resultat fra modulene. Ny funksjonalitet tydeliggjøres ved å presentere hvor de nye elementene er i brukergrensesnittet og hvilket bidrag de har til effektivisering av terrengfølgende ruteplanlegging.

Kapittel 11: Konklusjon

Kapittel 11 avslutter rapporten, hvilket innebærer å oppsummere resultatene og å vurdere valgte metoder for å oppnå disse. Videre legges det frem forslag og anbefalinger angående videre utvikling. Til sist trekkes rapportens viktigste funn frem, satt i kontekst av problemstillingen.

Kapittel 2

Prosjektbeskrivelse

2.1 Bakgrunn

2.1.1 Tidligere arbeid

MR har ikke jobbet med denne problemstillingen tidligere. Siden FALK er i en utviklingsfase har det ikke vært en prioritet å se på hvordan automatisering kan lette ruteplanleggingen.

QGC har en fungerende måte å hente terrengdata og legge ruter slik at dronen kan følge terrenget. Problemet er at distributøren mangler terrengprofilen til store deler av Norge.

2.1.2 Krav

MR ønsker norsk høydedata levert av Kartverket implementert i QGC. Samtidig er det et ønske om å utvikle skreddersydde funksjonaliteter som automatiserer og effektiviserer ruteplanlegging. Se vedlegg A for original oppgavetekst.

2.2 Ressurser

NTNU stilte med kontorplass. Gruppen hadde dermed et fast arbeidssted gjennom prosjektperioden. Gruppen stilte med eget utstyr i form av PC, skjermer, og lignende.

2.3 Avgrensninger

Prosjektarbeidet eksisterer utelukkende ved utvikling av programvaren QGC. Datainnhenting og funksjonaliteter som utarbeides er kun til nytte innad programmet. Forarbeid for ruteplanlegging eller eventuell virksomhet tilknyttet konfigurering eller synkronisering av fysiske droner regnes dermed som utenfor oppgavens omfang.

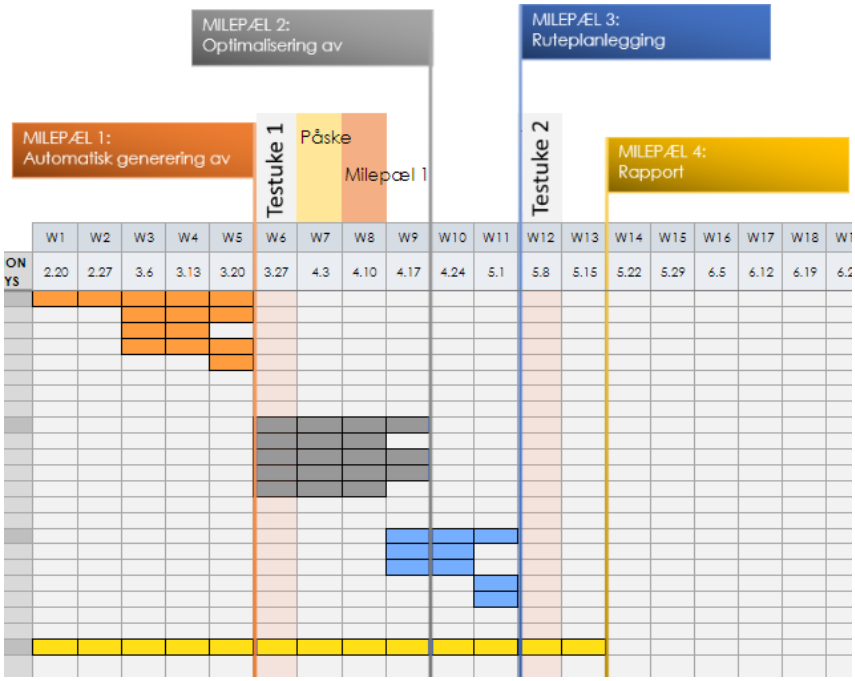
Kapittel 3

Metode

3.1 Forprosjekt

Hensikten med forprosjektet er å planlegge arbeidsflyten gjennom prosjektperioden. Forprosjektet tok utgangspunkt i tre funksjonaliteter spesifisert av oppdragsgiver. Disse funksjonalitetene var å anse som moduler som gruppen skulle jobbe mot. Modulene ble prioritert basert på ordlyden i oppgaveteksten og hvilke funksjoner som var viktigst for oppdragiver.

Modulene ble stykket opp i flere arbeidspakker. Dermed kan gruppemedlemmene jobbe uavhengig av hverandre. For oversiktens skyld ble det også satt opp et Gantt-diagram som inneholdt viktige frister i prosjektperioden. Figur 3.1 viser Gantt-diagrammet.



Figur 3.1: Ganttdiagram som viser planlagt arbeidstid per modul i prosjektperioden (Nygård mfl., 2023).

3.1.1 Arbeidsfordeling

Prosjektets resultat er programvare. Gruppemedlemmenes erfaringen innen programvareutvikling er variert, men ingen hadde erfaring med kodebase av denne størrelsen før prosjektstart. Det er derfor viktig med en tydelig arbeidsfordeling. Gruppen laget arbeidspakker slik at alle kunne jobbe uavhengig av hverandre og mot frister basert på individuelle behov. Totalt ble modulene partisjonert i 16 arbeidspakker fordelt mellom gruppemedlemmene.

3.1.2 Tidsplan

Fra prosjektstart til prosjektslutt skal tre moduler og en rapport fullføres iløpet av 13 uker. Modulene består av flere arbeidspakker med ulike frister, hvor alle arbeidspakker er separate dokumenter. Et Gantt diagram er opprettet for å ha god oversikt over progresjon, se figur 3.1). Diagrammet visualiserer tidsfrister for alle arbeidspakker, hver

modul, ferier, og testuker. Gruppemedlemmene har dermed oversikt over arbeidsflyten, og kan bruke diagrammet for å kartlegge progresjonen.

På grunn av manglende erfaring innen programvareutvikling, satt gruppen en hard frist for utvikling av modul 1. Omfanget av modulen var ukjent i og det er vanskelig å verifisere hvor tidbegrensede arbeidspakker var realistiske og oppnåelig. Derfor ble det satt en hard frist for stans av utvikling ved starten av uke 8. Dersom modul 1 ikke var ferdig innen da, ville arbeidet tilknyttet modulen avsluttes til fordel for å dokumentere fremgangen gjennom rapportskriving.

3.2 Fasilitering

3.2.1 Prosjektledelse

En prosjektleder oppnevnes for å holde styr på fremgangen i prosjektet. Hele gruppen får mulighet til å leder arbeidet ved å definere en rullerende prosjektlederrolle. Hvert medlem er prosjektleder i to uker av gangen før neste rullering. Prosjektlederens oppgaver er å initiere møter, gi ordet under møter, samt å skrive toukersrapport som tilsendes oppdragsgiver og veileder.

3.2.2 Waterfall SDLC

Waterfall software development lifecycle (SDLC) er en klassisk metode innen programvareutvikling. For grupper med lite erfaring innen programvareutvikling er Waterfall SDLC intuitiv. Den er lineær og forutsigbar. Det vil si at man jobber med en fase av gangen. Metoden består av seks faser; Kartlegging av krav, design, programmering, integrasjon og testing, utplassering, og vedlikehold. Oppdragsgiver spesifiserer sitt krav om å automatisere deler av ruteplanleggingen i QGC. Deretter fordeles arbeidspakker mellom gruppemedlemmene. Ved design av de ulike funksjonalitetene kom gruppen til enighet om å ha kreativ frihet til å programmere, så lenge det samsvarte med strukturen i kildekoden.(W3schools, u.å.)

3.3 Utstyr

3.3.1 Qt Creator

Qt Creator er en integrated development environment (IDE) spesialdesignet til bruk av utvikling innen Qt. IDEen støtter blant annet C++, JavaScript og QML og inneholder en rekke verktøy som støtter utviklere i arbeidsprosessen. Eksempler på slike verktøy er kodeeditor med syntaksutheving, feilretting, innebygd debugger og støtte for versjonskontrollsystemer som Git (Qt-Group, u.å.-a). Grunnen til at Qt Creator brukes til fordel for "Microsoft Visual Studio Code (MSVC)" er fordi QGC er bygget på Qt-rammeverket som Qt Creator tilbyr skreddersydd støtte til. QGC bruker enkelte "identifiers" som ikke er standard C++. Qt Creator har kjennskap til disse, og tilbyr debugging og andre verktøy for å håndtere dem. Noen eksempler er at Qt-rammeverket bruker en egen nettverksprotokoll, og identifiers som *QString*, *QList* istedet for standard *string*, *list*. Dette forklares dypere i kapittel 5.2.1.

3.3.2 QGroundControl

"QGC" er åpen kildekode programvare som brukes til å styre ubemannede droner. Det tilbyr et intuitivt brukergrensesnitt som gjør det enkelt å gjennomføre ruteplanlegging, samt overvåke sensordata i flytiden. QGC er kompatibel med alle autopilot-systemer som bruker kommunikasjons protokollen MAVLink (QGroundControl, u.å.-d). QGC er tilgjengelig for Windows, Linux og macOS, og under kontinuerlig utvikling av et fellesskap av utviklere og brukere. QGC er programvaren som MR bruker til å planlegge flyruter.

3.3.3 GitHub

"GitHub" er verktøyet for teamutvikling av QGC. GitHub gjør det mulig å utvikle på individuelle "build", samtidig som det gir mulighet for sammenfletting av individuelle endringer til en felles "Branch" ved å gjøre en "Push"-operasjon. Brukere som ønsker å hente inn endringer fra branchen til sitt personlige build, utfører en "Pull"-operasjon. Slik bestemmer hvert gruppemedlem selv hvilken versjon av programvaren de ønsker å

jobbe med. Alle endringer i branchen føres i en historielogg, som gjør det enkelt å ha kontroll på hvert bidrag fra teamet.

3.3.4 Miro

Miro er en digital samhandlingsplattform for teambaserte prosjekter, og ble under utvikling brukt til å lage diagrammer som beskriver kodefunksjonalitet. Kodebasen til QGC er stor og omfattende, og for uerfarne utviklere er det vanskelig å forstå sammenhengen mellom klasser, backend og frontend. Dette gjør Miro nyttig, da det kan brukes i forkant av programmering, og gi overordnet forståelse av hvordan programmets elementer jobber sammen.

3.3.5 GeoGebra

GeoGebra er en digital kalkulator med mulighet for visualisering av matematiske uttrykk. I prosjektet brukes det til visualisering av terreng, flykarakteristikk, flyruter, og generelt til utvikling av matematisk fremgangsmåte. Funksjoner som skal bli implementert i QGC, kan simuleres gjennom agil testing og dermed verifiseres før implementering. Programmet har også blitt brukt til generering av grafikk for deler av rapporten.

3.3.6 Teams

Teams blir brukt som en kommunikasjonsplattform og fil-lagringsystem. I tilfeller hvor det er nødvendig med møte med veileder eller oppdragsgiver blir Teams benyttet. Eksterne veiledere gis tilgang og slik at de har innsyn i all dokumentasjon. Forprosjekt, Gantt diagram, interne grupperegler, timelister, møteinnkalling og referat, tester, nyttige nettsider er alle eksempler på dokumentasjon som ligger i teamskanalen.

Kapittel 4

Programvareoppsett

4.1 Introduksjon

QGC kan lastes ned som et ferdig build fra deres egne integrerte installeringsprogramvare som ligger på QGC sin nettside. Alternativt kan den klones rett fra GitHub. I prosjektet gjennomføres sistnevnte metode fordi programvaren skal modifiseres. Til utvikling brukes IDEen Qt Creator. QGC legger opp til at brukere skal ha muligheten å tilpasse programvaren etter eget behov. Av denne grunn tilbys en developer guide og et forum som letter installasjon og utvikling.

4.2 Teoretisk rammeverk

4.2.1 Git

Git er et desentralisert versjonskontrollprogram designet av Linus Torvalds, skaperen av operativsystem Linux. Git brukes til programvareutvikling og er spesielt godt egnet til ulineær utvikling fordi det støtter avgrening og sammenslåing av grener i samme prosjekt. Den har også støtte for historikk av endringer og med mulighet for å enkelt gjenopprette tidligere versjoner av prosjektet dersom det skulle være ønskelig. Git er dermed designet for å være egnet til bruk i distribuerte utviklingsmiljø der mange små endringer skjer på forskjellige maskiner fra samtlige deltagere i utviklingen (Git, u.å.).

4.2.2 GitHub

GitHub er en nettstedshosting-tjeneste basert på det desentraliserte versjonskontrollprogrammet Git. Siden GitHub er basert på Git har det alle fordelene og egenskapene til Git, men legger da til flere tjenester utover dette. Den viktigste av disse er å fungere som en skylagringstjenste for filene til de forskjellige prosjektene. En annen funksjon som GitHub har lagt til heter ”fork”-ing og gir muligheten til å kopiere et helt prosjekt over i en ny mappe slik at den kan jobbes videre på som et nytt prosjekt istedenfor en variant av det eksisterende prosjektet.

4.2.3 QT

Qt, uttalt ’Cute’, er et rammeverk for design av kryss-platform programvareapplikasjoner med ett sett med kildekode. Det vil si at utvikleren kan skrive programmet uten å trenge å ta hensyn til å tilpasse det til de forskjellige platformene det skal kjøres på, da Qt ordner det automatisk. Blant funksjonaliteten inkludert i rammeverket er SQL database tilknyttning, parsere til XML og JSON filformatene, trådhåndtering til flertrådede programmer, og nettverkshåndtering. (Qt-Group, u.å.-b)

4.3 Metode og utstyr

4.3.1 Developer guide

QGC har en egen developer guide. Den dekker hvordan programmet installeres i operativsystemene Windows, MacOS, Linux eller Ubuntu. Videre dekker guiden hvilken versjon av IDEen, Qt Creator som trengs.

Developerguiden gir innblikk i designfilosofien rundt hvordan QGC er laget, hvilke programstrukturer som brukes, hvilke problemer utviklerne er klare over og hvordan det er tilrettelagt for modifisering av programmet. I guiden forklares konsepter som kan være til ettertanke. Det er blant annet oppgitt konvensjoner for private utviklere å følge ved navngiving av modifiserte versjoner.

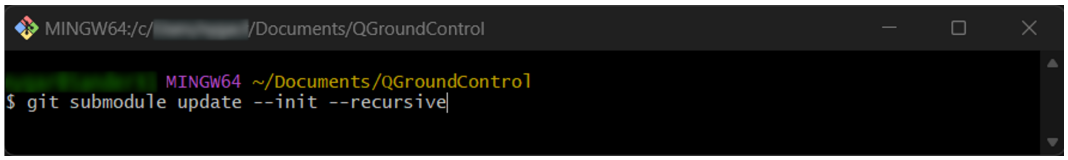
4.3.2 Git bash

Git Bash er et kommandovindu, lik terminal eller command prompt som kommer som standard med Windows PCer. Git lastes ned for å få tilgang til Git Bash. Kommandovinduet brukes til å klone et ”repositroium” som inneholder koden for å bygge QGC.

1. Last ned windowsversjonen av Git fra nettsiden deres . Når nedlastningen er fullført, opprettes en mappe og kall den QGroundControl.
2. Åpne Git Bash i den nyopprettede mappen. Klon QGC ved å skrive inn følgende i terminalen.

Figur 4.1: Kommandoen kloner programfilene og plasserer dem i ønsket mappe.

3. Initialiser og oppdater undermoduler i prosjektet. Dette sikrer at alle nødvendige ressurser er tilgjengelige og oppdaterte, noe som bidrar til enklere samarbeid og sikrer kodeintegritet. (QGroundControl, u.å.-a)



Figur 4.2

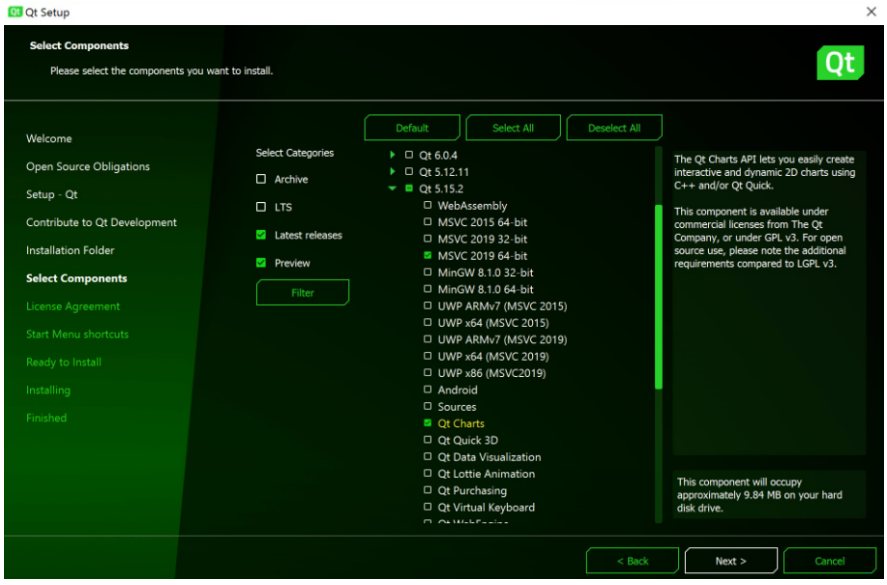
4. Programvaren QGC er nå plassert i en egenopprettet mappe som heter QGroundControl (Griffins-Gadgets, 2020).

4.3.3 Last ned kompilator

Selve programvaren QGC bygges i Qt Creator, men IDEen trenger en ekstern Kompilator. Kompliatoren installeres ved å laste ned MSVC.

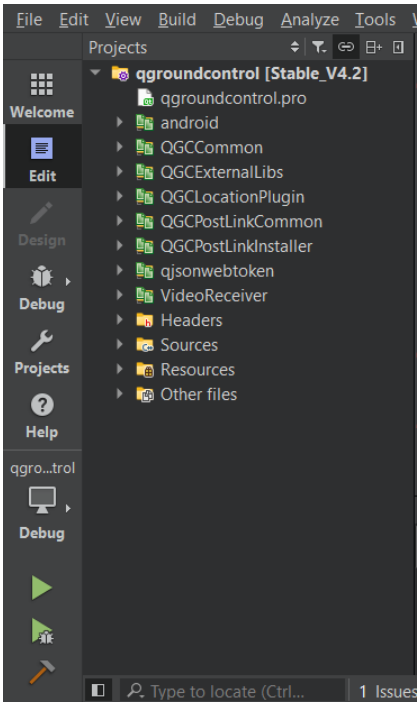
4.3.4 Last ned Qt Creator

1. Last ned og kjør Qt installer.
2. I *Select component* velges versjon Qt 5.15.2. Når denne velges dukker det opp en del valgmuligheter. Huk av boksene for kompilatoren *MSVC 2019 64 bit* og *Qt Charts*.



Figur 4.3: hentet fra instruksjonene å sette opp utviklingsmiljø (QGroundControl, u.å.-b)

3. Åpne Qt Creator, lokaliser QGroundControl-mappen og finn *qgroundcontrol.pro* for å åpne prosjektet. Trykk deretter på den grønne trekanten nederst i ventre hjørne for å bygge og kjøre QGC.
   Prosjektet kan så kjøres enten ved å builde (trykke på hammeren), eller å kjøre programmet i debug mode (trykke på den grønne pilen like over hammeren).



Figur 4.4: Prosjektet er åpnet i QT Creator

4.3.5 GitHub

For å kunne drive med versjonskontroll og effektivt gjøre endringer til det opprinnelige buildet av QGC er det nødvendig å sette opp et GitHub repositorium for dette (QGroundControl, u.å.-c)(GitHub, u.å.).

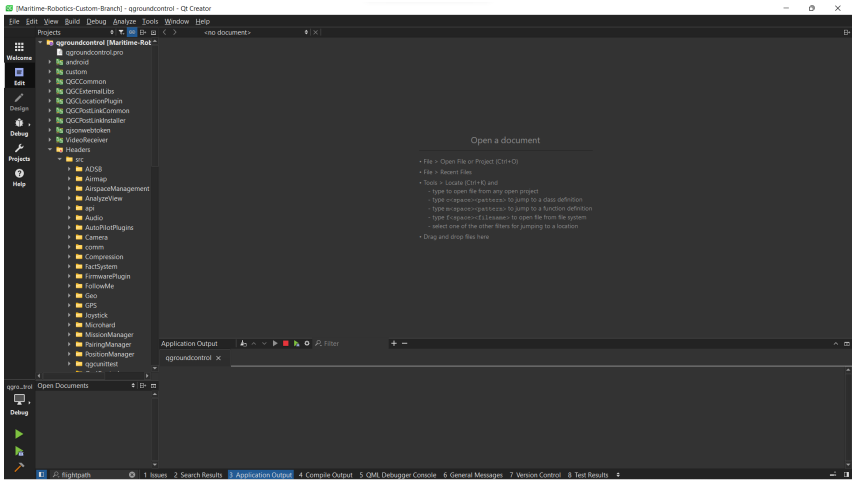
1. Logg inn på GitHub
2. Trykk på brukerikonet øverst til høyre. I menyen som dukker opp, trykk på 'Your repositories'.
3. I det nye nettleservinduet, velg den grønne knappen der det står 'New'.
4. Legg inn et navn på prosjektet. I forbindelse med dette prosjektet ble det valgt å sette prosjektet til å være 'private', men dette er ikke nødvendig. Ikke legg til andre filer i menyvalgene lengre ned. Opprett repositoriet med knappen 'Create repository'.
5. Nederst i det nye vinduet som åpnes er det et menyvalg der det står '...or import code from another repository'. Velg denne.
6. Legg inn URLen til QGCs GitHub repositorium og trykk 'Begin import'. Denne prosessen kan ta veldig lang tid, så det er her mulig å ta en kort pause ettersom

det blir sendt en email til brukeren når prosessen er ferdig.

7. Når denne prosessen er ferdig, er GitHub repositoriet opprettet og konfigurert. For å hente ut linken til dette repositoriet slik som mange andre programmer ønsker kan dette hentes ut ved å trykke på den grønne knappen markert med 'Code' og kopiere linken derfra.
8. Andre deltagere kan gis tilgang til GitHub repositoriet slik at de kan bidra til koden direkte. Dette gjøres ved å gå inn på 'Settings', 'Access', og så legge til GitHub brukerene til de det er ønskelig skal ha tilgang.

4.4 Resultater og empiriske funn

Dersom alle stegene er fulgt og det ikke har oppstått noen feil underveis, skal prosjektet være åpnet i QT Creator. Koden kan gjøres endringer på som ønsket, og kjøres ved bruk av knappene vist i figur 4.4.



Figur 4.5: Bilde av hvordan QGC ser ut i project view i Qt Creator.

Ved første build, vil programmet ta ekstra lang tid, da ingen buildfiler har blitt laget enda. I tillegg vil det oppstå en rekke varsler ved første build, men de kan ignoreres. Ved senere build, vil kompileringstiden variere avhengig av mengden endringer som har blitt gjort. Endringer i C++ tar typisk lengre tid å kompilere, mens endringer på QML-siden

kompileres raskt.

Etter fullført kompilering, vil et build av QGC åpnes og være klart til bruk.

4.5 Analyse og diskusjon

Utfordringer ved oppsett

Under oppsettet av dette oppsto det noen problemer innad gruppa. Den største utfordringen oppsto da en av gruppemedlemmene hadde valgt feil kontotype ved oppsett av brukeren sin hos QT. Dette gjorde at de ikke fikk lastet ned rett type Qt Creator, samt at versjonen de fikk var en prøveperiode på en uke. Da dette ble funnet ut av, måtte brukeren slettes og lages på ny. Alt av programvare måtte lastes ned på ny og rekonfigureres for å passe med GitHub. Gjennom hele denne prosessen sluttet også git å fungere, hvilket ble løst ved å laste ned en nyere versjon av git.

4.6 Kapittel konklusjon

Nedlasting av QGC krever en oppdatert versjon av Git. GitHub lastes ned og konfigureres til nedlasted QGC-build. Qt lastes ned sammen med Qt creator for open source utvikling. Prosjektet åpnes i Qt Creator og konfigureres til riktig kompilator. Qt Creator er dermed installert og GitHub-repositoriet er klonet inn på egen maskin. Da er det klart for utvikling og modifisering av programvaren.

Kapittel 5

Programvarearkitektur

5.1 Introduksjon

QGC er bygget opp slik at backend, kode som brukeren ikke ser, er programmert i C++. Her bygges alle objekter med sine funksjoner som gjør at programvaren fungerer. I frontend konstrueres et brukergrensesnitt, som gjør at brukeren kan sette i gang eller bruke funksjoner til å gjennomføre ruteplanlegging. Frontend er laget i QML. For å kunne modifisere dette programmet kreves en god forståelse av hvordan disse elementene henger sammen. Gruppens forståelse av programstruktur og forbindelsen mellom frontend og backend presiseres og i dette kapittelet.

5.2 Teoretisk rammeverk

5.2.1 Qt

Qt er et kryssplattform rammeverk som lar utviklere skape programvareapplikasjoner med grafisk brukergrensesnitt. Qt tilbyr en rekke verktøy slik som data- og nettverkshåndtering, 2D- og 3D-grafikk samt presis og enkel kontroll over ”UI”. Eksempler på dette er makroer Q_Property og Q_Invokable, som knytter sammen metoder og medlemmer mellom C++ og QML, grundigere dekt i kapittel 5.2.5.

Qt samler utvikling på ett sted i kjente omgivelser. Software bygget i Qt kan skrives på en rekke programmeringsspråk, inkludert Python, C++ og JavaScript. Applikasjonen kan raskt portes til andre plattformer Qt også tilbyr støtte for.

Qt har en rekke egne datatyper som sømløst kobles mellom ”UI” og ”Backend”, for eksempel QString, QVariant, QList, QDateTime og QGeoCoordinate, med innebygde funksjoner for dataprosessering og konvertering mellom dem.

5.2.2 C++

C++ er et objektorientert programmeringsspråk og er et av verdens mest populære språk å skrive applikasjoner med. C++ brukes ofte til å skrive programmer fordi det er veldig raskt sammenlignet med mange andre programmeringsspråk, slik som Python, og fordi det er relativt enkelt å utvikle programmer i. I kontekst av Qt og QGC brukes det for å skrive all kompleks logikk og for å holde styr på alt av data i programmet. Filene til C++ er delt i to deler, en ”Header”-fil med suffiksen ’.h’ og en kildekodefil med suffiksen’.cc’. Formålet med Header-filen er å deklarere hvilke variabler og objekter en kodefilen har tilgjengelig. Andre kodefiler kan bruke variabler og objekter deklarert av andre filers Headere.

5.2.3 QML

QML (Qt Modeling Language) er et programmeringsspråk brukt til design og modellering av brukergrensesnitt til applikasjoner. Det har mange likhetstrekk med både CSS og JavaScript, som brukes til de samme formålene som QML. CSS og JavaScript er også deklarative programmeringsspråk på lik linje med QML. QML kan brukes til å programmere enkle funksjoner, men det er intensjonen at all databehandling gjøres i et annet programmeringsspråk mer egnet til indre programstruktur. I QGCs tilfelle tar C++ den rollen.

5.2.4 Signal/Slot

Signal og slot er en del av QTs egne struktur. Signaler sendes av objekter ved tilstandsendring. Slots er funksjoner som kjøres når et signal blir sendt. Signaler kan sende variabler til slots. Signals og slots kobles sammen ved å bruke funksjonen "connect()"(Qt-Group, u.å.-c). Signalet kan kobles til andre objekter sin slot. Dette gjør det mulig for objekter å kommunisere seg i mellom.

5.2.5 Q_Property & Q_Invokable

Grensesnittet kjørt av QML trenger et grensesnitt for å kunne kommunisere med C++koden under panseret. Q_Property og Q_Invokable er syntaks som gjør klassemedlemmer og klassevariabler tilgjengelig for QML-objekter. En klassevariabel i C++ kan defineres som en Q_Property med et alias som QML kan nå. Med Q_Property satt opp riktig, vil alle QML-objekter med tilgang til klassen, også ha tilgang til interne variabler deklarert som Q_Property. Qt har i tillegg inkorporert signal/slot-systemet inn i Q_Properties slik at signaler kan sendes ved endring av data.

Dersom et objekt fra QML skal ha tilgang til et klassemedlem, kan medlemmet klassifiseres som Q_INVOKABLE ved deklarering i header-filen. Eksempler på disse er vist under:

```
46    // PlanMasterController.h
47    Q_PROPERTY(bool    dirty   READ   dirty     WRITE   setDirty    NOTIFY dirtyChanged)
```

Kodeliste 5.1: C++ Q_Property deklarasjon

```
1144    // CustomPlanView.qml
1145    QGCLabel {
1146    id:                 unsavedChangedLabel
1147    Layout.fillWidth:   true
1148    wrapMode:           Text.WordWrap
1149    text:               globals.activeVehicle ?
1150    qsTr("You have unsaved changes. You should upload to your vehicle, or save to a file.
        ") : qsTr("You have unsaved changes.")
1151    visible:            _planMasterController.dirty
1152 }
```

Kodeliste 5.2: QML Q_Property aksessering

```
58    // PlanMasterController.h
59    Q_INVOKABLE QString     flightTime(QString);
```

Kodeliste 5.3: C++ Q_Invokable deklarasjon

```
1492  // CustomPlanView.qml
1493  QGCLabel {
1494  var eftStr = _planMasterController.flightTime(_appSettings.inputDateTime.value)
1495  mainWindow.showMessageDialog(qsTr("Flight time estimate at time: "+_appSettings.
      inputDateTime.value), qsTr(eftStr), StandardButton.Ok, function() {})
1496  }
```

Kodeliste 5.4: QML Q_Invokable aksessering

5.2.6 Fact system

QGC anvender i stor grad et egetutviklet system for håndtering av variabler og parametre, en egen klasse kalt ”Fact”. Factsystemet gir kontekst til tallvariable med bruk av enheter, minimumsverdier og default values.

Factsystemet tillater parametre å lagres lokalt på Json-filer slik at programmet bruker tidligere verdier ved neste oppstart. Factsystemet integreres sømløst med QML, da det er utviklet egne tekstbokser for avlesing og endring av facts.

Facts kan initialiseres hvor som helst, og er eksempelvis brukt hos *appSettings*, *flightMapSettings* og *visualItems* (kap 7.2.1). Se kodeliste 5.5.

```
151   // AppSettings.cc
152   DECLARE_SETTINGSFACT(AppSettings, currentDate)
153   DECLARE_SETTINGSFACT(AppSettings, currentTime)
154   DECLARE_SETTINGSFACT(AppSettings, inputDateTime)
155   DECLARE_SETTINGSFACT(AppSettings, userAgent)
```

Kodeliste 5.5: Settingsfact deklarasjon

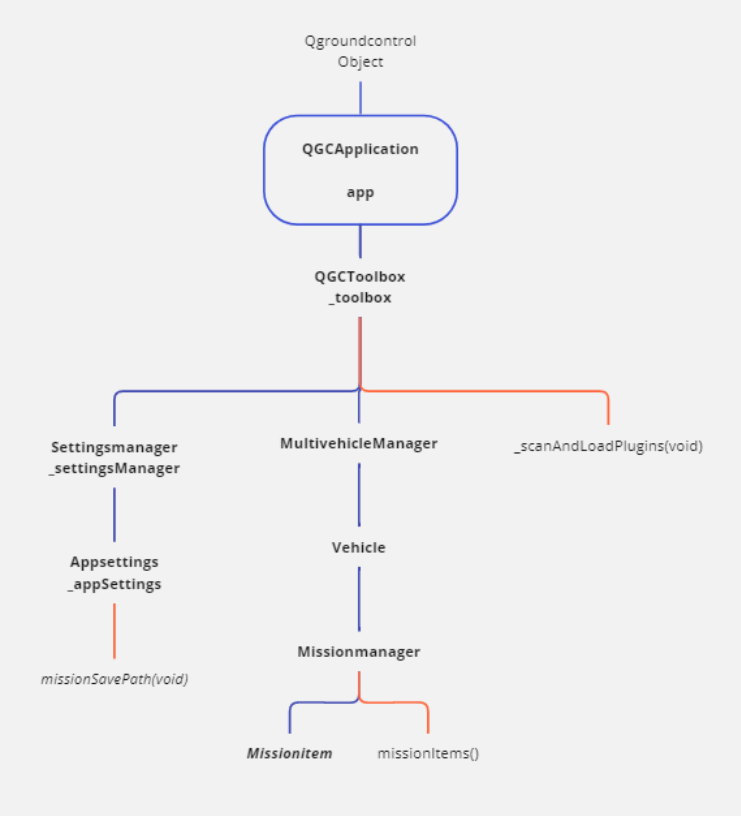
5.3 Metode og utstyr

5.3.1 Struktur

For administrering av de forskjellige komponentene som utgjør programmet brukes manager-objekter. Disse objektene opprettes ved oppstart av programmet. Som eksempel finnes terrain tile manager. Denne blir da brukt til å administrere objektene terrain tile. Figur 5.1 visualiserer hvordan hierarkiet med managere som kan inneholde undermanagere er bygd opp.

Alle disse objektene trenger en måte å kommunisere og utveksle data på. I QGC er signals og slots mye brukt. Det er en veldig enkel og intuitiv måte å dele data mellom objektene på. Her kan ett signal kobles til flere slots, eller så kan ett slot være koblet til flere signaler. QGC bruker dette blant annet mye til å trigge oppdateringer ved verdiendringer.

For å bedre informasjonsflyt, er det anvendt et factsystem for lagring og henting av preferanser, innstillinger og parametre. Dette factsystemet er mye brukt for kommunikasjon mellom brukergrensesnittet og grunnkoden. Facts har blant annet *'default values'* som hindrer feil i programmet ellers kunne støtt på i situasjoner hvor QML prøver å lese av en udefinert verdi.



Figur 5.1: Dette er en ufullstendig representasjon av manager-hierarkiet, men visualiserer hvordan det er en overordnet manager som inneholder mer spesifikke managere lengre ut i forgreiningen

5.3.2 Frontend og backend

I "Backend" administreres blant annet kontrollere for kjøretøy, kartdata, flyruten osv. Det er essensielt å utveksle data mellom QML og C++. Q_Property og Q_Invokable muliggjør kommunikasjonen mellom Frontend og Backend. Funksjonalitetene som legges til under prosjektarbeidet krever inntasting av verdier, nye knapper med handlinger og endring i innstillinger. Dette legger brukeren inn i brukergrensesnittet. Disse handlingene må overføres til Backend. Bruken av disse metodene for utveksling av data er derfor helt sentralt under prosjektarbeidet.

5.4 Resultater

Innvendig, er QGC satt opp med hierarkisk objektstruktur i C++ hvilket bidrar til en ryddig indre programstruktur samt rask og oversiktlig informasjonsflyt mellom objekter. Med qgcApp og videre QGCToolBox i roten, brer en rekke managere seg ut og etablerer basen for hvert sitt område. Eksempler av disse er *SettingsManager*, *MultiVehicleManager* *VideoManager* og *PositionManager*. Siden qgcApp er offentlig tilgjengelig for alle objekter, kan ethvert objekt innen hierarkiet nås fra hvor som helst. Alt man trenger for å legge til eventuell funksjonalitet, er å legge til en klasse et passende sted i koden.

Brukergrensesnittet er programmert i QML, og benytter seg også av hierarkisk objektstruktur, lignende C++siden. Objekter i QML aksesserer C++-objekter gjennom Q_Properties, og klassemedlem gjennom Q_Invokables. Q_Properties har i tillegg innebygd støtte for signaler for å sikre synkronisering på tvers av programmeringsspråk.

Factsystemet gir en robust måte å håndtere parameter og verdier og deres metadata.

5.5 Analyse og diskusjon

Innledningsvis, blir arkitekturen mer oversiktlig desto mer av programmet man får kartlagt. Fra et utenforperspektiv, kan det ta tid å finne et konkret punkt i koden å legge utvidelser til. Dersom man har erfaring med anvendelse av objektorientert programmering i større softwaredesign, er en hierarkisk tilnærming naturlig for valg av programstruktur. Tilnærmingen tillater enkel videre utvidelse av programmet, da man kan lage egne moduler hvor man skulle ønske i hierarkiet.

En viktig faktor for oversiktlig kode er valg av klasse-, medlem- og variabelnavn. Her kunne QGC lagt et bedre grunnlag, da mange klasser har svært like navn uten en overordnet navngivningsstandard. I koden er det en rekke klasser med lignende navn, som for eksempel *PlanCreator*, *PlanManager*, *PlanMasterController* og *PlanElementController*. Dersom QGC hadde konvensjoner for hvordan slike navn tildeles, kunne det vært mer tydelig hvordan disse jobber sammen. Mangelen på slike konvensjoner krever at nye utviklere må analysere sammenhengen mellom alle slike klasser individuelt.

Under ”runtime”, vil alle QGCs nødvendige manager-klasser og delklasser bli skapt etter

behov. Mange av disse klassene blir skapt tidligere enn de trengs. Dette kan være krevende for datamaskinen programmet kjører på, da mange objekter må lastes inn og være tilgjengelige i ”RAM”-minnet samtidig. Denne ordningen er ikke særlig effektiv, da det lastes inn mange deler av programmet som muligens ikke skal være i bruk ennå. Med slik programstruktur, minker skalerbarheten drastisk, da minnet brukes opp på deler av programmet som ikke trengs å lastes inn. Ved utvidelser av programmet, må dette tas med i betraktning, slik at det ikke skapes flere objekter enn nødvendig. Samtidig, må tilleggsmoduler designes med god utnyttelse av minnet, slik at programmet kan opprettholde god ytelse, selv på datamaskiner med mindre kapasitet.

5.6 Kapittel konklusjon

I sin helhet er programmet stabilt, og legger til rette for utvidelse på et oversiktlig vis. Når en setter seg ned og skal begynne å modifisere koden er det mye å ta innover seg. Men det blir klarere etterhver som man får en bedre forståelse av programmet. Den gode strukturene hjelper helt klart på å forstå koden fortere.

Kapittel 6

MODUL 1: Innhenting av norsk høydedata

6.1 Introduksjon

Planlegging av rute til autonome droner krever en viss kjennskap til terrenget det flys over. QGC har innebygde funksjoner som bruker terrenghøyden til å automatisk sette høyden til ”waypoints”. Programmet bruker Airmap som standard innstilling til distributør for terrenghøyde. Denne distributøren har en høydemodell som kun går mellom 56°sør og 60°nord (Airmap, 2023). Med denne konfigurasjonen klarer ikke programvaren å hente inn høydedata nord for 60°. Trondheim ligger på cirka 63°nord, og er utenfor dette området. MR har derfor et behov for å integrere en ny høydemodell i QGC som har data for hele Norge. En mulig kilde Kartverkets høydemodell. Denne passer bra da den har et åpent API. I tillegg har denne datakilden en god oppløsning på 1 meter (Kartverket, 2023). Denne modulen har som hensikt å integrere Kartverkets API inn i programmet på en slik måte at de innebygde funksjonene bruker data herfra i stedet for Airmap, og ivaretar sin funksjonalitet. I tillegg skal muligheten til å bruke Airmap sine datakilder bevares. Det legges derfor inn en mulighet for å bytte distributør av terrenghøyde. Implementeringen forsøker best mulig å følge plugin-arkitekturen som er det har blitt tilrettelagt for, av QGC.

6.2 Teoretisk rammeverk

6.2.1 API

Når QGC henter høydedata fra Kartverket bruker den et ”application programming interface (API)”. Et API er en mekanisme som lar to individuelle systemer snakke med hverandre gjennom et sett definerte regler (Amazon, u.å.). Dette gjør det mulig hente data fra eksterne kilder over internett. En av Kartverket sitt API sine funksjoner er å sende en liste med opp til 50 koordinater. Deretter returner den en JSON-fil med data om de forespurte punktene, deriblant terrenghøydene (Kartverket, 2023).

6.2.2 Hash-indeks

Om en liste blir lang vil det ta mye tid å iterere seg gjennom den. For å unngå dette kan en hash-funksjon brukes. Hashing er å ta en bit informasjon og gjøre den om til en adresse til data som lagres (Zhenov, 2021). Adressen gjør at en kan gå rett til riktig plass, heller enn å bruke lang tid på å iterere gjennom en listen.

6.2.3 Cache

Cache er et minne der ofte brukt data blir lagret. Istedenfor å sende samme forespørsel til en server flere ganger, lagres heller data lokalt slik at det er lettere tilgjengelig. Dette kan øke hastigheten til programmet, da det er mye raskere å lese av data fra lokalt minne. I tillegg reduserer det trafikken til serveren, som bidrar til at serverforespørslene blir raskere de gangene man trenger dem. (Bratbergsengen & Nätt, 2022).

6.2.4 Inkludering og ekskludering av filer

For at programmet skal vite hvilke filer som skal inkluderes når det kompileres, blir disse definert i forskjellige filer. Filer programmert i C++ bruker filtype ’.pro’ og ’pri’, mens bilder og QML-filer bruker filtype ’.qrc’.

For bilder og QML-filer oppgis det to data, filbane og alias. Alias er navnet som brukes for

å referere til filen i den gitte filbanen. For C++ filer inkluderes i '.pro' og '.pri' filer. C++ filer inkluderes i programmet ved å bruke '+=' operatoren og filbanen til filen under 'HEADERS' og 'SOURCES'.

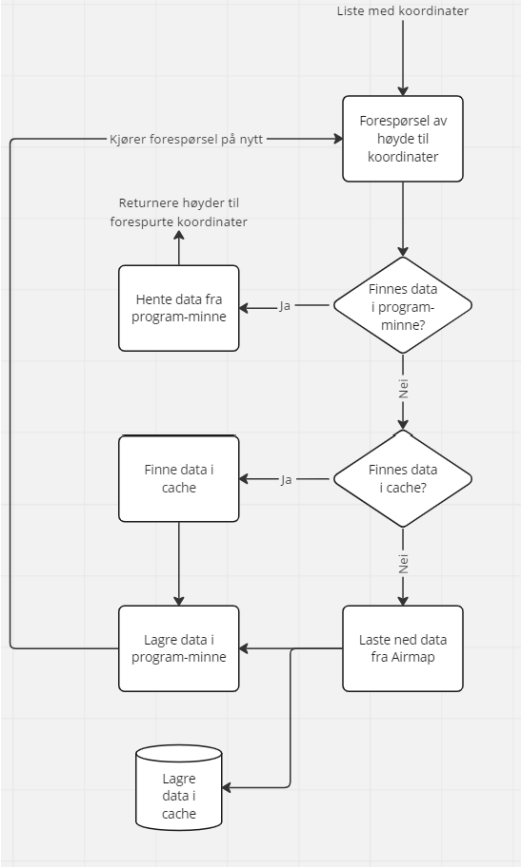
6.3 Metode og utstyr

6.3.1 Hente høydedata

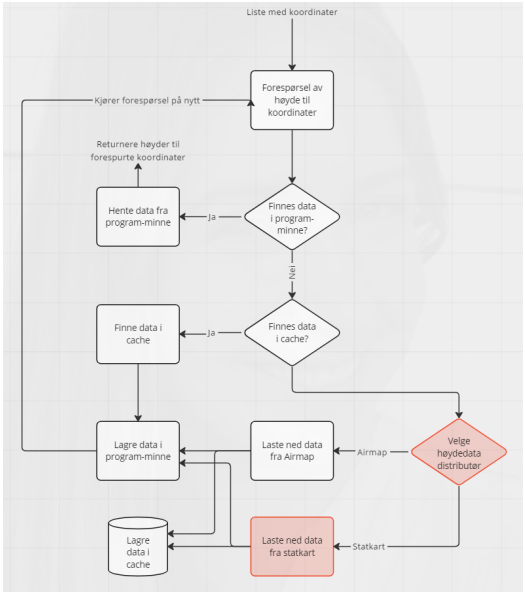
For å skjønne hvordan den nye distributøren skal integreres må det være klart hvordan den eksisterende fungerer. Ved å imitere måten den allerede er bygd opp på blir eksisterende funksjoner ivaretatt. I tillegg ville endringer kunne ha uforutsette konsekvenser da det er tidkrevende å sette seg i hele koden. Derfor skal den nye koden være så lik den gamle som mulig. Implementering av ny kode vil forsøke å være adskilt fra kildekode. Det gjøres ved å heller lage nye funksjoner enn å modifisere gamle. Da er det enklere å gå tilbake til hvordan koden var originalt.

Figur 6.1 viser hvordan dataflyten går originalt. Data kan komme fra tre forskjellige steder: program-minne, cache, eller fra internett. Program-minne er et array som opprettes når programmet startes. Først vil den sjekke om høyden som etterspørres ligger her. Om den ikke gjør det, sjekkes så cache-minne. Dette er en database som er lagret på disk. Om det finnes her hentes det, og blir lagt til i program-minnet. Forespørselen blir så kjørt for å sjekke program-minne på nytt. Om det ikke finnes i cache blir det lastet ned fra nettet. Deretter blir det lagt til i både cache og program-minne og forespørsel blir kjørt på nytt.

Figur 6.2 viser hvordan det ønskes å implementere Kartverkets API inn i koden. Endringene er merket i rødt. Når data skal lastes ned blir Airmap eller Kartverket brukt i henhold til valgt innstilling. Data blir lagret på samme måte og form som det fra Airmap.



Figur 6.1: Når terrenghøyden til et koordinat blir etterspurt blir program-minne, eller runtime-minne, prioritert. Om det ikke finnes blir det henholdsvis hentet fra cache eller lastet ned.



Figur 6.2: Med Kartverket implementert skal datafly gå på samme måte som 6.1, men om data lastes ned skal det kunne velges hvilken distributør som brukes

QGC lager en ”provider”-klasse for alle kartdistributører som brukes. Disse klassene brukes til å skreddersy hvordan URLene ser ut når data skal lastes ned. De inneholder også andre funksjoner som kreves for å bruke den gitte kartdistibutøren. For å ha kontroll på alle objektene lagres de i en liste med indeks lik navnet på distributøren, se kodeliste 6.1. Denne kan brukes når man velger distributør. Da er det enkelt å hente opp riktig objekt basert på navn.

```
76 \\QGCMapUrlEngine.cpp
77     _providersTable["Airmap Elavation"] = new AirmapElevationProvider(this);
78     _providersTable["Statkart Elavation"] = new StatkartElevationProvider(this);
```

Kodeliste 6.1: Providertable med kartobjekter

Her kan koden for *AirmapElevationProvider* kopieres og endres for å passe til hva Kartverkets API trenger. I dette tilfellet trengs det kun å endre påfunksjonene som returnerer URLen. Et problem her er at Airmaps API returnerer en "Tile" direkte, som også er måten data blir lagret på, se figur 6.3. Kartverkets antallsbegrensning av koordinater gjør at det må sendes flere forespørsler for å laste inn alle høydene som trengs. URL-funksjonen til Kartverkets objekt kan bare returne en URL. Derfor returneres kun punktene som rammer inn området til en tile. Videre kode må gjøre disse om til de URLene som trenges. URL-funksjonen er vist i kodeliste 6.2.


```
{"status":"success","data":{"bounds":{"sw":[49.97,14.12],"ne":[49.98,14.13]},"stats":{"max":400,"min":241,"avg":348.912280701754 4},"carpet":
[[388,387,386,384,382,378,375,372,369,368,366,364,358,352,348,342,334,326,318,310,303,296,292,290,289,289,288,286,282,280,272,264,253,244,241,242,245],
[392,391,391,388,386,383,380,377,374,372,370,366,361,355,350,345,337,326,318,309,300,294,288,283,281,280,278,276,274,269,264,255,248,245,246,248,253],
[393,392,391,389,387,386,384,381,378,376,373,367,361,356,352,345,335,326,320,312,303,295,290,286,283,279,276,271,266,260,256,250,245,245,246,252,261],
[396,393,390,387,384,383,382,381,378,375,371,365,359,355,349,342,335,328,325,320,313,305,300,295,291,286,280,275,268,262,255,249,244,244,249,256,264],
[395,391,385,381,378,377,377,376,374,370,366,361,358,353,347,341,337,335,332,328,324,317,311,307,300,294,289,284,277,267,257,250,247,245,247,256,264],
[388,382,377,372,372,372,373,373,370,366,364,362,358,353,348,344,342,341,340,338,334,327,322,316,310,304,297,292,284,274,261,251,246,246,246,254,263],
[383,376,371,369,369,371,371,371,369,364,361,360,357,353,350,348,346,346,346,345,342,336,331,326,320,313,303,295,287,275,261,253,246,244,249,256,265],
[381,376,372,369,367,368,370,370,367,362,359,358,356,354,352,351,351,352,354,354,351,346,341,335,328,319,309,299,286,274,261,252,248,248,251,259,268],
[381,378,376,372,370,368,367,365,363,361,360,360,358,356,354,354,357,359,361,360,359,354,349,342,334,325,313,304,291,278,263,255,248,248,255,260,266],
[376,376,376,375,373,370,367,364,361,361,361,361,360,360,359,361,363,366,366,365,363,359,353,347,337,326,314,302,292,280,271,260,252,249,249,254,258],
[372,372,373,374,374,372,369,366,364,364,364,364,364,364,365,368,371,370,370,368,366,362,356,350,341,326,311,298,288,280,272,265,258,251,249,250,251],
[378,376,376,376,376,375,374,372,370,369,370,370,370,370,373,375,375,376,375,373,370,364,356,347,338,327,312,301,292,286,279,273,266,259,255,252,249],
[381,380,380,380,380,379,378,376,375,375,376,376,376,376,379,380,380,379,376,374,371,365,354,343,333,325,318,308,302,296,292,286,277,269,263,258,253],
[382,382,382,383,382,381,381,380,379,379,379,378,378,379,381,382,382,380,377,375,371,365,357,344,333,326,321,316,310,307,305,301,292,282,274,268,263],
[382,383,384,385,385,385,384,384,384,384,382,380,380,381,381,382,383,382,381,378,373,366,359,348,337,331,327,323,319,315,314,309,303,293,285,280,276],
[381,382,384,385,387,389,389,389,389,388,386,384,384,384,384,385,386,386,386,382,376,369,361,351,343,338,336,332,328,324,319,315,308,300,295,292,289],
[376,378,382,385,387,389,391,392,392,391,391,390,389,389,390,389,389,390,389,387,382,374,366,357,351,347,345,343,339,334,328,321,316,309,307,305,303],
[372,377,382,385,388,389,391,393,394,396,395,395,394,395,395,395,395,394,393,391,388,383,376,369,362,358,355,353,351,346,341,335,328,323,320,318,316],
[373,376,381,385,387,389,390,392,395,398,398,397,396,395,396,397,398,398,398,397,397,393,389,380,375,370,367,363,361,358,354,349,342,334,331,327,322],
[374,375,378,381,383,384,387,389,392,395,398,398,396,395,396,397,399,400,400,400,399,399,395,390,384,381,377,374,370,367,362,356,350,341,334,329,323],
[374,374,374,376,380,383,384,386,388,391,394,395,395,395,395,397,399,400,400,399,399,399,397,393,388,385,383,379,375,371,366,358,351,345,337,329,320],
[374,374,374,376,377,380,382,383,385,388,390,393,393,393,393,393,396,398,398,396,395,395,395,393,390,387,384,379,374,370,363,357,352,346,337,327,319],
[377,376,376,376,376,376,378,380,382,385,387,387,386,385,384,386,388,392,393,391,388,387,388,387,387,386,383,379,373,368,364,360,354,347,337,329,321],
[379,378,377,376,375,374,375,377,378,379,380,380,378,379,379,380,383,386,387,384,383,382,381,381,382,383,383,379,373,369,366,360,353,344,338,332,324],
[381,380,379,379,377,376,374,374,373,373,373,373,375,375,376,377,380,382,381,380,379,378,376,376,376,377,375,372,368,365,361,353,345,339,336,331,323],
[383,382,381,380,379,376,374,372,371,370,368,368,369,370,372,374,375,376,376,375,374,373,371,370,369,367,365,363,361,359,353,345,336,329,323,319,316],
[380,381,380,379,377,374,371,370,369,368,367,367,367,367,366,367,367,368,369,369,368,368,366,364,360,358,356,356,355,351,344,337,329,321,317,312,309],
[372,373,373,372,370,368,367,367,368,368,367,367,366,366,365,365,363,363,363,363,364,364,362,360,357,354,352,350,348,343,340,334,330,324,320,316,312],
[362,364,365,364,363,363,362,362,363,364,363,362,362,363,363,362,361,360,360,362,363,363,362,359,356,353,351,348,347,344,341,338,335,332,328,326,323],
[353,355,358,359,358,357,357,358,358,357,355,354,355,357,359,360,360,360,362,364,366,365,363,360,357,355,353,352,351,350,347,343,341,342,341,338,335],
[345,348,349,349,350,348,349,350,350,350,350,350,352,355,358,360,362,363,365,366,368,367,365,363,359,358,357,357,357,356,354,351,350,349,349,348,345],
[333,336,338,337,335,335,336,340,341,343,345,347,350,354,357,360,362,363,363,364,365,365,363,362,361,360,360,359,358,359,359,358,356,354,353,352,350],
[326,329,329,328,326,326,328,332,335,338,341,344,346,348,350,354,358,358,357,356,357,357,357,357,358,359,360,358,358,360,361,361,360,358,356,355,353],
[325,327,327,326,326,326,327,330,332,335,338,340,340,341,343,347,350,351,350,348,347,348,349,350,353,355,356,354,353,354,357,359,359,359,359,356,354],
[320,321,322,324,324,324,326,327,328,332,334,334,335,334,336,337,342,344,342,339,338,339,340,341,344,346,347,345,344,348,352,353,354,354,353,352,350],
[314,314,314,316,318,319,320,323,326,328,329,329,328,329,329,330,332,334,335,335,334,333,334,334,335,336,336,336,337,339,342,344,345,347,346,343,343],
[310,311,312,313,315,317,319,320,322,323,325,326,327,328,328,329,331,332,332,331,332,332,332,331,329,329,331,332,333,336,338,337,338,338,339,338,338]]}}
```


Figur 6.3: JSON-fil som Airmap returnerer i form av en Tile

```
66    QString StatkartElevationProvider::_getURL(const int x, const int y, const int zoom,
       QNetworkAccessManager* networkManager) {
67    Q_UNUSED(networkManager)
68    Q_UNUSED(zoom)
69    return QString("%1,%2,%3,%4")
70        .arg(static_cast<double>(y) * TerrainTile::tileSizeDegrees - 90.0)
71        .arg(static_cast<double>(x) * TerrainTile::tileSizeDegrees - 180.0)
72        .arg(static_cast<double>(y + 1) * TerrainTile::tileSizeDegrees - 90.0)
73        .arg(static_cast<double>(x + 1) * TerrainTile::tileSizeDegrees - 180.0);
74    }
```

Kodeliste 6.2: URL-funksjon for Kartverket

Nedlastingen av høydedata fra Kartverket skjer gjennom en egen objekter. Det er valgt å lage en egen klasse for å skille ny kode fra original kode. Slik er det er lettere å gå tilbake til original kode om det er ønskelig. Første som skjer når et objekt av Kartverkets klasse lages er at den sjekker om data allerede finnes i databasen, om ikke laster den ned. Dette er samme måte den originale koden gjør det på. Som vist i figur 6.1 blir data hentet fra database lagret i program-minne. Dersom det er manglende data, blir det lastet ned umiddelbart.

Som nevnt tidligere trengs det bare å sende en forespørsel til Airmap sitt API for nødvendig data. Mottatt data kommer i ett svar. I koden omtales disse som request og reply. En tile har $37 \cdot 37 = 1369$ høydepunkter. Kartverkets API kan maks returnere 50 høydepunkter i et svar om gangen. Det vil si at programmet må sende $1369/50 = 27.38 \approx 28$ forespørseler for å bygge en tile. Kodeliste 6.3 viser hvordan programmet først genererer en liste med URLene som trengs for å bygge en tile. Så itererer den seg gjennom listen og sender alle forespørsler. Svarene fra APIet kobles til funksjoner for å håndtere mottatt data eller håndtere feil. Dette gjøres ved å koble signalet fra svaret til en slot for håndtering med funksjonen connect. Når et svar har kommet tilbake sendes signalet til den tilkoblete sloten *statkartElevation::networkReplyFinished()*. Siden der er 28 forespørsler må det være en funksjon for å vente til det siste svaret har kommet. Kodeliste 6.4 viser hvordan dette foregår. Når siste svar er mottat går koden videre til behandling av data og lagrer det i database. Når alt er ferdig, sender startkartElevation-objektet et signal om at prosessen er gjennomført slik at resten av koden vet at høydedata ligger tilgjengelig i program-minne.

```
86      void statkartElevation::startDownload(){
87
88      //Gets the needed urls to make tile and puts them in _urls
89      getTileURLs();
90
91      //Send all requests needed to build tile
92      _index = 0;
93      for(QUrl url: _urls){
94          _request.setUrl(url);
95          QNetworkReply* reply = _networkManager->get(_request);
96          _replies.append(reply);
97          connect(_replies.at(_index), &QNetworkReply::finished, this, &statkartElevation::
        networkReplyFinished);
98          connect(_replies.at(_index), SIGNAL(error(QNetworkReply::NetworkError)), this,
        SLOT(networkReplyError(QNetworkReply::NetworkError)));
99
100         _index++;
101     }
102 }
```

Kodeliste 6.3: Sende forespørsler til Kartverkets API

```
106     void statkartElevation::networkReplyFinished(){
107
108         //If not all replies are done return, and wait for next call
109          for(QNetworkReply* reply: _replies){
110             if(!reply->isFinished()){
111                 return;
112             }
113         }
114     ....
```

Kodeliste 6.4: Sjekke om Kartverket API har svart

Koden for Kartverket må sys sammen med den eksisterende koden. I den eksisterende koden hentes høyder gjennom funksjonen vist i kodeliste 6.5. Fordi det er lagt vekt på at den nye API-funksjonen skal oppføre seg likt som den originale er dette ganske simpelt. Kartverket kan implementers på samme måte som Airmap. Hvilken distributør som brukes velges gjennom en if-funksjon som sjekker hvilken distributør som er valgt. Distributør blir valgt gjennom en egen valgmeny i brukergrensesnittet.

```
514 bool TerrainTileManager::getAltitudesForCoordinates(const QList<QGeoCoordinate>&
        coordinates, QList<double>& altitudes, bool& error)
515 {
516     error = false;
517 
518     QString elevationProvider = qgcApp()->toolbox()->settingsManager()->flightMapSettings
        ()->elevationProvider()->cookedValueString();
519 
520     //Chooses which elevationprovider to use
521     QString tileHash;
522     for (const QGeoCoordinate& coordinate: coordinates) {
523         if(elevationProvider == "Airmap Elevation"){
524             tileHash = _getTileHash(coordinate);
525         } else if (elevationProvider == "Statkart Elevation"){
526             tileHash = _getTileHashStatkart(coordinate);
527         }
528     ....
```

Kodeliste 6.5: Funksjonen som henter høyder

6.3.2 Valgmeny for distributør av terrenghøyde

Brukeren trenger en valgmeny hvor det er mulig å velge mellom distributørene. For å kunne implementere denne funksjonaliteten må en vite hvor elementer i brukergrensesnittet lages. I tillegg må variabler bli opprettet slik at brukergrensesnittet kan kommunisere med resten av programmet.

I applikasjonens innstillinger kan brukeren, gjennom innstillingen *Map Provider*, velge hvilken aktør som distribuerer kartet i brukergrensesnittet. Dette valget er underlagt *Application Settings/General(Miscellaneous)* og er en av flere innstillinger implementert som en valgmeny. Dermed vil den nye valgmenyen naturlig passe inn på samme sted, under *Miscellaneous*, men som *Elevation Provider*.

Valgmenyens plassering er nå fastsatt, men det er bestemt visuelt i programvaren. Neste steg er å implementere denne i koden, men da må programfilen som inneholder elementer i brukergrensesnittet lokaliseres. Qt Creator gjør det enkelt å lokalisere filer og programelementer som objekter, variabler, og funksjoner. Som nevnt i avsnittet over, ligger innstillingene under *General*. I tillegg er det kjent fra kapittel 5 at brukergrensesnittets grafiske elementer opprettes i QML filer. For å lokalisere korrekt fil benyttes IDEens søkefeltet til å finne *GeneralSettings.qml*. Ettersom *Map Provider* innstillingen har det ønskede grafiske utseende, kopieres koden og tilpasses den nye valgmenyen.

Valgmenyen i QML trenger en liste med hvilke aktører brukeren kan velge mellom. Listen er laget som en funksjon i backendfilen QGCMapEngineManager.cc og aksesseres direkte fra QML-filens liste model. Se linje 535 i Kodeliste 6.8 Denne metoden er samme måte som eksisterende valgmenyer i programmet. Listen består kun av to valgmuligheter, *Airmap elevation* og *Statkart elevation*. Når brukeren velger aktør, sjekkes indeksen på den valgte aktøren og den brukes for å lagre strengen i en fact som kan aksesseres fra backendfilene *FlightMapSettings.cc/h*. I kodeliste 6.6 opprettes en 'elevationProvider', en fact som gjør at både brukergrensesnitt og backend til en hver tid vet hvilken terrengdistributør som velges.

```
    \\FlightMapSettings.cc
    DECLARE_SETTINGSFACT(FlightMapSettings, elevationProvider)
    \\FlightMapSettings.h
    DEFINE_SETTINGFACT(elevationProvider)
```

Kodeliste 6.6: Opprett terrengdistibutør fact

Fact må spesifiseres som JSON-objekt dersom en ønsker å aksessere dem i- og manipulere dem fra brukergrensesnittet. Objektet innholder informasjon om facten. Det forteller hvilken elementet som tilhører fact, hvilken datatype elemenet er, defaultverdier, og mer. Kodeliste 6.7 viser hvordan elevationProvider spesifiseres.

```
17 \\FlightMap.SettingsGroup.json
18 {
19     "name":             "elevationProvider",
20     "shortDesc": "Statkart or Airmap",
21     "type":             "string",
22     "default":      "Airmap Elevation"
23 },
```

Kodeliste 6.7: Spesifiser terrengdistributør fact

I etterkant av brukervalget passer funksjonen, Component.onCompleted, på å holde indeksen oppdatert da den brukes som argument når distributør skal velges på ny.

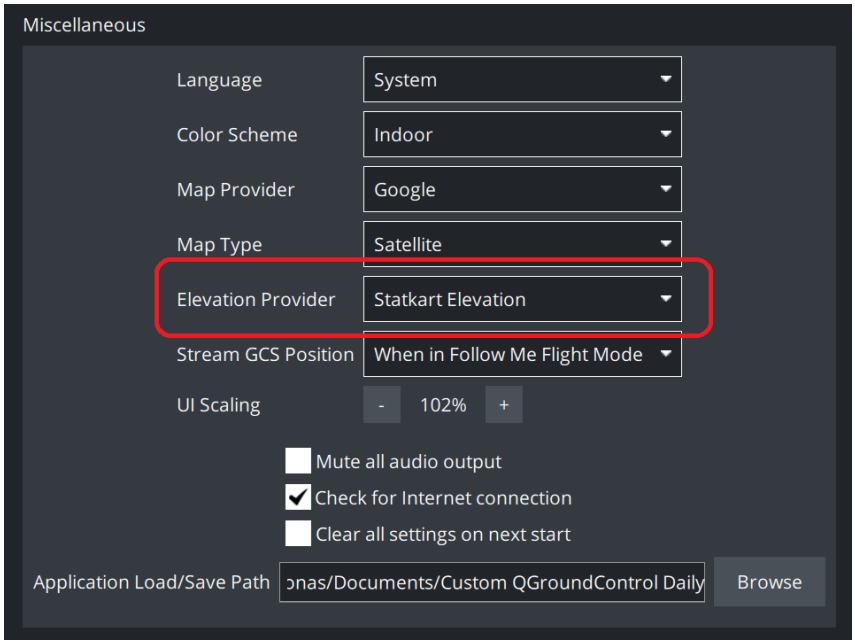
```
\\GeneralSettings.qml
QGCLabel{
    text:       qsTr("Elevation Provider")
    width:      _labelWidth
 }
 QGCComboBox
 {
    id:                     elevationProviderCombo
    model:                  QGroundControl.mapEngineManager.elevationProviderList
    Layout.preferredWidth:  _comboFieldWidth
    onActivated:
    {
        if(textAt(index) !== _elevationProvider){
            QGroundControl.elevationProviderChanged()
        }
        _elevationProvider = textAt(index)       QGroundControl.settingsManager.
     flightMapSettings.elevationProvider.value = textAt(index)
    }
    Component.onCompleted:
    {
        var index = elevationProviderCombo.find(_elevationProvider)
        if(index < 0) index = 0
        elevationProviderCombo.currentIndex = index
    }
}
```

Kodeliste 6.8: Menyvalg av terrenghøyde distributør

6.4 Resultater

6.4.1 Valgmeny for distributør av terrenghøyde

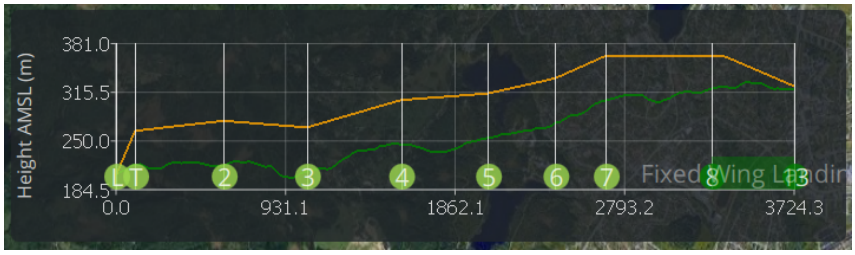
Valgmenyen er lagt inn sammen med andre kartvalg i programmets innstillinger. Listen med valg blir automatisk generert av de registrerte høydedistributørene i koden. Ved bytte av distributør sendes et signal videre inn i koden som trigger en oppdatering av UI-elementer. Valgmenyen er vist i figur 6.4.



Figur 6.4: Valgmeny for høydedistributør ligger i eksisterende side for innstillinger i "Application settings".

6.4.2 Høydedata

Terrenghøyde hentes på samme måte som når Airmap sin API brukes. Når ”Statkart Elevation” er valgt inn, lastes data automatisk ned der ruten er lagt opp. Nedlasting fra Kartverket tar noe lengre tid ettersom en forespørsel fra Airmap tilsvarer 28 fra Kartverket. Som nevnt tidligere, sendes et signal når distributør byttes. På dette signalet oppdateres terrengprofil og andre elementer, se figur 6.5. Funksjoner som bruker terrenghøyder fungerer som normalt med Kartverket. Dette gjelder for eksempel ”calc above terrain”. Den bruker høydedata til å automatisk sette høyden til waypoints en gitt distanse over terrenget. Når en tile med høydedata er lastet ned blir den lagret i program-minne og databasen. Slik tar det kort tid å hente høyde neste gang koordinaten blir etterspurt.



Figur 6.5: Høydedata for denne terrengprofilen er hentet fra Kartverket.

6.4.3 Endringslogg

Tabell 6.1 viser alle kodeendringer som er gjort under utviklingen av modul 1.

Filsti	Filnavn	Tillegg
src\Terrain\	TerrainQuery.h	Deklarere medlemer: TerrainTileManager::_terrainDoneStatkart, TerrainTileManager::_getTileHashStatkart
src\Terrain\	TerrainQuery.cc	Endret medlem: TerrainTileManager::getAltitudesForCoordinates Definere medlemer: TerrainTileManager::_terrainDoneStatkart, TerrainTileManager:: _getTileHashStatkart
src\QtLocationPlugin\	ElevationMapProvider.h	Deklarere klasse: StatkartElevationProvider
src\QtLocationPlugin\	ElevationMapProvider.cpp	Definere klasse: StatkartElevationProvider
src\QtLocationPlugin\	QGCMapUrlRngine.cpp	Endring: Legge til StatkartElevationProvider i objektliste
src\StatkartAPI\	statkartelevation.h	Deklarere klasse: statkartElevation
src\StatkartAPI\	statkartelevation.cpp	Definere klasse: statkartElevation
src\	QGCApplication.h	Deklarere medlem: signalElevationProviderChanged Deklarere signal: elevationProviderChanged
src\	QGCApplication.cc	Definere medlem: signalElevationProviderChanged
src\QtLocationPlugin\QMLControl\	QGCMapEngineManager.h	Deklarere medlem: elevationProviderList
src\QtLocationPlugin\QMLControl\	QGCMapEngineManager.cc	Definere medlem: elevationProviderList
src\Settings\	FlightMapSettings.h	Definerer settingsfact: elevationProvider
src\Settings\	FlightMapSettings.cc	Deklarere settingsfact: elevationProvider
src\Settings\	FlightMap.SettingsGroup.json	Definere FactMetaData: elevationProvider
src\ui\preferences\	GeneralSettings.qml	Legge til valgmeny: elevationProviderCombo

Tabell 6.1: Endringslogg for modul 1

6.5 Analyse og diskusjon

6.5.1 Datastruktur

QGC lagrer høydedata i form av en tile. Dette fungerer bra for Airmap sitt API ettersom at den kan returnere en matrise med høyder mellom et sørvestlig og nordøstlig punkt. Da kommer all data etter kun en forespørsel. Når Kartverket sitt API må sende 28 separate forespørsler for å oppnå det samme vil dette bruke mer tid. Dette skjer fordi hver

forespørsel bruker litt ekstra tid. Et alternativ er å hente kun de koordinathøydene som blir etterspurt. Dette var en løsning som ble undersøkt i en tidlig fase av utviklingen. Da var innhentingen mye kjappere. Grunnen til et denne løsningen ble skrinlagt kommer av hvordan høydedata lagres. Mye av koden til QGC er lagt opp til at det er Airmap som brukes. Den eksisterende løsningen for å lagre data går ut i fra at det er en tile. Derfor må data fra Kartverket settes opp på samme måte, og da må alle høyder for hele tile-en lastes ned. Fordelen er at når data først er lastet ned blir den lagret gjennom eksisterende funksjoner. Høyder som hentes fra databasen gjøres like kjapt som Airmap. Ofte vil også flere koordinathøyder bli brukt fra samme tile samtidig.

6.5.2 Distributørmeny

I praksis kan også listen lages direkte i QML-filen ved en annen deklarasjon for oppsettsmodellen. Fordelen med denne varianten er at den er mer oversiktlig. Listeelementene er synlig i QML-filen i stedet for å være ”gjemt” i backend. Dermed er det enklere å legge til eller fjerne listelementer. Ulempen er at det bryter med kodeskikken i kildekoden. Utviklerene av QGC ønsker bruke eksisterende datatyper direkte fra objekter i backend, som kollektivt bidrar til redusere behovet for prosessorkraft fordi det tar opp mindre plass i programminnet.

6.5.3 Implementering i eksisterende kode

6.5.4 Plugin-arkitektur

Under modul 1, var det opprinnelig en arbeidspakke for å bygge rammeverket for et plugin. QGC har allerede et plugin-eksempel liggende i kildekoden til bruk dersom det ønskes å gjøre egne justeringer til programmet. Plugin-arkitektur er gunstig for små tilleggskomponenter som funksjonalitetene utviklet under bacheloroppgaven. Årsaken til dette er at det ikke er nødvendig å laste ned mer enn pluginfilene for å kunne benytte seg av tilleggsfunksjonalitetene. I tillegg, dersom oppdragsgiver senere ønsker å legge til egen funksjonalitet, vil plugins kunne eksistere i samme build. Rammeverket til et slikt system krever muligheten til å legge til endringer hos alleredeeksisterende filer, slik at egne funksjoner og filer kan få tilgang til kildekoden. Til dette, ble det dannet en egen *custom\src*-mappe med C++-filer. Kildekode ble kopiert til custom-mappen og

endret på, slik at kildekoden kunne erstattes med justert kode ved kompilering. Denne tilnærmingen virket for ordinære klasser, men ikke for klasser som benyttet seg av nettverkstjenester. Arbeidspakken ble dermed sløyfet, da mye av modul 1 innebærer å designe et eget system for håndtering av nettverksforespørsler.

Likevel, ble det utviklet et system for erstatning av QML-filer, da det tilsynelatende allerede lå til rette for dette i QGCs plugin-eksempel. I modul 2 og 3, er det lagt til UI-elementer til filen *PlanView.qml* som ble implementert via en egen fil ved navn *CustomPlanView.qml*. Likevel, på grunn av problemer med erstatting av C++-filer, ble rammeverket for slik plugin-arkitektur ferdigstilt såpass sent at andre endringer også hadde blitt gjort i QML. Grunnet manglende helhetlig suksess med plugin-arkitekturen, ble det avgjort å ikke gjøre videre arbeid med det. Som et resultat, er enkelte deler av koden implementert rett inn i kildekoden, mens andre deler enda ligger i egne erstatningsfiler i *custom*-mappen.

En ideell løsning på dette hadde vært å funnet ut av plugin-situasjonen tidligere slik at all implementering kunne skjedd på konsistent vis gjennom prosjektet. Da kunne all tilleggskode vært samlet på ett og samme sted. Som en løsning på dette problemet, er det inkludert endringslister for alle kodetillegg i hver modul.

6.6 Kapittel konklusjon

Selv om prinsippet med tiles gjør at mye data lastes ned unødvendig er dette nok beste alternativ. Det tar lengre tid å laste ned første gang, men etter da så er data lagret på disk. Distibutørmenyen ligger på et naturlig sted sammen med andre kartinnstillinger og trigger en oppdatering av brukergrensesnittet og ”waypoints”. Sammen gir disse komponentene brukeren en intuitiv og lett måte å administrere hvilken høydemodell som brukes.

Skulle det blitt gjort på nytt ville nok det blitt prioritert å optimalisere den originale koden fremfor å gjøre ny og gammel kode så uavhengig som mulig. Ettersom det måtte gjøres mange endringer i kildekode uansett. Likevel er dette en tilstrekkelig måte å implementere løsningen på med tanke på størrelsen av endringene.

Kapittel 7

MODUL 2: Autokorrigering av kollisjonskurs

7.1 Introduksjon

Modul 2 har som hovedoppgave å automatisk gjennomføre oppgaver som ellers ville vært repetitivt og tidkrevende arbeid for operatør. Dersom en rute blir lang nok, vil selv de enkleste rutinesjekker eller endringer bli repetert mange nok ganger til at det heller lønner seg med automatikk. Selv om programmet legger til rette for alle slags flyvninger, er det enkelte aspekter ved ruten, operatøren uansett trenger å ta stilling til. Derfor er det implementert tre hovedfunksjonaliteter ved modul 2:

- **Kollisjonsunngåelse**:
  Plassering av waypoints slik at ruten legger seg over terrenget
- **Baneutjevning**:
  Utjevning av ruten for å tilpasse seg dronens stigning- og nedstingningegenskaper
- **Minimumshøyde**:
  Sikring av minimumshøyde over bakken

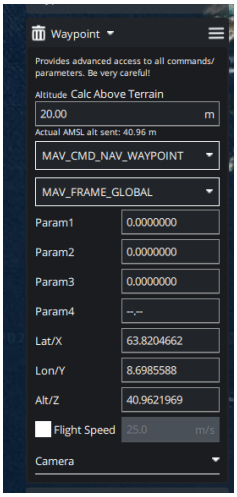
Alle disse skal implementeres på et presist, effektivt og brukervennlig vis.

7.2 Teoretisk rammeverk

7.2.1 VisualItem

Navnene visualitem og missionItem brukes om hverandre i koden. MissionItems er QGCs måte å lagre all nødvendig informasjon om waypointene til en rute under planlegging. Listen *visualItems* hos *missionController* består av disse missionItemene, som er hvorfor de ofte kalles VisualItems. Figur 7.1 viser et eksempel på en rute. Hvert waypoint er et eget visualItem med informasjon om koordinater, punkthøyden, hvilket "Altitude mode" den er i, flight speed og lignende. Figur 7.2 viser hvordan dette presenteres i QGC.

Figur 7.1: Figuren viser en ruten bestemt av 7 visualItems. 5 av dem er av typen generic.



Figur 7.2: Visualitem parametre

7.2.2 FlightPathSegment

Et segment er flystrekningen mellom to waypoints. Disse blir lagret som objekter av klassen FlightPathSegment, og inneholder informasjon om området mellom to waypoints. Det innebærer avstand og stigningsrate, segmenttype, kollisjoninfo og alle terrenghøyder langs strekningen. Tabell 7.1 viser oversikten over de viktigste datapunktene lagret i segmenter.

Navn	**Beskrivelse**	**Datatype**
coordinate1	Koordinat til startpunkt	QGeoCoordinate
coordinate2	Koordinat til sluttpunkt	QGeoCoordinate
coord1AMSLAlt	Høyden til startpunkt (moh)	double
coord1AMSLAlt	Høyden til sluttpunkt (moh)	double
amslTerrainHeights	Liste av terrenghøyder	QVariantList
totalDistance	Kollisjonssjekk	bool
segmentType	Takeoff / generic / landing	SegmentType

Tabell 7.1: De viktigste datatypene flightPathSegmenter inneholder.

7.2.3 Matriser og vektorer

Dersom et ligningssett er lineært, slik som i eksempel 7.2, kan ligningssettet formuleres på matrise-vektor-form. Slik kan man løse store lineære ligningssett hurtig og oversiktlig. Ligningssettet kan skrives på formen vist i eksempel 7.3 og 7.4.

$$a_{1,1} \cdot x_1 + a_{1,2} \cdot x_2 = b_1 \tag{7.1}$$
$$a_{2,1} \cdot x_1 + a_{2,2} \cdot x_2 = b_2 \tag{7.2}$$

$$Ax = b \tag{7.3}$$

$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{7.4}$$

Løsningen for x kan finnes ved å invertere A-Matrisen. Eksemplet under (7.6) viser denne prosessen for en matrise av dimensjon 2x2. Invertering av større matriser krever naturligvis mer arbeid. En matrise kan kun inverteres dersom A er kvadratisk og $det(A) \neq 0$.

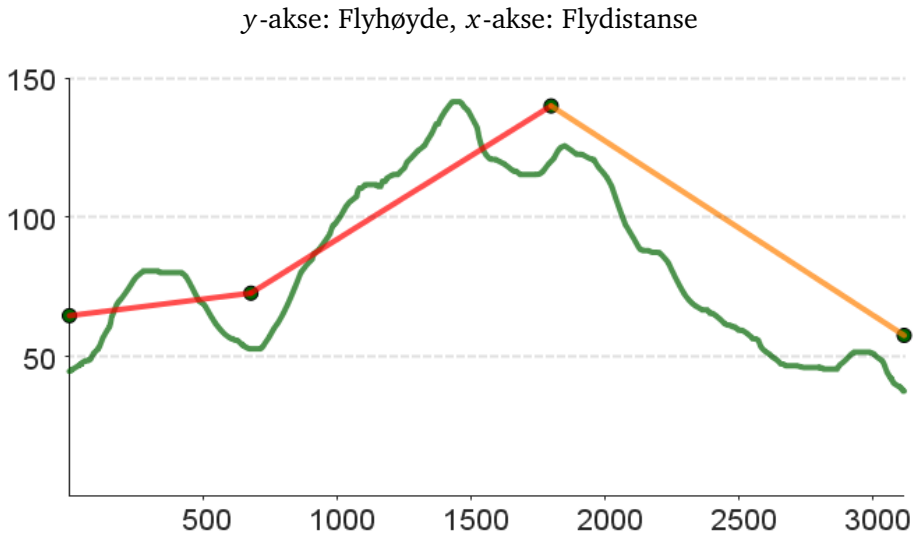
$$x = A^{-1}b \tag{7.5}$$

$$A^{-1} = \frac{adj(A)}{det(A)} = \frac{\begin{bmatrix} a_{2,2} & -a_{1,2} \\ -a_{2,1} & a_{1,1} \end{bmatrix}}{a_{1,1} \cdot a_{2,2} - a_{1,2} \cdot a_{2,1}} \tag{7.6}$$

7.3 Metode og utstyr

7.3.1 Tilpass ruten til terrenget

Det er utviklet funksjonalitet som tar inn en planlagt rute, og tilpasser den over terrenget. Funksjonen skal ikke endre koordinatene dronen flyr gjennom, kun gjøre justeringer hos waypoint-høyder.



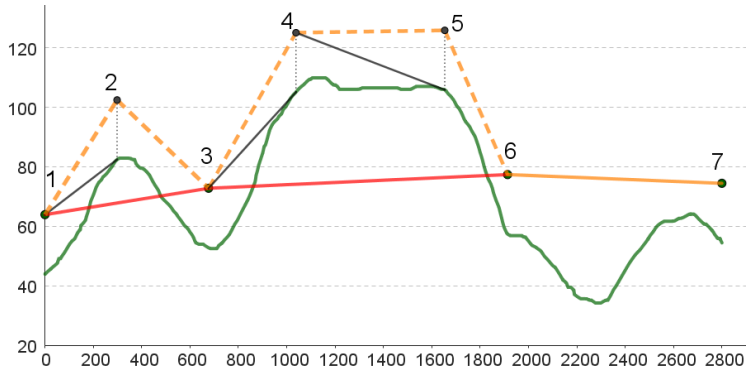
Figur 7.3: Selv om punktene automatisk justerer seg i henhold til terrenget (grønn), er plasseringen slik at segment 1 og 2 går gjennom bakken. Dette varsles med rød linje.

Utfordringen med oppgaven er å lage en algoritme som analyserer terrenget for å bedømme hvor det trengs å legges inn punkter underveis. Figur 7.3 viser et eksempel på en situasjon hvor dette trengs. Punktene er plassert for langt fra hverandre, og på hver sin side av hindringer. I praksis, vil dette forårsake at dronen improviserer en rute for å unngå kollisjon, noe som gjør flyvningen uforutsigbar og potensielt farlig for andre luftferdende i området.

Valgt tilnærming er en iterativ prosess som gradvis bygger en flybar rute helt frem til neste waypoint. Hvert middelpunkt settes et sted hvor det er garantert å kunne fly til. Dette oppnår den ved å plassere neste waypoint på stedet i terrenget med størst stigning fra forrige waypoints perspektiv. Dette kan tenkes på som å sette neste punkt i enden av

forrige punkts horisont (se figur 7.4). Dette er en rask måte som er garantert å finne en løsning hvor et hvert strekke. I tillegg er prosesseringstiden tilnærmet proporsjonal med flyavstanden, hvilket er gunstig, da kjøretiden til algoritmer typisk øker eksponentielt med lengden på data de arbeider gjennom.

Figur 7.4 viser hvordan en rute på kollisjonskurs får nye punkter innlagt for å komme seg over terrenget. Hvor mange punkter som settes, avhenger av terrenget og lengden mellom punktene. Som sikkerhetstiltak, er det satt en maksgrense på 100 punkter innad hvert segment. Sperren er der for å hindre at programmet kræsjer ved lengre strekninger, da det å legge til mer enn 100 punkter på én gang krever svært mye prosesseringskraft. Dersom et segment er veldig langt, med svært ujevnt terreng og 100 punkter ikke er nok, kan man trykke på knappen en gang til.

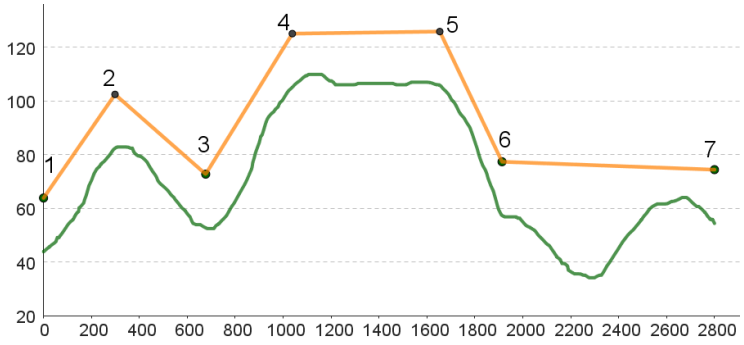


Figur 7.4: Algoritmen setter nye punkter hvor stigningen fra forrige punkt er brattest. Det kan tenkes på at basen til hvert nye punkt settes lengst mulig unna, men fremdeles innen synsrekkevidden til forrige punkt.

For å tildele koordinater til hvert nye waypoint brukes lineær interpolering. Eksempelvis, gis puntk 4 og 5 en verdi fra 0 til 1 som korresponderer med hvor langt langs segment $3 \rightarrow 6$ punktet skal plasseres. Videre gis nye punkters koordinater ved bruk av lineær-interpolering mellom endepunktkoordinatene med dette forholdstallet som parameter. Se formel 8.1 for matematisk begrunnelse. Nye punkter har dermed fått tilhørende koordinater, og med riktig styring på indekser, legges punkter inn med følgende funksjon:

```
1    Q_INVOKABLE VisualMissionItem* insertSimpleMissionItem(QGeoCoordinate coordinate, int
       visualItemIndex, bool makeCurrentItem = false);
```

Kodeliste 7.1: Funksjonen for å legge et nytt missionItem i visualitems.



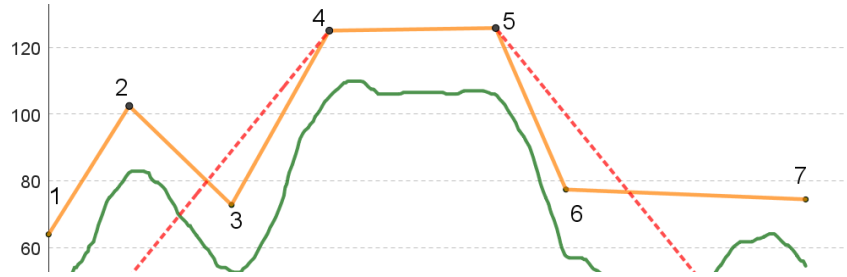
Figur 7.5: Når algoritmen er ferdig, ser ruten slik ut, og operatør blir informert om hvor mange punkter som ble lagt til underveis. I dette tilfellet ble det lagt til 3 nye punkter.

7.3.2 Jevn ut ruten for stigningsrater

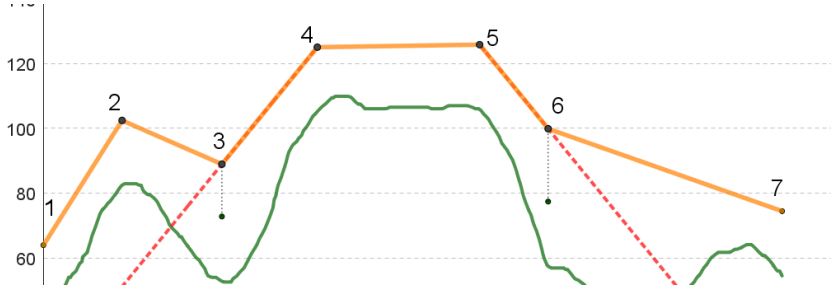
Videre er det utviklet en funksjon som sørger for at ruten er mulig å fly med dronens egne begrensninger for stignings- og nedstigningsrate. Etter at QGC har gjennomført kollisjonsunngåelse, kan segmenter få ubegrensede bratte stigninger. QGC informerer operatør om stigningen for hvert segment, men har ingen innebygde funksjoner for å rette opp i dette. Uten å glatte ut banen, er ikke dronen istand til å fly ruten, og den blir nødt til å improvisere underveis i flyvningen. Funksjonen skal oppfylle følgende kriterier:

- Glatt ut flyruten slik at hvert segment har stigning innenfor stigningsratene.
- Ikke endre ruten mer enn nødvendig. Maksimal terrengfølging er ønskelig.
- Justering i høyde skal kun heve punkter, ikke senke dem.
- Ikke gjør endringer i xy-planet, kun i høyde (z-retning).

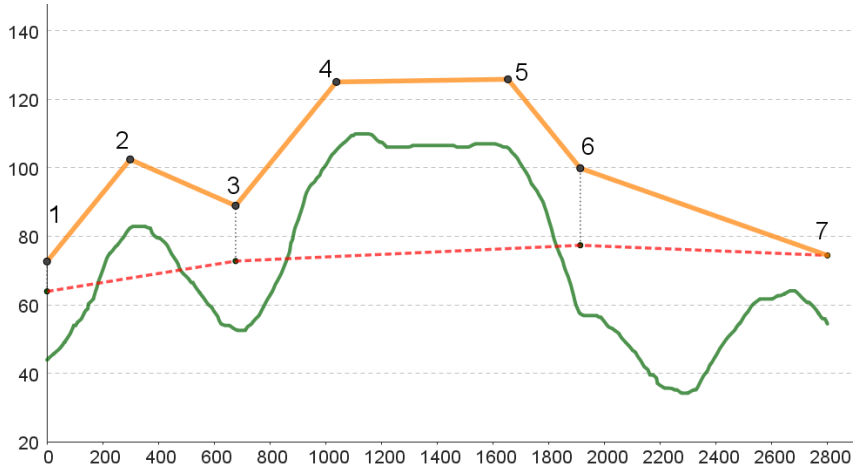
Løsningen på dette problemet er en funksjon som finner ut ny høydeplassering for hvert punkt individuelt. For eksempel, dersom et segment er for bratt, er det det laveste punktet som må løftes. Se figur 7.6.



Figur 7.6: Punkt 3 ligger lavere enn hva stigningsraten tillater, mens Punkt 6 ligger lavere enn hva nedstigningsraten tillater.

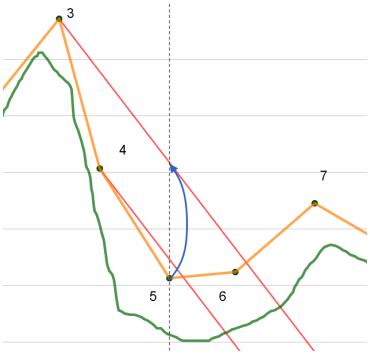


Figur 7.7: Punkt 3 og 6 har blitt løftet. Merk at segment 1-2 også er for bratt, hvilket må også tas hensyn til for å kunne gjennomføre flyvningen.

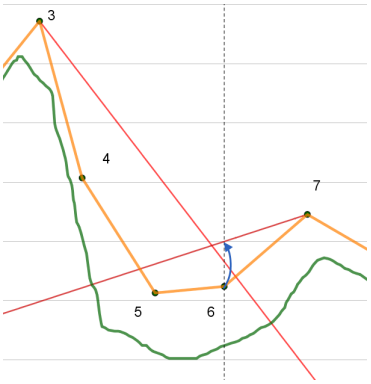


Figur 7.8: Slik blir den ferdige ruten, som nå kan bli fløyet med de bestemte begrensningene. Opprinnelig rute er vist i rød.

Nedre høydeverdier evalueres for hvert punkt i henhold til stigningsratene. Hvert punkt justerer seg så etter den høyeste. Prosessen gjennomføres for alle punkter, og tar hensyn til at hvert punkt skal ligge tilgjengelig for alle de andre. Dersom et punkt ligger for lavt, flyttes det akkurat innforbi rekkevidde.

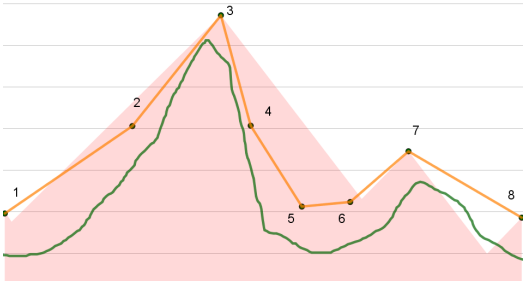


Figur 7.9: Punkt 5 må flyttes rett opp til over linjen fra punkt 3 for å gjøre ruten gjennomførbar.

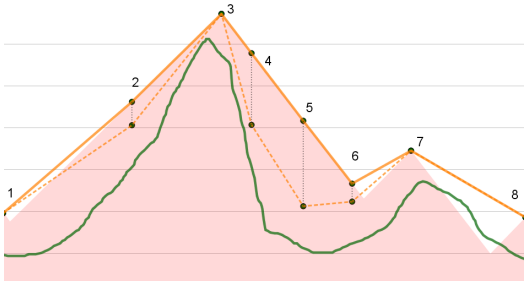


Figur 7.10: Punkt 6 må flyttes rett opp til over linjen fra punkt 7 for å gjøre ruten gjennomførbar.

Evaluert slik, vil det danne seg en sone hvor ingen punkter skal ligge. Dersom punktene ligger her, vil det oppstå segmenter med for bratte stigningsrater. Alle punkter i sonen ”flyter” opp til et godtatt høydenivå.



Figur 7.11: Alle punkter i rød sone må flyttes ut av den for å kunne gjennomføre flyvningen. Punkter flyttes rett opp.



Figur 7.12: Slik ser ruta ut etter at alle punkter er flyttet over grensene.

Algoritmen er implementert som et Klassemedlem hos PlanMasterController. Medlemmet krever tilgang til missioncontroller-objektet for å kunne justere høyden til visualItems. Missioncontroller-objektet er også innehaveren av simpleFlightPathSegments, en liste som inneholder mye data om selve strekningene mellom visualItems. Algoritmen er delt inn i fire steg.

1. Sett opp hvert punkt med tilbakelagt bakkeavstand X og høyde Y langs ruten fra startpunkt.
2. Evaluer data og finn høyeste nye høyde for hvert waypoint.
3. Iterer over listen med visualItems og oppdater verdiene dersom endringen i høyde er positiv.
4. Oppdater brukergrensesnittet.

Steg 1:
Lag lister for x- og y-verdiene til hvert waypoint slik at utregning kan foregå grafisk. X-verdi er bakkeavstand til hvert punkt, Y-verdi er høyde over bakken. Tradisjonelt sett blir høyde betegnet med Z i denne rapporten. Unntaket her kommer av at tilnærmingen er svært grafisk, som gjør det naturlig å heller benytte xy-planet enn XZ-planet.

Algorithm 1 Sett opp X-, og Y-verdier

1: Set $flightDistance$ as 0
2: Create empty lists called $xVals$ and $yVals$
3: Get the segments list from $\_missionController.simpleFlightPathSegments()$
4: **for** each $segment$ in $segmentsList$ **do**
5: Get the total distance of the segment and add it to $flightDistance$
6: Append $flightDistance$ to $xVals$
7: Append $segment$'s $coord2AMSLAlt()$ to $yVals$
8: **end for**

Steg 2:
Hvert punkt, i, sjekker alle andre punkter, j, for laveste tillatte relative høyde. Sjekker så at laveste tillatte høyde er høyere enn nåværende høyde for å bestemme om endringen skal skje. Listen *newDeltas* inneholder alle punkters nye høyder.

Algorithm 2 Finn nye høyder for hvert punkt

```
1: Set lowLim as 0
2: Create an empty list called newDeltas
3: for i from 0 to segmentsList's count - 1 do
4:     Set highestLimit as 0
5:     for j from 0 to segmentsList's count do
6:         if i = j then
7:             Continue
8:         end if
9:         if j < i then
10:            Set lowLim as (−descentRate ∗ (xVals[i] − xVals[j])) + yVals[j]
11:        else
12:            Set lowLim as (climbRate ∗ (xVals[i] − xVals[j])) + yVals[j]
13:        end if
14:        if lowLim > highestLimit then
15:            Set highestLimit as lowLim
16:        end if
17:    end for
18:    Get the ith segment from segmentsList and assign it to segment
19:    Get the i+1th mission item from _missionController.visualItems() and assign
   it to currentItem
20:    if segment's coord2AMSLAlt() < highestLimit then
21:        Append highestLimit − segment's coord2AMSLAlt() + currentItem's
   altitude() value to newDeltas
22:    else
23:        Append currentItem's altitude() value to newDeltas
24:    end if
25: end for
```

Steg 3:

Hvert punkts høyde oppdateres med ny høydeverdi fra steg 2.

Algorithm 3 Implementerer endringer

```
1: Print "Fix should be done now. Making changes:"
2: for i from 0 to segmentsList's count - 1 do
3:     Get the i+1th mission item from _missionController.visualItems() and assign
   it to currentItem
4:     Set currentItem's altitude() cooked value to the ith value in newDeltas
5: end for
6: Update the user interface with the new changes
```

Steg 4:

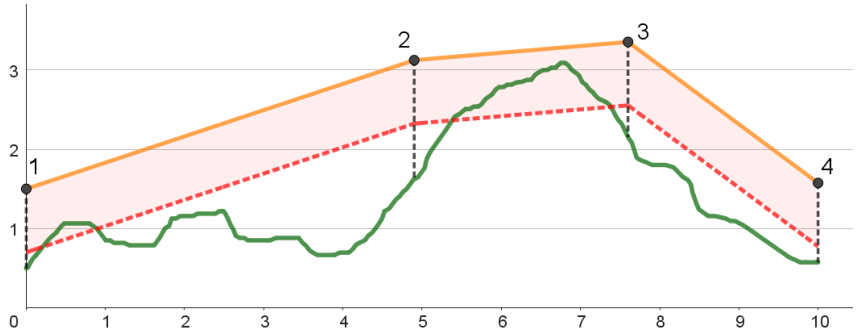
Etter ”fact”-endringer hos visualItems, sliter QGC med å oppdatere brukergrensesnittet. Et eksempel på dette er å endre ”fact”-et visualItem.altitude. Ideelt ville QGC oppdatert høydeprofilen, mission time og andre viktige verdier som i realitet kun blir satt til NaN. Det ble undersøkt mange måter å tvinge dette til å oppdatere seg, hovedsakelig ved bruk av signaler. Disse metodene ga ikke ønskede resultater, og det ble funnet en rask løsning i stedet. For å tvinge en oppdatering for et visualItem, kan man skru på ”show all values”, som tvinger visualitemet til å hente inn all data på ny. Dette funker som en refresh-knapp man kan skru på fra c++koden. Hvert visualItem har en *rawEdit*-fact som skrur av eller på denne modusen. Reserveløsningen går gjennom alle items og skrur på og av ”rawEdit” slik at brukergrensesnittet tvinges til å oppdatere seg. Dette kjøres da på slutten av alle større funksjoner som endrer visualItems fra c++siden.

Algorithm 4 Oppdatering av UI

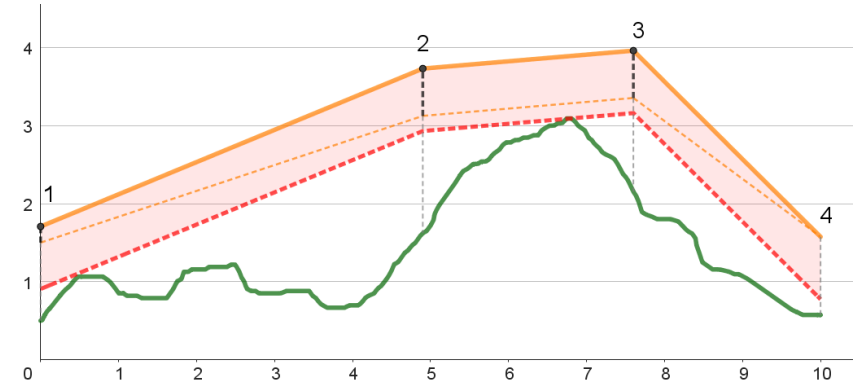
```
1:  function UpdateUi
2:      for i from 1 to _missionController.visualItems()'s count - 1 do
3:          Get the ith mission item from _missionController.visualItems() and assign
    it to currentItem
4:          set wasRawEdit to currentItem.rawEdit
5:          set currentItem.rawEdit to True
6:          if wasRawEdit is False then
7:              set currentItem.rawEdit to False
8:          end if
9:      end for
10: end function
```

7.3.3 Sikre minimumshøyde over bakken

Siste steg for å sikre ruten er å kunne sette en minimumsavstand til bakken som ruten alltid holder seg over. Måten høydedata integreres i modul 1, tar hverken hensyn til vegetasjon, bebyggelser eller annet som strekker seg over bakken. Uventede hindringer vil føre til improvisering hos dronen, og vil være uforutsigbart. I verste fall vil det kunne være skadelig for dronen, bygninger eller folk. Derfor vil en algoritme som sikrer minimumshøyde over bakken kunne bidra i vesentlig grad til flyvningens sikkerhet. Løsningen på dette problemet er å lage en knapp som løfter nødvendige segmenter høyt nok til å garantere en minimumsavstand til bakken.



Figur 7.13: For flyvningens sikkerhet skal terrenget alltid være lavere enn det røde varselområdet. Terrenget er enkelte steder høyere enn hva minimumsavstanden tillater.



Figur 7.14: Segment 1 og 2 er løftet for å opprettholde den nødvendige klaringen til bakken. Det er ikke lenger terreing i varselområdet. Den tidligere ruta vises i stripet oransje linje. Merk at ikke alle punkter økes med samme mengde.

Knappen som aktiverer funksjonen er programmert i QML, og aksesserer et klassemedlem i *PlanMasterController*-objektet. Algoritmen er inndelt i tre steg:

1. Sjekk minimumsavstand til bakken. Lagre laveste avstand.
2. Løft nødvendige segmenter.
3. Oppdater brukergrensesnittet.

Steg 1:
Medlemmet sjekker nærmeste bakkeavstand for hvert segment ved å iterere over segmentets liste med høydeverdier. Først beregnes en ligning for segmentets stigning. Deretter evalueres avstanden til bakken for hvert punkt langs segmentet. Gjennom prosessen tas det vare på den minste bakkeavstanden. Dersom dette er mindre enn tillatt, legges en endring inn i listen med endringer. Dersom korteste avstand er større enn minimumsavstanden, gjøres det ingen endringer for det segmentet, som oppnås ved å legge til 0 til listen av endringer.

Algorithm 5 Regn ut høydeendringer

Declare $segmentsList$ as $\_missionController.simpleFlightPathSegments()$
Declare $itemList$ as $\_missionController.visualItems()$
Create empty QList of doubles called $heightChanges$
for each $segment$ in $segmentsList$ **do**
    Get the slope using the equation $slope = (alt2 - alt1)/(segmentlength)$
    Get the y-intercept from $yIntercept = alt1$
    Get the minimal altitude value
    Set $minDist$ to a very large value
    for each index x in $segment->amslTerrainHeights()$ **do**
        Get the terrain height at that index
        Calculate the current distance from the ground using $currDist = slope * x + yIntercept - y$
        if $currDist < minDist$ **then**
            Update $minDist$ to be $currDist$
        end if
    end for
    if $minDist < minimalaltitude$ **then**
        add minimal altitude - minDist to heightChanges
    else
        add 0 to heightChanges
    end if
end for

Steg 2:
Implementer endringene ved å iterere over segmentene nok en gang. Hvert waypoint får sin tilhørende høydeendring.

Algorithm 6 Implementer høydeendringer hos missionItems

1: Create a variable called $biggestChange$
2: **for** every visualItem exept the first and last **do**
3: set biggestChange equal to the biggest height change of both segments the visualItem is connected to
4: Calculate the new altitude for the current item by adding $biggestChange$ to the current altitude
5: Set the new altitude for the current item
6: **end for**
7: Update the UI

Steg 3:
Oppdater brukergrensesnittet med samme funksjon som ved baneutjevning. Se algoritme 3.

7.3.4 Implementering

Tidlig i utvikling, ble det implementert testfunksjoner i PlanMasterController, da dette objektet allerede kunne aksesseres fra QML. Videre i utvikling, ble tilleggsfunksjonaliteter, selv i ferdigutviklet form, lagt til her. Med ønsket pluginarkitektur, kunne dette flyttes til en egen PlanMasterController i mappen custom/src/. Ved kompilering, overskrives QGCs src/PlanMasterController med custom/src/PlanMasterController.

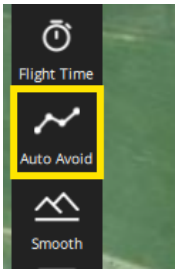
7.4 Resultater

7.4.1 Autokorrigering av kollisjonskurs

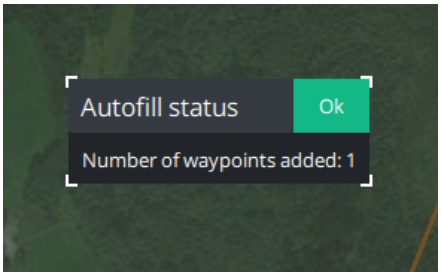
Ved trykk på ”Auto Avoid”, legger programmet om ruten til å stige over hindrende terreng. Løsnignen fungerer raskt og godt, med gode resultater, da algoritmen håndterer alt slags terreng uten utfordring. Antallet plasserte punkter er svært passende, da det gjør at ruten hverken kommer for nære bakken, eller at prosesseringstiden blir for høy.

Figur 7.15: Ruten kolliderer med en liten åstopp. Dette bises tydelig med rød linje både i kartet og høydeprofilen nederst i bildet.

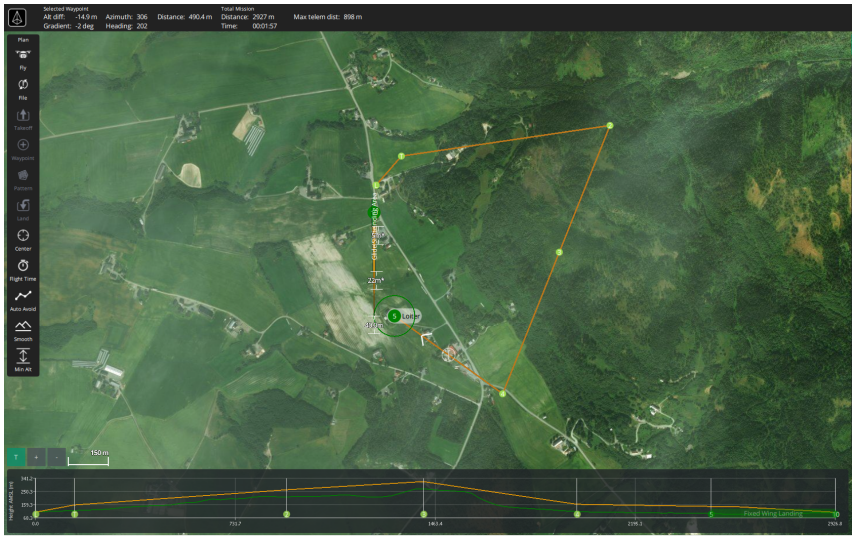
Etter funksjonen har lagt inn nye waypoints, informeres operatør om hvor mange punkter som ble lagt til. Dette gjøres med en popup slik vist i figur 7.17.



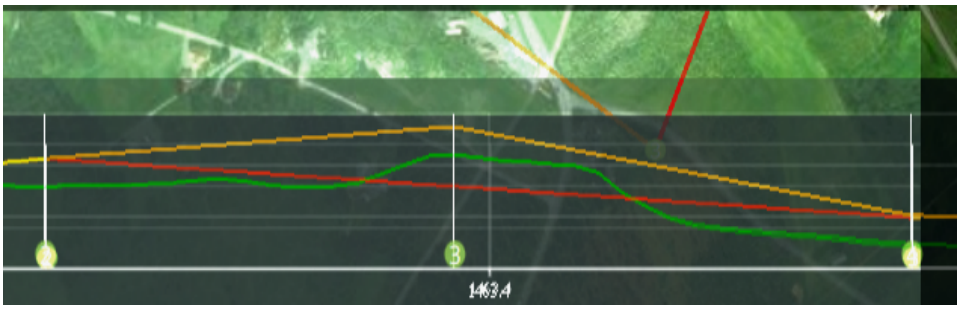
Figur 7.16



Figur 7.17



Figur 7.18: Etter trykk på *Auto Avoid*, oppstår det ikke lenger en mulig kollisjon. Ruten har lagt til et waypoint mellom tidligere punkt 2 og 3.

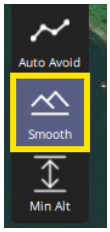


Figur 7.19: Figuren viser ruten før og etter lagt oppå hverandre.

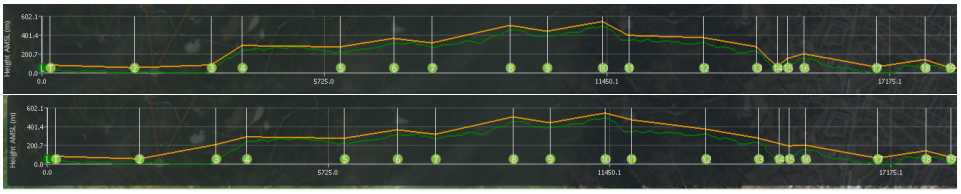
Figur 7.20: Figuren viser resultatet av en stresstest hvor det er mange hindringer. Programmet er ved noen segmenter nødt til å legge til flere punkter underveis for å kunne klarere hele ruten.

7.4.2 Baneutjevning

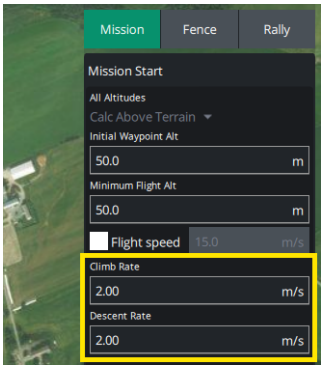
Ved trykk på knappen *Smooth* (se figur 7.21), løfter QGC enkelte punkter slik at stigningen til segmentene ikke er brattere enn stigningsratene tillater. Det skjer kun endringer til eksisterende waypointers høyde. Figur 7.22 viser hvordan QGC endrer ruten ved påtrykk av knappen. Stigningsratene i figuren er satt til $2m/s$ climb rate og descent rate for tyedelig effekt av funksjonaliteten.



Figur 7.21



Figur 7.22: Figuren viser hvordan *Smooth* glatter ut ruten. Merk at stignings- og nedstigningsratene her er overdrevent små for å vise effekten av knappen tydeligere.

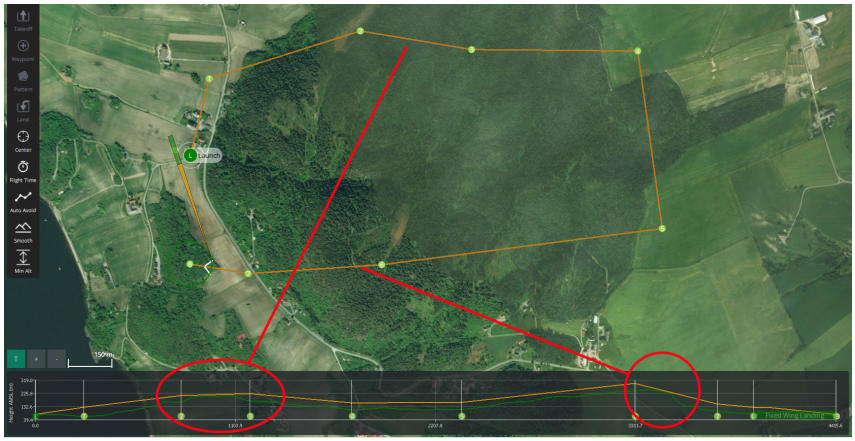


Figur 7.23: Stigningsrater kan justeres i mission settings til høyre på skjermen.

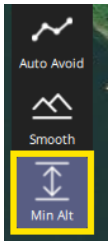
Funksjonen fungerer akkurat slik som den skal. Etter påtrykk er ruten så jevn som stigningsratene krever.

7.4.3 Minimumshøyde over bakken

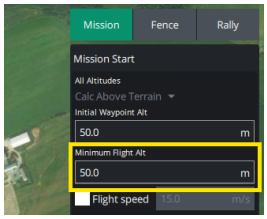
Ved trykk på *Min Alt*, løfter QGC de segmentene som er nærmere bakken enn hva minimumshøyden tillater.



Figur 7.24: Figuren viser ruten før minimumshøyde er sikret. Ved segment 2 og 6 kommer dronen farlig nær bakken.



Figur 7.25: Knappen for å sikre minimumshøyde ser slik ut og er plassert helt til venstre på skjermen.



Figur 7.26: Slik ser inntastingsfeltet for minimumshøyde ut. Det er plassert i mission settings helt til høyre på skjermen.

Figur 7.27: Ruten har gitt god klaring til terrenget, og har alltid en avstand på minst 50*m* fra bakken i henhold til innstillingene.

7.4.4 Endringslogg

Tabell 7.2 viser alle kodeendringer som er gjort under utviklingen av modul 2.

Filsti	**Filnavn**	**Tillegg**
custom\src\	PlanMasterController.h	Deklarere medlemmer: autoAvoid, smoothPath, minimalAlt, updateUI
custom\src\	PlanMasterController.cc	Inkludere headere: FlightPathSegments, qmlObjectListModel, simpleMissionItem, FlightMapSettings Definere medlemmer: autoAvoid, smoothPath, minimalAlt, updateUI
custom\res\	CustomPlanView.qml	Legg til knapper: Auto Avoid, Smooth, Minimal Altitude
src\Settings\	AppSettings.h	Definere settingsfact: minimalSegmentAltitude
src\Settings\	AppSettings.cc	Deklarere settingsfact: minimalSegmentAltitude
src\Settings\	App.SettingsGroup.json	Definere FactMetaData: minimalSegmentAltitude
src\Settings\	FlightMapSettings.h	Definere settngsfacts: cClimbRate, cDescentRate
src\Settings\	FlightMapSettings.cc	Deklarere settingsfacts: cClimbRate, cDescentRate
src\Settings\	FlightMap.SettingsGroup.json	Definere FactMetaData: cClimbRate, cDescentRate

Tabell 7.2: Endringslogg for modul 2

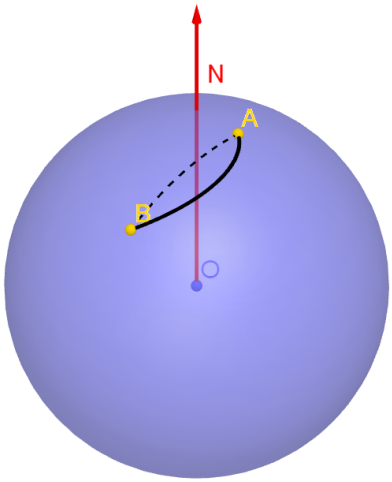
7.5 Analyse og diskusjon

7.5.1 Tilpass ruten til terrenget

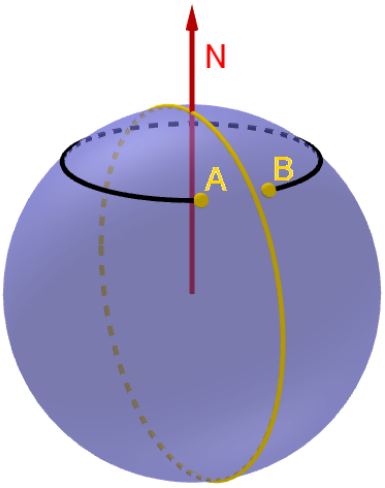
Algoritmen som plasserer nye waypoints kan garantere å finne en rute for alle typer terreng relativt raskt. Den kan også gjøre dette med ganske lav mengde nødvendige punkter underveis. Likevel, er det mer optimale algoritmer som kan benyttes for å oppnå like gode eller bedre resultater. Grunnet begrenset utviklingstid og marginal forbedring i programmets ytelse ved å optimalisere algoritmen, ble dette nedprioritert til fordel for å gi mer utviklingstid til andre programområder. Dersom man ønsker å legge til færrest mulig punkter underveis, vil oppløsningen bli lav, tilpasningen til terrenget svekkes, og man kan risikere å komme farlig nær bakken i enkelte tilfeller. Samtidig, kan ytelsen til programmet belastes dersom det plasseres for mange punkter.

Videreutvikling av kollisjonsunngåelse ville innebære å justere algoritmen slik at den fordeler punktene bedre utover et segment. Slik algoritmen fungerer nå, vil den kunne gi stor avstand mellom de første par middelpunktene, for så å plassere ett rett ved endepunktet. Det ble kort utforsket ideer om å frekvensanalysere høydedata for hvert segment med FFT for så å tilpasse antall middelpunkter ut fra de dominerende frekvenskomponentene. En slik tilnærming ville forsøkt å analytisk komme frem til et terrengstykkes knekkpunkter og ville trengt mye teoretisk arbeid for selv før implementering. Dersom det heller skulle velges en generell analytisk metode for å bestemme beste nye punkter, kunne det vært hensiktsmessig å analysere de delene av høydeprofilen med størst negativ akselerasjon. Bunnpunktene til høydeprofilens andrederiverte fremkaller høydeprofilens nedovervendte knekkpunkter, og vil da være naturlig å forholde seg til dersom man ønsker å unngå slike utstikkende deler av terrenget.

Ved utplassering av nye punkter interpoleres det lineært mellom endepunktene til segmentet punktet skal plasseres på. Altså gjøres det lineær interpolering av kulekoordinater. Dette viser seg å være metoden QGC bruker for å vise hvor ruten går innad i programmet, da segmenter vises som kartesisk rette linjer over kartet.



Figur 7.28: Den striplede linjen viser korteste vei mellom A og B. Lineær interpolering av koordinater mellom dem går langs den heltrekte linja. Avviket er tydeligst ved lange avstander, likevel ville korrekt interpolering tatt hensyn til jordas krumming.



Figur 7.29: Også ved krysse over datolinjen, vil interpoleringen gå rundt hele jorda dersom det skal settes nye punkter mellom A og B. Dette er en sjelden feil, og ikke en som vil forekomme ved bruk innenfor Norges grenser. Likevel er det en svakhet med metoden.

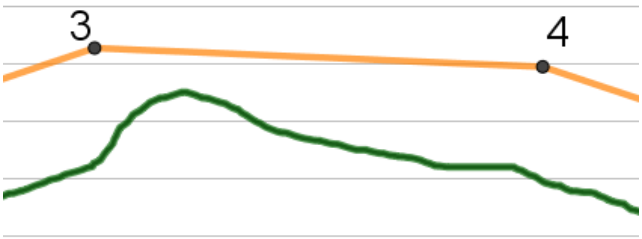
7.5.2 Jevn ut ruten for stigningsrater

For å finne ut hvor høyt et punkt må flyttes, evalueres stigningen til alle andre waypoints. Dersom punktet må flyttes, betyr det at ligger i skyggen til et annet waypoint. Det vil si at selv om et annet waypoint må flytte seg for å tilpasse seg det punktets høyde, er det et annet som trumfer. Det er derfor mer effektivt å ikke evaluere stigningen til de punktene som selv må flyttes. Dette ville spart minimal prosesseringskraft, som er årsaken til at det ikke er implementert. Alle slike endringer er tiltak som kan gjøres dersom det oppleves at programmet kjører mye tregere enn ønsket. Da er det mulig å gjøre små forbedringer må mange steder slik at programmet i en helhet kjører mye raskere.

I enkelte situasjoner, ved bruk av *Smooth*-funksjonen, vil et waypoint kunne havne midt i mellom to andre waypoints både i koordinater og høyde. I slike tilfeller, vil punktet i mellom være overflødig, og kan i grunn bare fjernes. Fjerning av waypoins på denne måten er ikke implementert, da ideen falt inn som en baktanke ved observasjon av slikt et fenomen under testing.

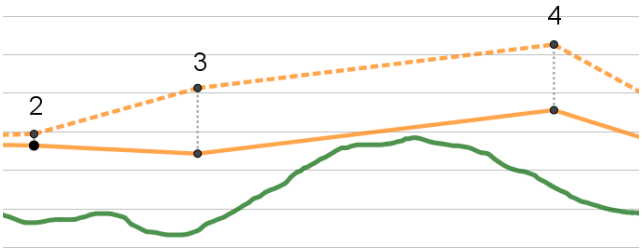
7.5.3 Sikre minimumshøyde over bakken

Ved utjevning av ruten er det satt et mål om å ikke gjøre større endringer enn nødvendig. Selv om det ikke er et krav om dette, er det alltid interessant å finne den meste effektive og fineste løsningen til et problem. Løsningen som er implementert, løfter hvert segment like mye på begge visualItems. Løsningen forflytter altså kun segmenter direkte vertikalt, med samme orientering og stigning. I enkelte situasjoner er det ikke nødvendig å flytte mer enn det ene.



Figur 7.30: Hinderet i terrenget er nærmere punkt 3, og er intuitivt å løfte punkt 3 mer enn punkt 4.

Videre, vil segmentets naboer oppleve skjev høydeendring, som overgår minstekravet drastisk, og vil i tillegg kunne risikere å bli brattere enn hva dronen er i stand til å fly.



Figur 7.31: Siden punkt 3 løftes såpass høyt, vil det ikke være nødvendig å heve punkt 2 for å unngå den lille hindringen mellom punkt 2 og 3. Dette gjøres likevel i den implementerte løsningen.

I tillegg er den valgte løsningen implementert med en begrensning som hindrer den i å flytte noen punkter nedover. Dette er gjort for å spare tid og å kunne garantere klaring

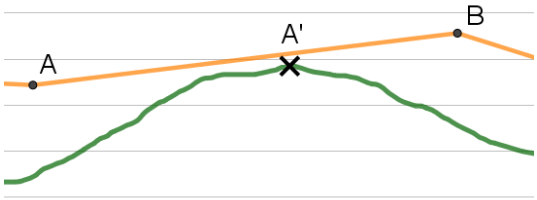
for minimumsavstanden som er satt. Dette er ikke en særlig stor ulempe, men det gjør at algoritmen er mindre tilpasningsdyktig i ujevnt terreng. I tillegg, vil man ikke kunne senke ruten, dersom man ønsker å sette en lavere minimumshøyde senere.

En bedre tilnærming til denne funksjonen vil kunne oppfylle følgende krav:

- Gjøre minst mulig endringer til høyden.
- Gjøre ruten gjennomførbar med hensyn til stignings- og nedstigningsrater.
- Heve hvert visualItem individuelt.
- Tillat forflytning både opp og ned.
- Garantere klaring for minimumsavstanden til bakken.
- Være en prosesseringsmessig effektiv løsning.

Under følger forarbeidet som ble gjort for å ettersøke om det ville være mulig realisere en slik løsning.

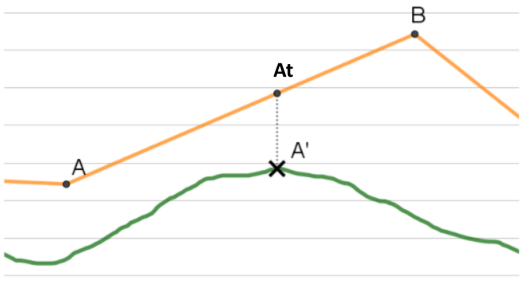
Det vil være ett punkt på hvert segment som befinner seg nærmest den planlagte ruten. La oss kalle dette det kritiske punktet. Simplifisert, kan man tenke at rutens eneste bekymring er å sikre minimumsavstanden over det kritiske punktet. Se figur 7.32.



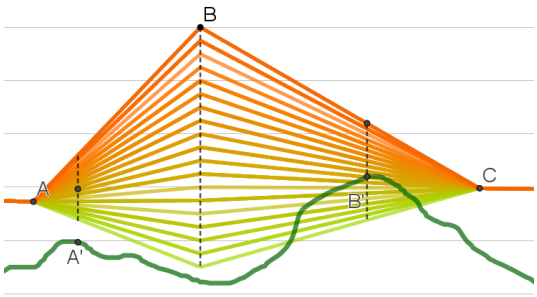
Figur 7.32: Det kritiske punktet A' er det punktet langs segmentet hvor ruten ligger nærmest bakken. Dette punktet er det eneste som er avgjørende å holde avstand til.

Gå ut ifra at minimumsavstand til bakken er 50m. For å ikke gjøre større endringer enn nødvendig, skal ruta være 50m over bakken ved det kritiske punktet. Videre, vil det være et slikt kritisk punkt langs neste segment. Med samme logikk, skal ruta krysse 50m over bakken langs dette kritiske punktet også. Se figur 7.33. Problemstillingen omformuleres til at det skal bestemmes en rekke kritiske punkter som skal flys over med en gitt høyde h. Punktene h meter over kritiske punkter kalles terskelpunkter og navngis foreløpig At, Bt, Ct og videre.

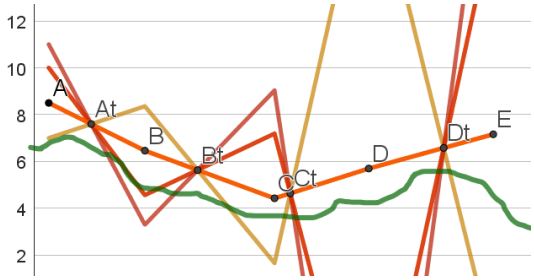
Oppgaven kan videre formuleres slik: Segmentet fra A til neste punkt må krysse gjennom At. Det samme gjelder for alle punkter unnatt det siste. Å finne en løsning på dette iterativt, vil være utfordrende, da høydejusteringer gjort i punkt B påvirker høyden ved både A' og B' (se figur 7.34). Videre, vil endringer gjort tidlig i ruten få større og større virkninger for senere segmenter (se figur 7.35). Med en klarere definisjon på hva problemet går ut på, kan det hele formuleres med et sett lineære ligninger, som videre kan fås over på matriseform.



Figur 7.33: Det kritiske punktet skal flys over med akkurat minimumshøyden. Dette brukes som utgangspunkt for å sikre at det ikke gjøres større endringer enn nødvendig. Ruten må krysse gjennom punkt At.



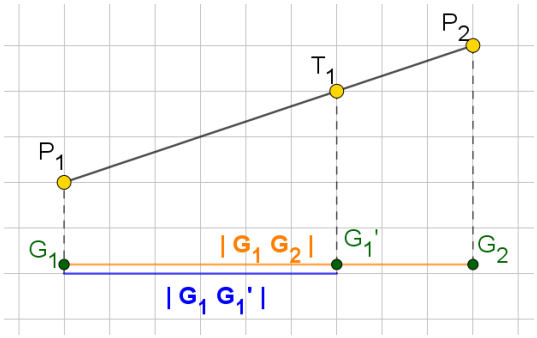
Figur 7.34: Forflytting av punkt B påvirker høyden over både A' og B'



Figur 7.35: Forflytting av punkt A kan gi store utslag for kommende punkter. Lengre ruter øker sannsynligheten for at det forekommer minst ett punkt som kraftig forsterker endringer i A.

Herfra er det gjort noen endringer i variabelnavn fra hvordan de tidligere er omtalt i dette kapitlet. Dette gjøres for å få mer konvensjonell matematisk notasjon.

$$n : \text{Antall segmenter} \tag{7.7}$$
$$P_n : \text{Startpunktet til et segment} \tag{7.8}$$
$$P_{n+1} : \text{Endepunktet til et segment} \tag{7.9}$$
$$T_n : \text{Terskelpunktet til et segment} \tag{7.10}$$
$$G_n : \text{Bakkekoordinatene til } P_n \tag{7.11}$$
$$G_{n+1} : \text{Terskelpunktet til et } P_{n+1} \tag{7.12}$$
$$G' : \text{Kritisk punkt langs segment n} \tag{7.13}$$



Figur 7.36

Regner ut hvor langt langs hvert segment det kritiske punktet ligger.

$$r_1 = \frac{|G_1 G_1'|}{|G_1 G_2|} \tag{7.14}$$
$$r_2 = \frac{|G_2 G_2'|}{|G_2 G_3|} \tag{7.15}$$
$$\dots \tag{7.16}$$
$$r_n = \frac{|G_{n-1} G_{n-1}'|}{|G_{n-1} G_n|} \tag{7.17}$$

... hvilket gir følgende sammenheng mellom endepunkter til et segment og dets terskelpunkt.

$$P_{1,y} \cdot (1 - r_1) + P_{2,y} \cdot r_1 = T_{1,y} \tag{7.18}$$
$$P_{2,y} \cdot (1 - r_2) + P_{3,y} \cdot r_2 = T_{2,y} \tag{7.19}$$
$$\dots \tag{7.20}$$
$$P_{n-1,y} \cdot (1 - r_{n-1}) + P_{n,y} \cdot r_{n-1} = T_{n-1,y} \tag{7.21}$$
$$\tag{7.22}$$

Ønsker å løse ligningsettet på matriseform hvor:

A : A-Matrise
x : Ønskede nye høydeverdier for punktene
t_y : Høyden til terskelpunktene
h : Høydeforskjeller mellom terskelpunkter og kritiske punkter
u : Standardisert variabel for å justere $P_{1,y}$
v : Standardisert variabel for å justere h-vektor

$$Ax = t_y + h \quad (7.23) \qquad x = \begin{bmatrix} P_{1,y} \\ P_{2,y} \\ \dots \\ P_{n-1,y} \\ P_{n,y} \end{bmatrix} \quad (7.24) \qquad t_y = \begin{bmatrix} T_{1,y} \\ T_{2,y} \\ \dots \\ T_{n-1,y} \\ u \end{bmatrix} \quad (7.25) \qquad h = \begin{bmatrix} v \\ v \\ \dots \\ v \\ 0 \end{bmatrix} \quad (7.26)$$

Skriver om på matriseform slik vist i figuren til høyre (7.27).

$$A = \begin{bmatrix} r_1 & 1-r_1 & 0 & 0 & 0 \\ 0 & r_2 & 1-r_2 & 0 & 0 \\ 0 & 0 & \dots & \dots & 0 \\ 0 & 0 & 0 & r_{n-1} & 1-r_{n-1} \end{bmatrix} \quad (7.27)$$

Legger inn et ekstra krav slik at matrisen blir kvadratisk. Nederste rad bestemmer høyden $P_{1,y}$. *A*-matrisen ender opp slik vist i ligningen til høyre (7.28).

$$A = \begin{bmatrix} r_1 & 1-r_1 & 0 & 0 & 0 \\ 0 & r_2 & 1-r_2 & 0 & 0 \\ 0 & 0 & \dots & \dots & 0 \\ 0 & 0 & 0 & r_{n-1} & 1-r_{n-1} \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (7.28)$$

Fullt ligningssett på matriseform:

$$\begin{bmatrix} r_1 & 1-r_1 & 0 & 0 & 0 \\ 0 & r_2 & 1-r_2 & 0 & 0 \\ 0 & 0 & \dots & \dots & 0 \\ 0 & 0 & 0 & r_{n-1} & 1-r_{n-1} \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_{1,y} \\ P_{2,y} \\ \dots \\ P_{n-1,y} \\ P_{n,y} \end{bmatrix} = \begin{bmatrix} T_{1,y} \\ T_{2,y} \\ \dots \\ T_{n-1,y} \\ u \end{bmatrix} + \begin{bmatrix} v \\ v \\ \dots \\ v \\ 0 \end{bmatrix} \quad (7.29)$$

Løser for x:

$$x = A^{-1}(t_y + h) \quad (7.30)$$

Finn beste u og v

Merk at det er to frie variabler: u og v. u representerer høyden til P_1 spesifisert av nederste rad i matrisen. Variabelen v gir muligheten til å løfte terskelpunktene enda lengre opp dersom det ikke finnes en løsning for gitt terskelhøyde. Altså vil det være 2 frihetsgrader tilgjengelig: u og v. For ligningssettets skyld er det overflødig med to ekstra frihetsgrader. Det trengs kun 1, da A-Matrisen i utgangspunktet var overdimensjonert av formen $R^{n-1,n}$. Grunnen til at det kan være nødvendig med 2 frihetsgrader er at dersom det ikke finnes en gyldig løsning for alle forskjellige u, må terskelpunktene heves. Derav to frie variabler: u og v.

Hver enkelt punkt i uv-planet vil være en måte å justere høyden til waypointene slik at de holder en viss avstand til de kritiske punktene. Enkelte områder i dette planet vil være ugyldige løsninger, da de medfører kollisjoner med bakken eller for bratte segmenter. Blant de løsningene som er innenfor gyldig sone, vil det være enkelte løsninger som er mer effektive enn andre. Det ønskes å finne den løsningen som gjør minst endringer til ruten som helhet.

$x(u,v)$: Nye høyder for hvert waypoint.
x_0 : Gamle høyder for hvert waypoint.
P : Funksjon for å regne ut kostnaden til forskjellige løsninger.

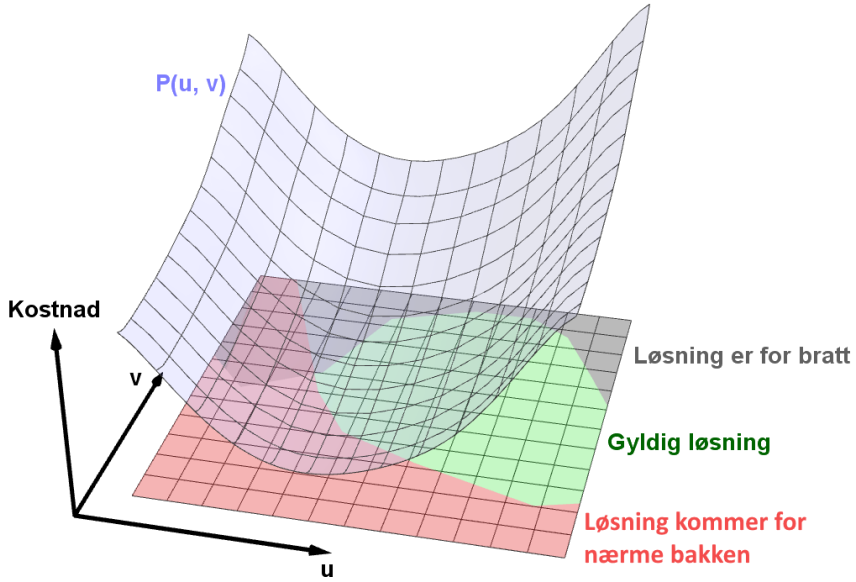
Gyldige verdier for u og v som gir lavest P bestemmer løsning for x.

Her er det valgt å definere P som minste kvadratsum av waypoints og terskelpunkters høydeendring. Endring i waypoints gis av $\Delta x = x - x_0$, mens endring i terskelpunkter gis av t_y.

P defineres dermed slik:

$$P(u,v) = (|x(u,v) - x_0|)^2 + (|t_y(v)|)^2 \tag{7.31}$$

P definerer en flate i uv-planet vist i figur 7.37. Optimal løsning ligger i laveste gyldige punkt innenfor den gyldige uv-sonen. Samtidig, siden det ønskes en løsning for x som endrer minst mulig, antas det at beste løsning ligger et sted på grensen mellom gyldige og ugyldige uv-verdier.



Figur 7.37: Et eksempel på hvordan kostnadsfunksjonen P og diverse soner i uv-planet kan se ut. Billigste gyldige løsning er laveste punkt på P som også ligger i det grønne området.

Problemet endres igjen til hvordan å raskest finne laveste gyldige punkt P_{min} på flaten. Det er ikke eksplisitte funksjoner som bestemmer dette, men heller diverse sjekker som gjøres manuelt ved å iterere gjennom terrenghøyde. Av slike årsaker, er det ikke mulig å finne en eksakt analytisk definisjon for områdene i uv-planet. Derfor er metoden for å finne P_{min} samplingsbasert. Prosedyren er som følger:

- Plasser ut et antall tilfeldige punkter i uv-planet. Flere punkter gir bedre resultater, men øker prosesseringstiden.
- Ta utgangspunkt i det gyldige punktet med lavest P-verdi.
- Flytt punktet i motsatt retning av gradienten sin på flaten P. Dette lar punktet trille nedover på flaten.
- Evaluer om dette er en gyldig løsning.

- Prosessen gjentas frem til siste gyldige punkt er funnet. Punktets uv-verdier gir vektoren x som bestemmer alle nye waypoint-høyder.
- Deretter implementeres endringene ved å iterere over missionItems og å sette waypointenes nye høyder i samsvar med x-vektor.

Ulemper ved denne metoden

Dette er en tenkt fremgangsmåte, og det er ikke gjort noen tester på hvor effektiv algoritmene er. Det kreves mye utregning med tunge matematiske modeller som kun blir større og større for lengre ruter. Dette kan ta tid å regne ut for ruter på 20+ waypoints. For ruter som skal scanne et større område i mønster kan komme opp i flere hundre waypoints, noe som kan gjøre all utregning og evaluering ekstremt tungt å gjennomføre.

Det er en vesentlig mulighet for at løsningene fra de to forskjellige fremgangsmåtene ikke er merkbart forskjellige. I tillegg er det heller ikke sikkert at den alternative fremgangsmåten er optimal. Konseptet om kritiske punkter og terskelpunkter er en heuristikk som mulig kan ha alternativer som gir bedre resultater.

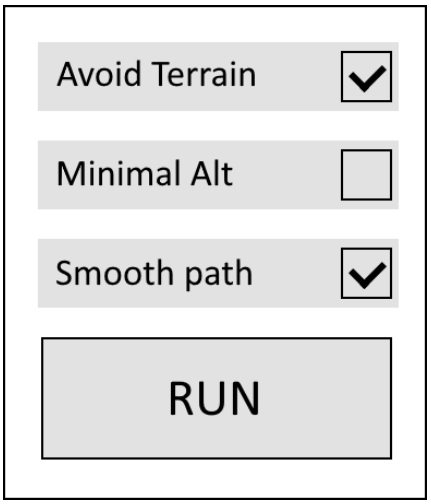
Litt av målet med en slik fremgangsmåte som dette er at det ikke skal gjøres større endringer enn nødvendig. Da er det mot sin hensikt at variabelen v flytter alle terskelpunkter likt. En bedre tilnærming kunne vært å ha individuelle høydeverdier for alle terskelpunkter. Dette ville gitt mye større frihetsgrad, og kunne gitt mer effektive løsninger. Slik tilpasningsdyktighet har også ulemper, blant annet at flere frihetsgrader også medbringer lengre prosesseringstid for å finne beste løsning.

7.5.4 Oppdatering av UI

Løsningen som er valgt er ikke særlig effektiv, da alle verdier i hvert visualItem tvinges til å oppdatere. Likevel er det en god nok løsning, da dette kun skjer på slutten av andre tunge operasjoner som bare kjøres av og til. Det påvirker da ikke brukeropplevelsen tilstrekkelig til å være verdt tiden det hadde tatt å spore ned alle nødvendige signaler som må varsles. Alternativt kunne man kjørt diverse oppdateringsfunksjoner selv, men det hadde trolig tatt noen dager med søk i koden.

Individuelle knapper

Et av hovedformålene ved denne modulen er å automatisere repetitivt arbeid for operatør. Det er derfor et argument for å samle alle funksjonsknappene under én og samme knapp. Denne knappen vil da være en endelig fiks for alle problemer som kunne oppstå langs ruta.



Figur 7.38: Forslag til oppsett hvor alle funksjonaliteter fra modul 2 kan samles under samme knapp.

Problemet med denne ordningen er at enkelte verdier ikke får tid til å oppdatere seg. Første funksjon i rekka kunne kjøre, men programmet kræsjet ved kall på funksjon nummer to, da visse nøkkeltall hos segmenter returnerer NaN ved rekalibrering. Grunnet tekniske utfordringer ved en slik løsning, samt potensielt behov for operatør om å kun gjennomføre enkelte, men ikke alle funksjoner, ble forsøket avlyst. Alle funksjonene har vær sin knapp med eget ikon, og for prosjektets omfang, anses det som tilstrekkelig.

7.6 Kapittel konklusjon

I den offisielle versjonen av QGC, er det mangel på enkelte automatiske funksjoner. Funksjonene omhandler å korrigere kollisjoner med bakken, endre stigningsrater til å

være innforbi fartøyets begrensninger, og å sikre at minimumsavstand til bakken er overholdt. For å møte disse behovene, er det lagt inn knapper som tar seg av hver av disse funksjonene: *Auto Avoid*, *Smooth*, og *Min Alt*.

Utviklingen har innebært å legge til knapper i alleredeeksisterende områder i brukergrensesnittet, samt å lage funksjoner i C++ som tar seg av nødvendige beregninger. I hovedsak, har algoritmene geometrisk tilnærming med fokus på rask utviklingstid samt lav prosesseringtid ved bruk.

Det ble gjort et grundig forarbeid for en potensiell løsning for å sikre minimumshøyde over bakken. Etter arbeidet som ble gjort rundt teorien av løsningen, ble det konkludert at tiden som gjensto i av arbeidspakken ikke var tilstrekkelig for å kunne implementere løsningen på et vanntett vis. I tillegg manglet det teoretiske verktøy i C++, eksempelvis å kunne invertere store matriser raskt eller å gjennomføre gradient descent i flere dimensjoner. Derfor ble det beholdt den allerede implementerte løsningen, som også oppfylte mange av de sentrale kravene til funksjonen.

Enkelte funksjoner i modul 2 krever at brukergrensesnittet oppdateres manuelt i etterkant av påførte ”Fact”-endringer. Til dette ble det funnet en reserveløsning, da ideell løsning ikke ga ønskede resultater. Reserveløsningen fungerer tilstrekkelig for nødvendig funksjonalitet, da programmet hovedsakelig er designet til intern bruk hos kunde. Dersom denne versjonen av QGC skulle vært offentlig tilgjengelig på nett, ville det vært større insentiv rundt det å gjøre koden mindre prosessorbelastende. Videre arbeid for å slippe en reserveløsning, ble nedprioritert til fordel for mer utviklingstid til modul 3.

Videre arbeid tilknyttet modul 2, kan innebære å se nærmere på følgende problemer:

Interpoler mellom koordinater i henhold til korteste luftstrekning på en kule.
Effektivisere søkeprosessen ved baneutjevning.
Sammenkoble oppdatering av brukergrensesnitt til programmets interne systemer.
Koble sammen alle funksjoner i én knapp med avkrysning for ønskede funksjonaliteter.

Kapittel 8

MODUL 3: Estimer flytid med hensyn til vind

8.1 Introduksjon

Dersom man skal foreta egen droneflyvning, er det en rekke faktorer som kan havarere fartøyet ditt. Den planlagte ruten kan være helt i orden, men likevel kan sterk vind være forskjellen mellom å holde eller drive vekk fra kursen. QGC støtter ikke offisielt noen former for advaring mot sterke vinder. I tillegg, tar ikke programmet hensyn til at sterk medvind kan korte ned flytid betraktelig. Eller at sterk motvind kan tillate brattere stigninger enn til vanlig. Det er derfor implementert et system som tar hensyn til dette, som er hoveddelen i modul 3. Modulen bygger videre på automatikken gitt av modul 1 og 2, og legger til rette for at operatører av programvaren kan ta informerte beslutninger om flyruten og hvordan været påvirker forholdene. For å oppnå dette har det blitt lagt vekt på brukervennlighet og maksimal nyttig informasjonsmengde uten at det skal være forstyrrende elementer. Metoden som er brukt er en geometrisk løsning basert på fartøystyring av små luftfartøy, satt sammen med en eulersolver for å simulere flyruten gjennom tid og rom.

8.2 Teoretisk rammeverk

8.2.1 Lineær Interpolering

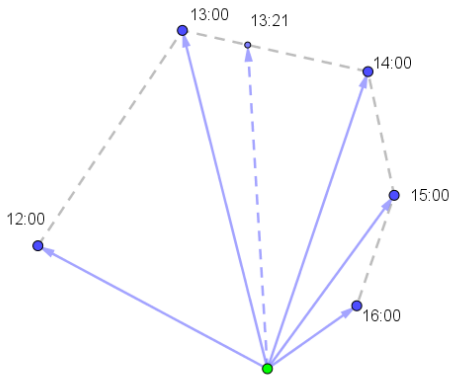
Interpolering er å glatte overgangen mellom to verdier. Da vinddata kun gis ved hele klokkeslett, brukes det i denne sammenhengen til å få kontinuerlig vinddata for alle tidsrom i mellom. Det er en rekke metoder innenfor interpolering. Den vanligste er lineærinterpolering. Dette er raskeste og enkleste metode, men den produserer ikke glatt data. Alle vindverdier mellom datapunktene i tid gis fra en ligning. Ligningen plotter vindverdiene med t som parameter. $t = 0$ gir P_1, og $t = 1$ gir P_2. Ligningen er gitt:

$$f(t) \approx P_1 \cdot (1-t) + P_2 \cdot t \tag{8.1}$$

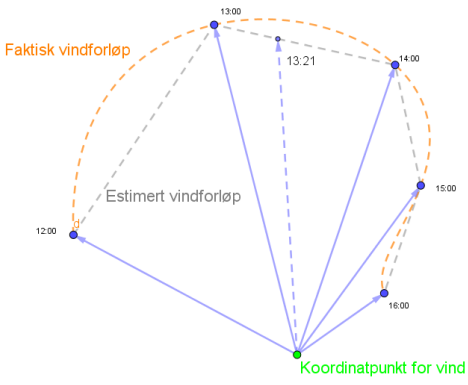
Figur 8.1: (Freya Holmér, 2022)

Etterhvert som t går fra 0 til 1, går L_1 fra å være lik P_1 til å bli lik P_2.

En ulempe med denne metoden er at interpoleringen får knekkpunkter, som den faktiske endringen i vind sjelden har. Dersom vinden endrer seg gradvis, med glatte overganger over tid, vil den estimerte vindstyrken i snitt være en del lavere enn hva den egentlig er.



Figur 8.2: Lineær interpolering gir en enkel tilnærming til vind mellom data i tid.

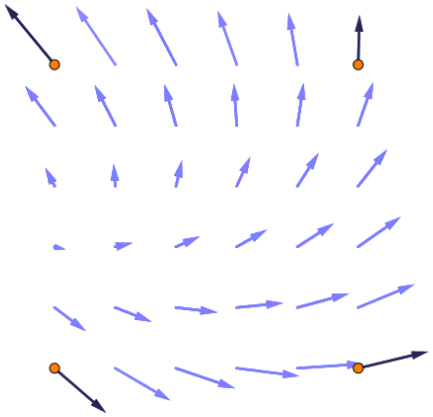


Figur 8.3: Den interpolerte modellen vil generelt forkorte vektorene i forhold til hva de egentlig er.

8.2.2 Bilineær Interpolering

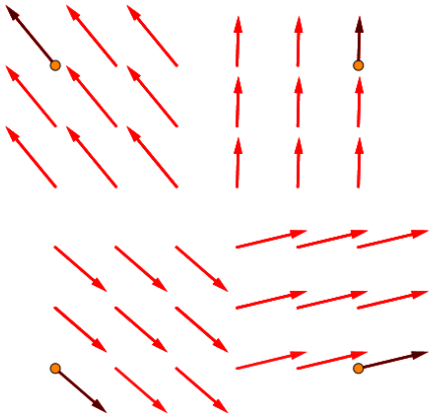
Dersom man ønsker større oppløsning mellom datapunkter i bakkeplanet, kreves bilineær interpolering. Denne metoden kan glatte overgangen mellom data fra fire hjørner av et kartstykke. Yr gir data med en horisontal oppløsning på 2.5km, som kan gi brå overganger mellom vindvektorer rett ved hverandre (Yr, u.å.). Eksempel vist under.

Vind fra interpolert modell:



Figur 8.4: Bilineær interpolering gir en enkel tilnærming til vind mellom datapunkter i bakkeplanet. Dette gir også glatte overganger.

Vind rett fra YR:



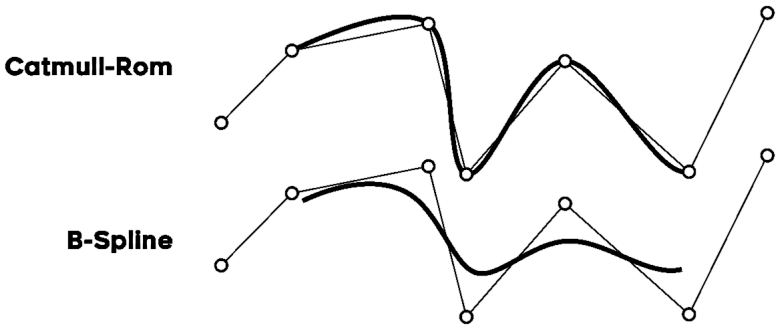
Figur 8.5: Yr leverer vinddata med metoden nærmeste nabo. Forespørsler for vind mellom datapunkter gir nærmeste datapunkts verdi.

Formelen for bilineær interpolering er en todimensjonal versjon av lineærinterpolering. Det tenkes at kvadratet mellom nedre venstre hjørne og øvre høyre hjørne trekkes ned til et kvadrat med sidelengder $x = y = 1$. Formelen for lineariseringen er så gitt ved:

$$f(x,y) \approx f(0,0)(1-x)(1-y) + f(1,0)x(1-y) + f(0,1)(1-x)y + f(1,1)xy \quad (8.2)$$

8.2.3 Ulineær interpolering

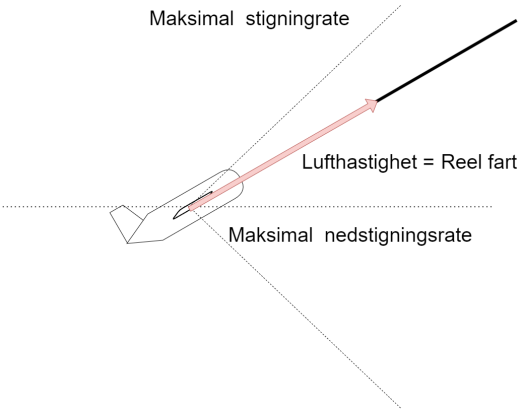
Dersom man ønsker en glattere funksjon for tilnærming, kan man bruke ulineære interpoleingsmetoder. Splines er en gruppe algoritmer for ulineær interpolering som brukes mye i CAD programvare, robotikk og i datagrafikk (Ogniewski, Jens, 2019). Det finnes mange varianter av spline, men fellestrekkene er at de tar inn et sett med punkter og gir ut et polynom som definerer en bane mellom minst to av punktene. Utover disse fellestrekkene kan forskjellige varianter av splines ha veldig forskjellig karakteristikk. De viktigere av disse egenskapene til interpolering av vindvektorer er parametrisk kontinuitet, og om den genererte kurven interpolerer gjennom settet med punkter, og ikke rundt dem. parametrisk kontinuitet sikter til glattheten ved en funksjon. En kontinuerlig funksjon har parametrisk kontinuitet C_0, der for at en funksjon skal ha C_n parametrisk kontinuitet må n-te derivat av funksjonen være sammenhengende. Enkelte splines følger kontrollpunktene i liten grad, da de genererer svært glatte baner som kun holder seg i nærheten av punktene uten å være innom dem Freya Holmér, 2022. For slike splines, er kun start- og sluttpunktene til kurven og kontrollpunktene like, eksempelvis har B-spline denne karakteristikken. Catmull-Rom, derimot, krysser innom alle kontrollpunkter. Se figur 8.6.



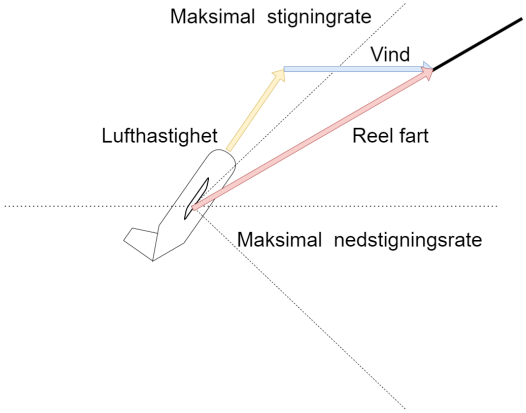
Figur 8.6: Figuren viser hvordan Catmull-Rom rører alle kontrollpunkter, mens B-Spline kun trekkes svakt i kontrollpunktenes retning. I oppgaven brukes Catmull-Rom. Hentet fra (Freya Holmér, 2022)

8.2.4 Fastvinge flyvekarakteristikk

Å ha faste vinger på en drone kan gi store økninger i effektivitet, men kommer også med noen begrensninger. Den største og mest relevante begrensningen er den manglende fleksibiliteten sammenlignet med multikopter, helikopter, og andre "Vertical Take-Off and Landing (VTOL)" fartøy. En av de største forskjellene er begrensninger ved stigningrater dronen trygt kan operere ved. Disse begrensningene kan gi problemer spesielt i situasjoner hvor kraftig nok vind blåser i samme retning som fartøyet ønsker å følge. I slike situasjoner kan det hende at den eneste orienteringen av fartøyet, som gir totalhastighet riktig retning, ikke er mulig å gjennomføre. Det betyr at enkelte ruter kun kan gjennomføres under riktige vindforhold. Andre fartøy som har annerledes flyvekarakteristikk og mer fleksibilitet kan det da hende ikke har problemer med mange av disse strekkene i det hele tatt.



Figur 8.7: Bratt stigning uten vind. Fartøyets pitchvinkel innenfor grenser for stigningrate og nedstigningsrate, og er derfor gjennomførbar.



Figur 8.8: Bratt stigning med vind. Fartøyets pitchvinkel utenfor grenser for stigningrate og nedstigningsrate, og er derfor ikke gjennomførbar.

8.2.5 Høydejustert vindestimering

Høydene som en drone operer ved kan variere mellom oppdrag, men er ofte under 120m over grunnet generelt lovverkLuftfartstilsynet, u.å. Værdata som hentes fra YR er oppgitt ved 10m over bakken(Yr, u.å.). Det er dermed nødvendig med en måte å estimere vind ved de forskjellige høydene dronen skal operere ved for et gitt punkt. En måte å gjøre det på som er et godt estimat opp til en høyde på omtrent 150m er Hellmaneksponent-loven,

som brukes mye til estimering av vind til planlegging og kartlegging av vindmølleparkerFrancisco mfl., 2011. Formelen er basert på at man vet hva vindstyrken er ved en gitt høyde nærme bakken, og en friksjonkoeffisient som kalles Hellmaneksponenten. Denne Hellmaneksponenten velges empirisk basert på terrenget utifra terrengtypene med tilhørende koeffisienter oppgitt i tabell 8.1.

$$V \approx V_0(\frac{H}{H_0})^{\alpha} \tag{8.3}$$

Figur 8.9: Hellmaneksponent-loven. V er vinden ved høyde H, gitt at man vet vinden V_0 ved høyden H_0 og Hellmaneksponenten for terrenget α. Hentet fra formel 3 i (Francisco mfl., 2011)

Landskaps type	**Friskjons koeffisient α**
Innsjøer, hav og harde glatte underlagg	0.10
Gressletter (bakkenivå)	0.15
Høye avlinger, hekker og busker	0.20
Tett skog	0.25
Liten by med noen trær og busker	0.30
Byområder med høyhus	0.40

Tabell 8.1: Verdier for Hellmaneksponenten gitt terrengtype. Hentet fra Table 1 i (Francisco mfl., 2011)

8.2.6 Vilkår for bruk av Yr REST-API

Vinddata er nødvendig for å gjennomføre alle kalkulasjoner i tidligere avsnitt. Ved bruk av Yr sitt "REST API" må enkelte krav oppfylles.

Begrensning av antall forespørsler

Forspørsler av vær er geografisk presise inntil fire koordinatdesimaler. Dette tilsvarer rektangulære områder på rundt $1km^2$. Det er derfor ansett som overflødig å be om værdata for lokasjoner rett ved hverandre, da data vil være den samme. I tillegg kan brukeren maksimalt gjennomføre 20 forespørsler i sekundet. Ved manglende overhold

av regler risikerer brukeren å miste tilgang.(Yr, 2023)

Identifisering gjennom User-Agent header

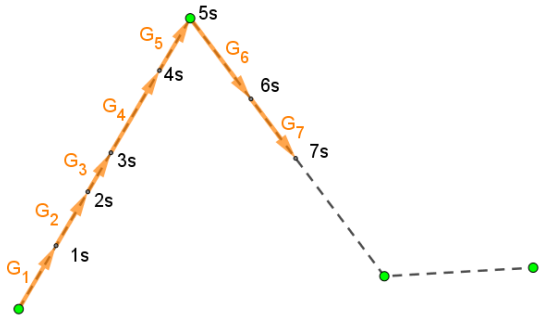
Alle brukere av Yr sitt REST-API må deklarere en ”User-Agent (UA)”. Det betyr å identifisere seg selv og oppgi hva data brukes til. UA formatet er "*Applikasjon/domenenavn, personlig epostadresse*". Ved manglende UA risikerer brukeren å bli blokkert fra tjenesten. (Yr, 2023)

8.3 Metode og utstyr

8.3.1 Beregne flytid basert på vindvektorer

Flytid er et resultat av hvilken bakkefart dronen er istand til å overholde i løpet av flyvningen. Bakkefarten er avhengig av hvordan ruten er lagt opp, dronens cruisehastighet, og vindvektoren ved hvert punkt på ruta. Det er andre faktorer til stede, slik som dronens karakteristikk, dynamikk, og styresystem, men disse anses ikke som vesentlige i sammenligning. Gitt vindvektorer for alle punkter langs flybanen, samt flykarakteristikken, vil dronen ha varierende bakkehastighet G med lengde G_S. Kontinuerlig summering av alle disse lengdene vil til slutt være like langt som total banelengde. Dette illustreres i figuren under. Merk at G_1, G_2 og videre er bakkehastigheten til dronen ved punktene hvor vektorene ligger.

Her kan man tydelig se at det tar 7 sekunder å fly litt over halvveis gjennom ruten. Det er også tydelig at dette er en iterativ prosess, da man ikke vet hvor stor G_2 er før man har funnet G_1. Det samme gjelder for G_3 i henhold til G_2 og så videre.



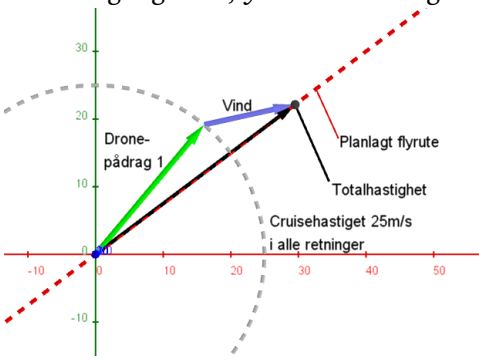
Figur 8.10: Antall fartsvektorer samsvarer med flytid.

Det er kun bakkehastigheten ved hvert punkt på ruten som trengs for å simulere hele flyvningen. Den kan hentes fra horisontalkomponenten til dronens totalhastighet. Dronen forsøker hele tiden å holde seg på rett kurs. Summen av dronens hastighet, relativ til vinden, og vindhastigheten skal derfor peke i retningen til neste waypoint.

Presisjonen til denne metoden kommer veldig an på oppløsningen til punktene. Hver iterasjon kan representere en ønsket mengde tid. For mer presisjon, men også lengre beregningstid, kan man velge små tidssteg. Dersom hver iterasjon representerer ett sekund, vil oppløsningen være god nok til å estimere minutter nøyaktig, mens sekundnøyaktigheten ikke vil være god nok. Ønskes det heller presisjonen på minuttnivå, greier det seg med større steglengder. Lengre steglengder gjør at beregningen går raskt, mens den fremdeles gir et godt estimat til hvor lang flyvningen blir.

Flyruten T er parameterisert med variabelen t slik at alle punkter på linja kan representeres med en t-verdi. Dronens flyretning finnes ved å sette totalfart V slik at $V + W = T(t)$. I praksis representerer ikke t noe substansielt. Den viser hvor stor andel av aktuelt segment som blir fløyet hver iterasjon. Med konstant vind, kunne dette vært nyttig, da $\frac{1}{t}$ tilsvarer flytiden for det segmentet.

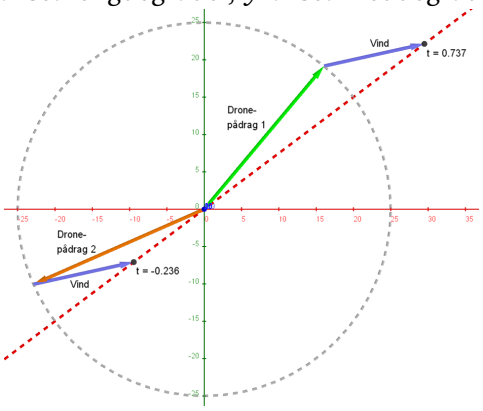
Figuren viser et 2D eksempel ovenfra.
x-akse: lengdegrader, y-akse: breddegrader.



Figur 8.11: Totalhastigheten skal ligge på linjen mellom nåværende posisjon og neste waypoint.

Det vil være to løsningner for hvilket dronepådrag som holder kursen, da det å fly bakover fremdeles telles som å være på riktig vei. Dermed kan det velges høyeste t-verdi for å gjøre videre utregninger, slik at man velger den løsningen som bidrar til størst fart i riktig retning. Her er det verdt å merke at dette viser bildet ovenfra og ned. For flyruter med stigning, kreves det en tredimensjonal modell, men prinsippet er likt for begge tilfeller.

Figuren viser samme 2D eksempel ovenfra.
x-akse: lengdegrader, y-akse: breddegrader.



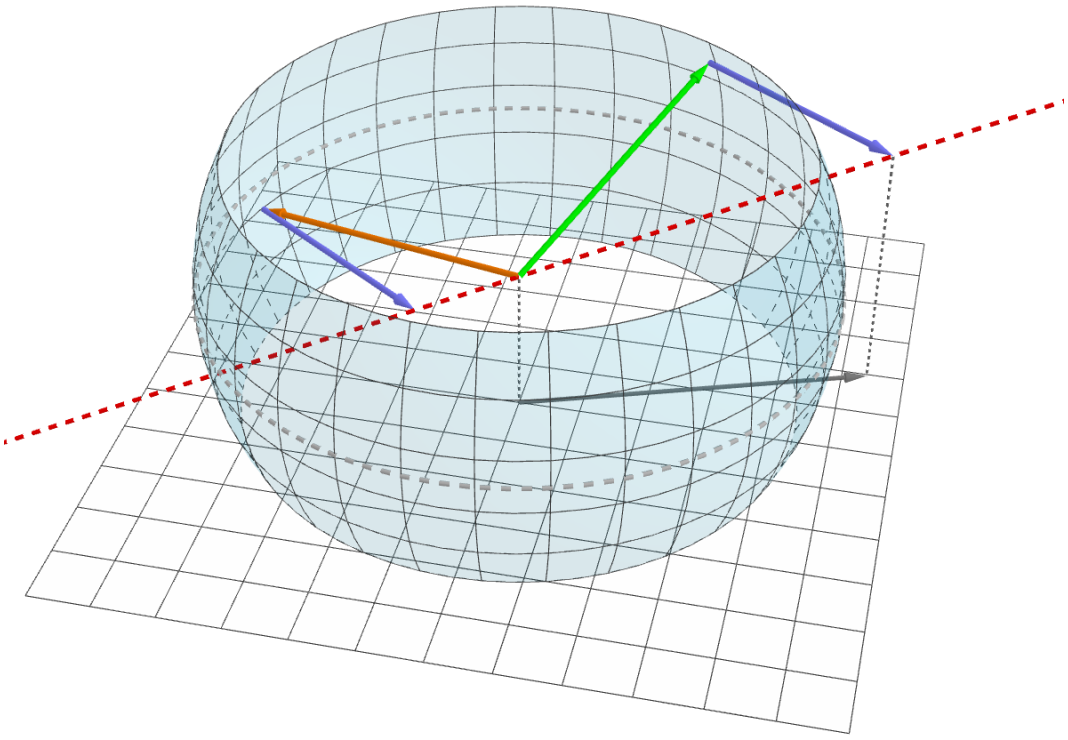
Figur 8.12: Det er to løsninger for dronens flyretning, da både F_1 (grønn) og F_2 (Oransje) får totalhastighet F til å ligge langs flyruten.

Ekstremtilfeller oppstår når vinden er sterkere eller lik 25m/s. Da vil løsningene kunne sammenfalle, eller begge være negative. Ved negative løsninger, velges den høyeste av de to, som vil si den som er minst negativ.

I tre dimensjoner blir dette veldig likt. Flyruten vil være en linje med både retning og stigning, samt at dronepådraget vil ligge på en sfære da dronen kan fly alle retninger begrenset til en pitch mellom minste og største stigningsrate. I figur 8.13, kan man se hvordan figur 8.12 utvides til tre dimensjoner. Merk: Her er stigningsratene begrenset til å være mellom $-27°$ og $27°$. I tillegg brukes det sterk vind for å tydeligere illustrere effekten vind har på bakkefarten. Vinden her er rundt 19m/s, som er utrygg vindhastighet for små luftfartøy.

- Flyrute
- Mulige dronepådrag
- Vindvektor
- Dronepådrag løsning 1
- Dronepådrag løsning 2
- Bakkehastighet

Merk at mulige dronepådrag ligger på et bånd langs midten av den blå sfæren. Der sfæren stopper er hvor pitchvinkelen overstiger dronens metning.



Figur 8.13: En tredimensjonell fremvisning av hvordan dronens flyretning påvirkes av styrken og retningen til vinden.

Utregning for G_s:

Vindvektor:

$$W = \begin{bmatrix} W_x \\ W_y \\ 0 \end{bmatrix} \tag{8.4}$$

Flyretning, parametrisert:

$$T(t) = \begin{bmatrix} t \cdot x \\ t \cdot y \\ t \cdot z \end{bmatrix} \tag{8.5}$$

Summen av dronepådrag og vind skal ligge langs flyruten:

$$F_x + W_x = T(t)_x = t \cdot x \tag{8.6}$$
$$F_y + W_y = T(t)_y = t \cdot y \tag{8.7}$$
$$F_z = T(t)_z = t \cdot z \tag{8.8}$$

Lengden av dronepådraget skal ligge rundt arbeidspunktet satt av cruisehastigheten Cs:

$$F_x^2 + F_y^2 + F_z^2 = Cs^2 \tag{8.9}$$

Skriver om slik at F er isolert:

$$F_x = t \cdot x - W_x \quad (8.10)$$
$$F_y = t \cdot y - W_y \quad (8.11)$$
$$F_z = t \cdot z \quad (8.12)$$

Setter inn F_x, F_y og F_z for ligning 8.9.

$$(t \cdot x - W_x)^2 + (t \cdot y - W_y)^2 + (t \cdot z)^2 = Cs^2 \quad (8.13)$$

Skriver om ligning 8.13 slik at t er på høyre side:

$$t = \frac{W_x x + W_y y + \sqrt{Cs^2(x^2+y^2+z^2) - W_x^2(y^2+z^2) - W_y^2(x^2+z^2) + 2W_x W_y xy}}{x^2+y^2+z^2} \quad (8.14)$$

Her brukes + i steden for ±. Med to løsninger for t, velges den høyeste, da den sikrer høyest bakkefart i retning neste waypoint. Se 8.12

Dronehastighet:

$$V(t) = T(t) = \begin{bmatrix} x \cdot t \\ y \cdot t \\ z \cdot t \end{bmatrix} \quad (8.15)$$

Horisontal dronehastighet:

$$G(t) = \begin{bmatrix} G_x(t) \\ G_y(t) \end{bmatrix} = \begin{bmatrix} x \cdot t \\ y \cdot t \end{bmatrix} \quad (8.16)$$

Ønsker å finne lengden til horisontal dronehastighet G_s:

$$|G| = \sqrt{G_x^2 + G_y^2} = \sqrt{(x^2+y^2) \cdot t^2} = t \cdot \sqrt{x^2+y^2} \quad (8.17)$$

Sjekker hvilken retning G peker:

$$dir = sgn(xG_x + yG_y) \quad (8.18)$$

Fart langs flyruten:

$$G_s = dir \cdot |G| \quad (8.19)$$

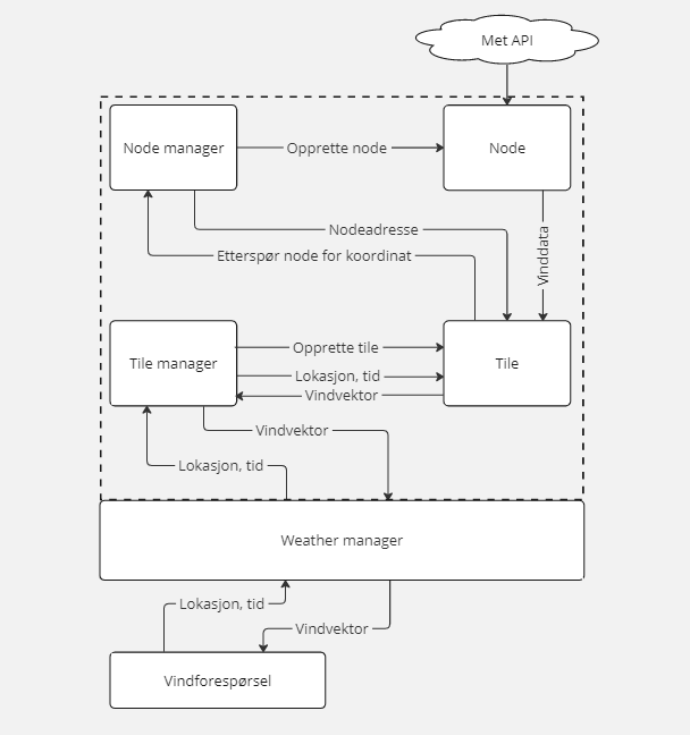
Variabler:

- Cs: dronens hastighet relativ til luften (cruisespeed) [m/s]
- dir: fortegnsvariabel som justerer G_s til å peke riktig vei [1 eller -1]

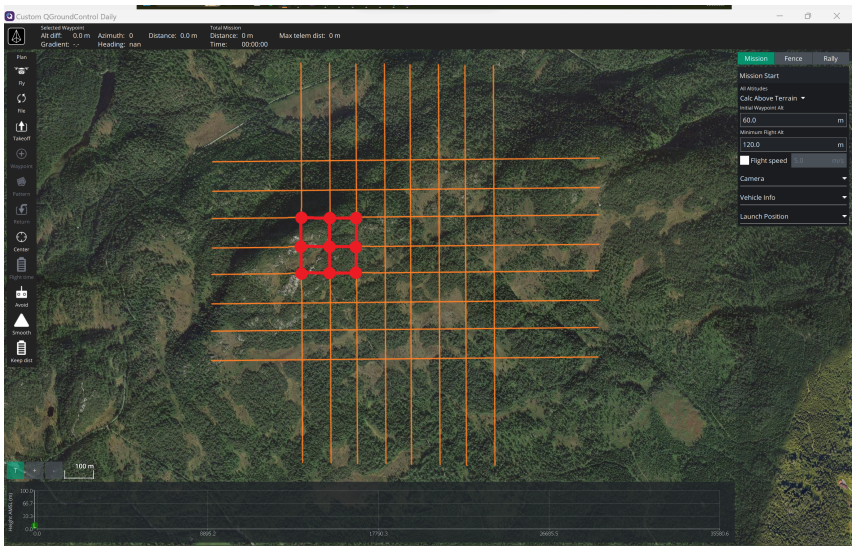
- F_x: dronens hastighetspådrag i x-retning [m/s]
- F_y: dronens hastighetspådrag i y-retning [m/s]
- F_z: dronens hastighetspådrag i z-retning [m/s]
- G: dronens bakkehastighet [m/s]
- G_s: dronens bakkefart i flyrutens retning [m/s]
- t: parametervariabel [ingen enhet]
- V: dronens hastighet [m/s]
- W: vindhastighet [m/s]
- x: avstandsforskjell i x mellom waypoints til aktuelt segment [m]
- y: avstandsforskjell i y mellom waypoints til aktuelt segment [m]
- z: høydeforskjell i z mellom waypoints til aktuelt segment [m]

8.3.2 Hente værdata

Vinddata lastes ned for å beregne flytid. Data blir lagret fordi nedlasting tar tid, og beslaglegger ressurser til APIet. For å ha et oversiktlig og skalerbart system er det strukturert på følgende måte, se figur 8.14 for en grafisk oversikt. Værdata som hentes inn for et koordinat lagres i et node-objekt. Nodene er spredt i en rutenettform med en distanse på 0.01°i lengde- og breddegrader, se figur 8.15. En tile er et objekt som består av fire noder, et i hvert hjørne. Det opprettes manager-objekter som administrerer om noder og tiles skal opprettes. Disse administrerer også eksisterende noder og tiles.



Figur 8.14: Skjemaet viser hvordan dataflyt går for å hente og administrere vinddata



Figur 8.15: Prikkene representerer hvordan noder er spredt. Strekene mellom nodene viser hvordan fire noder lager en tile.

Node

Noder har som oppgave å holde og administrere vinddata. Det innebærer at hver node selv laster ned data, sjekker om den er utgått, og oppdaterer innholdet.

Når node-objektet blir konstruet, opprettes et ”QNetworkAccessManager”-objekt som bestyrer innhenting av vinddata. QNetworkAccessManager henter inn vinddata ved å sende JSON forespørsler i gjennom Yr sitt REST-API. En forespørsel er en formatert URL. Formatet bestemmer hvordan data presenteres for mottaker. APIet har to formatvalg, men i dette prosjektet brukes ’/compact’-formatet. URLen skal inneholde posisjonen for å returnere data fra ønsket område. Et eksempel på en /compact-URL er:

https://api.met.no/weatherapi/locationforecast/2.0/complete?lat=63.418&lon=10.405

URLen returnerer JSON-objektet som QNetworkAccessManager-objektet etterspør. En kan undersøke JSON-objektet ved å laste URL inn i nettleser og se mengder med værdata. Datainnholdet lagres i en intern variabel som videre i programmet blir manipulert til å kun inneholde vinddata for 48 timer frem i tid.

I seksjon 8.2.6 nevnes vilkår for bruk av APIet. Et vilkår var å ikke gjøre unødvendige forespørsler, da det kan overlaste serveren. Løsningen finnes ved å sjekke et felt i pakkeheaderen. Headeren inneholder informasjon om svaret på forespørselen. I dette tilfellet undersøkes et tidsstempel som forteller når datainnholdet sist var oppdatert. Svaret som QNetworkAccessManager-objektet mottar inneholder en pakkeheader som er separat fra JSON-objektet. Pakkeheaderen inneholder blant annet et felt som sjekkes og oppdateres ved hver forespørsel. Feltet gjenkjennes som "IfModifiedSinceHeader"og er et tidsstempel som ved nodekonstruksjon informerer om sist gang APIet oppdaterte datainnholdet. Etter første forespørsel oppdateres feltet ved tidspunktet for neste gjennomførte forespørsel. Dersom noden ønsker å oppdatere datainnholdet må den sjekke feltet med inneværende tidspunkt. Betingelsen for å gjennomføre en forespørsel oppfylles ved at sist forespørsel er minst en time eldre enn den nye. På denne måten forhindres noder å be APIet om samme data flere ganger, og pekes heller mot programminnet hvor data fra forrige forespørsel er lagret.

```
    ...
        //_initialRequest = true in first program sequence
        if(!_initialRequest){
            _request.setHeader(QNetworkRequest::IfModifiedSinceHeader,_lastModified);
        }
        _initialRequest = false;
        _lastModified = QDateTime::currentDateTimeUtc();
    ...

```

Kodeliste 8.1: Oppdater header-'felt' ved vellykket API-forespørsel.

```
    ...
    void WindNode::_checkExpiryDate()
    {
       if (_lastModified.addSecs(3600) <= QDateTime::currentDateTimeUtc() ) {
           QJsonObject windData = _downloadWind();
           _readWeatherFromJson(windData);
           _interpolateWeatherData();
       }
    }
    ...

```

Kodeliste 8.2: Sjekk om vinddata er utdatert.

"UA" spesifiseres i Nodeklassens medlemsfunksjon _downloadWind(). Siden det varierer hvem som planlegger ruten, må brukeren selv identifisere seg ved å skrive sin epostadresse i brukergrensesnittet. For å få til dette må en "UA"-fact opprettes i Appsettings slik at frontend og backend kommuniserer og oppdaterer UA under programkjøring.

```
QJsonObject WindNode::_downloadWind(){
...
    _userAgentApplication = "QGroundControlApp ";
    _userAgentMail = qgcApp()->toolbox()->settingsManager()->appSettings()->userAgent()->
     cookedValue().toByteArray();
    _request.setRawHeader("User-Agent", _userAgentApplication+_userAgentMail);
    //"QGroundControlApp student@stud.ntnu.no"
...
}
```

Kodeliste 8.3: Implementering av User-agent i Node-objekt

Vinddata har oppløsning i timesintervaller opp til 48 timer fram i tid, men brukeren kan maksimalt planlegge 36 timer frem. Da vind er et estimat, vil værdata lengst i fremtiden ha størst usikkerhet. Dermed er kvaliteten på estimert flytid negativt korrelert med

økende usikkerhet i vindata.

Noden behandler datainnholdet og interpolerer vinddata mellom timene. Dette gjør den med en gang data blir lastet ned, slik at behandlet data ligger lett tilgjengelig i programminnet.

Node Manager

Nodemanager administrerer nodene. Når en node til et koordinat blir etterspurt vil nodemanager sjekke om den allerede er lagt til i en hashmap. Om den ikke finnes her blir det opprettet en ny node. Adressen til den nye noden blir lagt til i hashmap og returnert. Ved å bruke adressene til nodene istedenfor noden selv spares det minne da objektet ikke trenger å bli kopiert. Node manager vil også passe på at det ikke sendes flere forespørsler til APIet enn det som er tillat.

Tile

Tile-objektet inneholder adressen til de fire nodene som utgjør hjørnene. Ved forespørsel, vil tile-en hente vinddata til et gitt tidspunkt fra nodene. Deretter blir disse interpolert for å få vinden til et punkt som kan ligge et vilkårlig sted innen arealet lukket av nodene. På denne måten er det mulig å anta hvordan vinden er i et koordinat mellom nodene.

Tile Manager

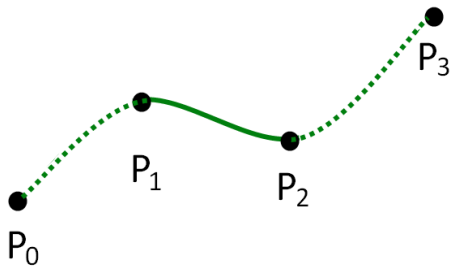
Tile manager administrerer tile-ene, og gjør mye av det samme som node manager. Om en tile ikke finnes fra før blir denne opprette og adressen lagt inn i en hashmap. For å indeksere tilene brukes koordinaten til det sørvestre hjørnet, da denne er unik for alle tiles. Tile manager administrerer også forespørsler etter vinddata sendt fra Weather Manager. Forespørselen inneholder en koordinat og en tid, som brukes til å hente riktig tile, og deretter gi forespørselen videre til tile-en. Vindvektoren Tile Manager får tilbake fra tile-en sendes direkte til Weather Manager.

Weather Manager

Weather manager er en overordnet manager for vær. Den tar imot forespørsler fra resten av programmet, og dirigerer den til riktig tjeneste. Den kan se overflødig ut nå, men er der for å legge til rette for å utvide til flere værtjenester. I tillegg er det weather manager som oppretter instansene av node- og tile manager. Da trenger ikke hovedprogrammet ta hensyn til at finnes flere managere. Figur 8.14 viser hvordan disse komponentene er knyttet sammen.

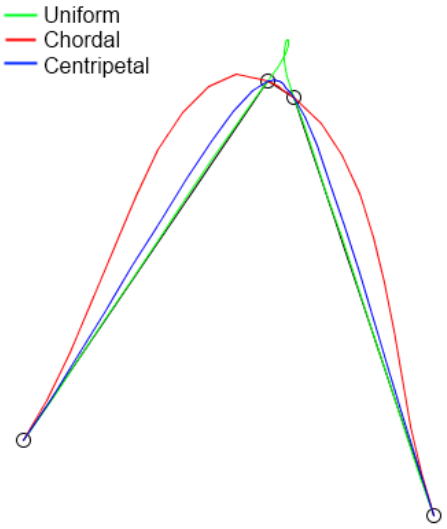
8.3.3 Tidsinterpolering av vinddata

Vinddata kommer med en oppløsning på ett datapunkt per time. Interpolering mellom disse må være glatt, da vindstyrken kan endres raskt på kort tid. Interpolering av vinddata gjennom tid oppnås ved bruk av en glatt interpoleringsmetode, av typen Catmull-Rom Twigg, 2003. Den er oppkalt etter de to informatikerene som opprinnelig utarbeidet den, Edwin Catmull og Raphael Rom. Den er et spesialtilfelle av en kardinal splinen.



Figur 8.16: Catmull-Rom gir kun polynom mellom to av punktene, men tar inn fire.

Catmull-Rom har noen egenskaper som gjør den godt egnet til dette prosjektet, deriblant at den garantert beveger seg gjennom alle kontrollpunkter underveis. Splinen har C_1 parametrisk kontinuitet. Splinen gir mulighetene til å velge hvor skarp interpoleringen skal være med parameteren τ. Andre splines deler de to første egenskapene, men det er spesielt den siste egenskapen som er ønskelig. Parameteren τ velges basert på hvor skarpt man ønsker interpoleringen, i dette prosjektet ble den valgt til $0,5$. En $\tau = 0,5$ er et bra mellom punkt som gjør at overgangen ikke tar store omveier for å gjøre banen mykest mulig, og ikke for skarpt slik at man kan få knekk eller at polynomet unødig krysser seg selv. Eksempler på disse ekstremtilfellene kan ses i figur 8.17.



Figur 8.17: Catmull-Rom gir mulighet til å velge hvor skarp interpoleringen skal være. Uniform er $\tau = 0$, centripetal er $\tau = 0,5$, og chordal er $\tau = 1$

$$c_0 = P_i \tag{8.20}$$

$$c_1 = (-\tau)P_{i-1} + (\tau)P_i + 1 \tag{8.21}$$

$$c_2 = (2\tau)P_{i-1} + (\tau - 3)P_i + (3 - 2\tau)P_{i+1} + (-\tau)P_{i+2} \tag{8.22}$$

$$c_3 = (-\tau)P_{i-1} + (2 - \tau)P_i + (\tau - 2)P_{i+1} + (\tau)P_{i+2} \tag{8.23}$$

Figur 8.18: Tar inn et sett med fire punkter og regner ut koeffisientene til det tilhørende kubiske polynomet for interpolering mellom P_i og P_{i+1}.

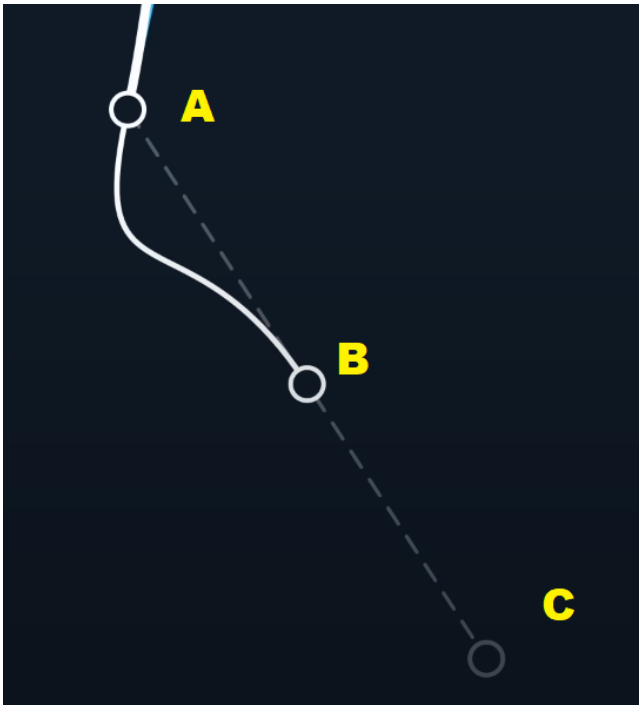
$$P(t) = c_3 t^3 + c_2 t^2 + c_1 t + c_0 \tag{8.24}$$

Figur 8.19: Kubisk polynom for å regne ut punkt verdi for input $0 \leq t \leq 1$ der t er tidsindeksen mellom to punkter P_i og P_{i+1}.

Et problem som måtte løses med denne teknikken er at dersom man ønsker å interpolere mellom de første punktene i datasettet P_0 og P_1 så prøver algoritmen også å hente ut P_{-1} i formelen. Et tilsvarende problem oppstår ved siste og nest siste indeks i listen. Det betyr at når listen har lengde n og indeksen er $i = n-1$ prøver formelen å hente P_{n+1}, og tilsvarende når indeksen er $i = n$ prøver formelen å hente ut P_{n+1} og P_{n+2}. Dette ble løst ved å speile punkter for å kompansere for manglende data. Eksempelvis ble det første unntakstilfellet løst ved å speile P_1 rundt P_0 for å representere P_{-1}.

$$P_{i-1} = 2P_i - P_{i+1} \tag{8.25}$$

Figur 8.20: Formel for speiling av punkt.



Figur 8.21: Punkt A er her speilet rund punkt B for å lage punkt C. Hentet fra (Freya Holmér, 2022)

8.3.4 Bilineær interpolering av vindvektorer

Vinddata blir lagret i et rutenett, og det er derfor nødvendig å ha en måte å interpolere punktene dersom man ønsker en vindvektor for steder som ikke lander akkurat på kryss-punktene i rutenettet. Teknikken som ble valgt for dette var bilineær interpolering fordi metoden er basert på å interpolere et punkt i et kvadrat der man vet verdien i hjørnene. Ved da å velge hjørnene til interpoleringskvadratet lik kryspunktene i rutenettet og å normalisere koordinatene til punktet man ønsker å interpolere til verdier mellom 0 og 1 kan man bruke formelen 8.26 for å finne vindvektoren for punktet.

$$P(x,y) = P_{SV}(1-x)(1-y) + P_{S\emptyset}x(1-y) + P_{NV}(1-x)y + P_{N\emptyset}xy \tag{8.26}$$

Figur 8.22: Formel for bilineær interpolering av vindvektor mellom punkter i vinddata rutentett.

P_{SV} : Punkt i sydvestlige hjørne.
$P_{SØ}$: Punkt i sydøstlige hjørne.
P_{NV} : Punkt i nordvestlige hjørne.
$P_{NØ}$: Punkt i nordøstlige hjørne.
x : Breddegradene til punktet normalisert til en verdi mellom 0 og 1.
y : Lengdegradene til punktet normalisert til en verdi mellom 0 og 1.

8.3.5 Høydejustering av vindvektorer

Siden vinddata fra Yr er for 10m over bakken, er det ønskelig å regne ut et estimat for vindvektoren ved høyden dronen opererer i. For å gjøre dette ble det brukt Hellmaneksponentloven 8.9. Denne formelen trenger fire parametre for å estimere vinden ved en høyde;

- En høyde som ikke for langt over bakken
- Vinden ved høyden nærme bakken
- Høyden man ønsker å estimere vinden ved
- Hellmaneksponenten for terrenget i det punktet

En kjent vindvektor og høyden dens over bakken er hentet fra Yr eller interpolering av vinddata fra Yr, og er derfor mulig å hente ut for et ønsket punkt. Det er også nødvendig å vite dronens høyde over terrenget til punktet man ønsker vindvektoren for. Den siste parameteren som mangler er Hellmaneksponenten. En tabell for dette er gitt i 8.1. Terrengtypen for et punkt i rutenettet hentes fra Statkart når et Node-objekt opprettes. Disse terrengtypene tilsvarte ikke direkte med de terrengtypene som var gitt i den eksisterende tabellen. Det var derfor nødvendig å lage en ny tabell der det kunne hentes ut Hellmaneksponenter basert på terrengtypen fra Statkart sitt API gitt i tabell 8.2. Disse verdiene ble valgt basert på intuisjon om terrengtypen og hvordan disse samsvarer med terreng typene fra den opprinnelige tabellen 8.1. Disse variabelene interpoleres med bilineær interpolering til et ønsket punkt i en Tile.

Terrengtype	**API terreng verdi**	**Hellmaneksponent α**
Åpent område	ÅpentOmråde	0.10
Europaveg	Europaveg	0.10
Industriområde	Industriområde	0.30
Steintipp	Steintipp	0.10
Havflate	Havflate	0.10
Idrettsplass	SportIdrettPlass	0.15
Park	Park	0.20
Elv, bekk	ElvBekk	0.20
Riksveg	Riksveg	0.10
Snø, isbre	SnøIsbre	0.15
Regulert innsjø	InnsjøRegulert	0.10
Golfbane	Golfbane	0.15
Skog	Skog	0.25
Fylkesveg	Fylkesveg	0.10
Ferskvann tørrfall	FerskvannTørrfall	0.15
Alpinbakke	Alpinbakke	0.20
Rullebane	Rullebane	0.10
Innsjø	Innsjø	0.10
Gravplass	Gravplass	0.20
Lufthavn	Lufthavn	0.10
Myr	Myr	0.15
Dyrket mark	DyrketMark	0.20
Tettbebyggelse	Tettbebyggelse	0.30
Bymessig bebyggelse	BymessigBebyggelse	0.40
Privat veg	Privat veg	0.10
Kommunal veg	Kommunal veg	0.10
Steinbrudd	Steinbrudd	0.20

Tabell 8.2: Verdier for Hellmaneksponenten gitt terrengtype.

8.3.6 Knapp og meny i brukergrensesnitt

Knappen plasseres i planmenyen til venstre i brukergrensesnittet. Elementer av denne menyen blir opprettet i planView.qml. Det finnes flere knapper fra før av, og dermed er det trivielt å opprette en ny som kan tilpasses etter våre ønsker. Kodeliste 8.4 hvordan knappen er implementert i brukergrensesnittet.

```
782 //CustomPlanView.qml
783 ToolStripAction {
784     text:       "Flight Time"
785     iconSource: "/qmlimages/FlightTime.svg"
786     enabled:    _planMasterController.controllerVehicle.fixedWing ||
     _planMasterController.controllerVehicle.vtol
787     visible:    true
788     dropPanelComponent: flightTimeDropdown
789 }
```

Kodeliste 8.4: 'Flight Time' knapp i planmeny.

Knappen er kun synlig og mulig å bruke dersom brukeren planlegger en rute for en VTOL- eller en fixed wing drone. Dette skyldes at FALK er en fixed wing VTOL-drone. Når knappen, flight time påtrykkes, dukker det opp en popup meny hvor brukeren er nødt til å spesifisere dato, tidspunkt, estimation stepsize, og UA epostadresse. Videre vil kodeeksempler illustere hvordan popupmeny er implementert ved å bruke UA som eksempel.

Kodeliste 8.5 viser hvordan innstastningsfeltet blir navngitt og lagt til i menyen.

```
1390 //CustomPlanView.qml
1391 Component {
1392     id: flightTimeDropdown
1393         ColumnLayout {
1394             id:         columnHolder
1395             spacing:    _margin
1396             GridLayout {
1397                 columns:            1
1398                 columnSpacing:      _margin
1399                 rowSpacing:         _margin
1400                 Layout.fillWidth:   true
1401                 visible:            true
1402                     QGCLabel{
1403                         text:           qsTr("User-Agent: Email")
1404                         font.pointSize: ScreenTools.smallFontPointSize
1405                     }
1406                     FactTextField {
1407                         fact:  QGroundControl.settingsManager.appSettings.userAgent
1408                         Layout.fillWidth:   true
1409                     }
1410             }
1411         }
1412     }
```

Kodeliste 8.5: UA-input for flytidestimering

I feltet FactTextField ligger facten som kobler variabler i backend mot funksjonalitet i frontend. Disse opprettes i AppSettings.cc/h som i kodeliste 8.6.

```
// AppSettings.cc
DECLARE_SETTINGSFACT(AppSettings, userAgent)

// AppSettings.h
DEFINE_SETTINGFACT(userAgent)
```

Kodeliste 8.6: Deklarasjon av UA fact i AppSettings.cc/h

Fact må spesifiseres som JSON-objekt dersom en ønsker å aksessere dem i- og manipulere dem fra brukergrensesnittet. Objektet innholder informasjon om facten. Det forteller hvilken elementet som tilhører fact, hvilken datatype elemenet er, defaultverdier, og mer. For denne undermenyen opprettes objektet i App.SettingsGroup.json som i kodeliste 8.7.

```
{
//App.SettingsGroup.json
    "name":             "userAgent",
    "shortDesc":        "Valid user-Agent ex.: johnDoe@gmail.com",
    "longDesc":         "Terms of Service from the weather API requires a User-Agent mail
     .",
    "type":             "string",
    "default":          ""
}
```

Kodeliste 8.7: JSON UA fra meny for flytidestimering

Når brukeren har spesifisert alle nødvendige elementer i menyen gjenstår det kun å sette i gang beregningene. Dette gjøres ved å legge til en 'Run'-knapp nederst i menyen. Ved klikk kjøres funksjonene som setter i gang beregningene. Kodeliste 8.8 viser syntaks for knappen og alle funksjonskall den gjør. Når beregningene er ferdig åpner et vindu som viser den estimerte flytiden.

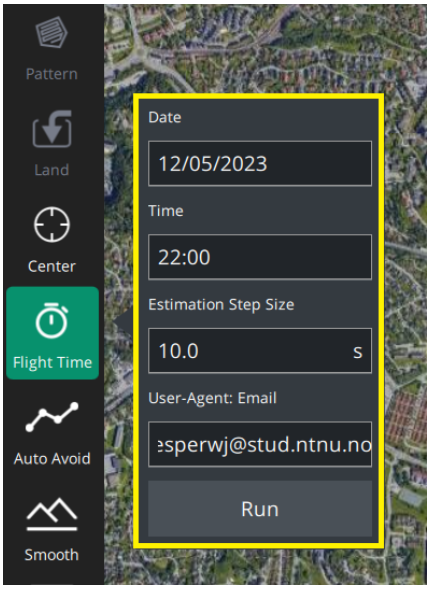
```
1436 //CustomPlanView.qml
1437 QGCButton {
1438         text:               qsTr("Run")
1439         Layout.fillWidth:   true
1440         onClicked: {
1441             //console.log(_appSettings.currentTime.value)
1442             _planMasterController.presentTime()
1443             _planMasterController.presentDate()
1444 
1445             var inputTime = _appSettings.currentTime.value
1446             var inputDate = _appSettings.currentDate.value
1447 
1448             _appSettings.inputDateTime.value
1449             _planMasterController.checkValidDateTime(inputDate,inputTime)
1450             console.log(_appSettings.inputDateTime.value)
1451             dropPanel.hide()
1452 
1453             var eftStr = _planMasterController.flightTime(_appSettings.inputDateTime.
    value)
1454             mainWindow.showMessageDialog(qsTr("Flight time estimate at time: "+
    _appSettings.inputDateTime.value), qsTr(eftStr), StandardButton.Ok, function() {})
1455         }
1456 }
```

Kodeliste 8.8: Run-knapp for endelig flytidestimering

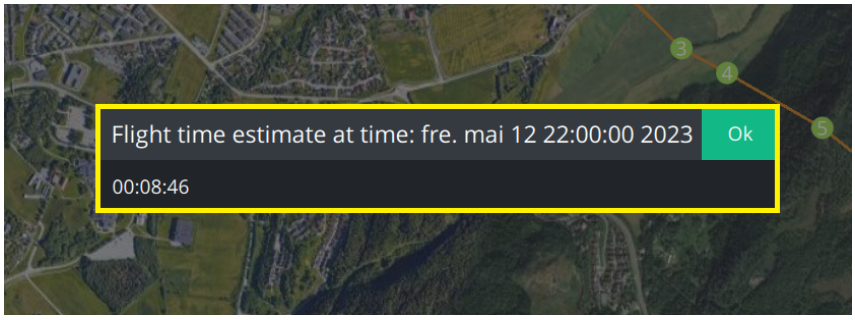
8.4 Resultater og empiriske funn

8.4.1 Estimering av flytid

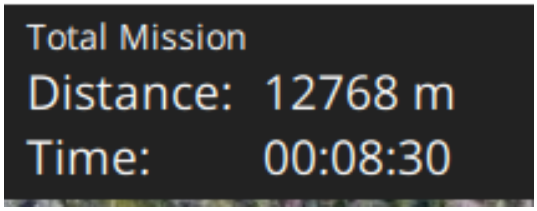
De nye funksjonaliteten for estimering av tidsforbruket til en rute er mer realistisk enn QGCs egne flytidsberegning, da QGC hverken tar med stigning eller vind med i betraktning. Ved å trykke på knappen ”Flight Time”, vist i figur 8.23, dukker det opp en ny meny der man har mulighet til å taste inn et ønsket avreise tidspunkt innen de neste 36 timene. Det er også nødvendig å taste inn en epostadresse da dette er et krav fra YR (Yr, 2023). Det er nødvendig å definere tidsintervallet for oppløsningen til simuleringen av flyruten. Ved å så trykke Run, bruker programmet noe tid til å hente ut nødvendig data og regne ut resultatet. Det er verdt å merke at mye av denne tiden går til innhenting av data, men denne tiden kuttes ned betraktelig siden denne data lagres lokalt så lenge programmet er åpent når data først har blitt lastet ned.



Figur 8.23: Meny for valg av avreisetidspunkt, simulering-soppløsning, og email til API krav.



Figur 8.24: Resultatvindu for simulering av flyrute.



Figur 8.25: QGCs estimat for flytid, og total bakkedistanse.

8.4.2 Endringslogg

Tabell 8.3 viser alle kodeendringer som er gjort under utviklingen av modul 3.

Filsti	Filnavn	Tillegg
custom\src\MissionManager\	PlanMasterController.h	Deklarere medlemmer: presentTime, presentDate, checkValidDateTime, checkValidUserAgent, getWind, getCruiseSpeed, flightTime
custom\src\MissionManager\	PlanMasterController.cc	Definere medlemmer: presentTime, presentDate, checkValidDateTime, checkValidUserAgent, getWind, getCruiseSpeed, flightTime
src\WindAPI\	Node.h	Deklarere klasse: WindNode
src\WindAPI\	Node.cpp	Definere klasse: WindNode
src/WindAPI\	NodeManager.h	Deklarere klasse: NodeManager
src\WindAPI\	NodeManager.cpp	Definere klasse: NodeManager
src\WindAPI\	Tile.h	Deklarere klasse: Tile
src\WindAPI\	Tile.cpp	Definere klasse: Tile
src\WindAPI\	TileManager.h	Deklarere klasse: TileManager
src\WindAPI\	TileManager.cpp	Definere klasse: TileManager
src\WindAPI\	WeatherManager.h	Deklarere klasse: WeatherManager
src\WindAPI\	WeatherManager.cpp	Definere klasse: WeatherManager
custom\res\	CustomPlanview.qml	Legg til knapper: Flight Time
src\Settings\	AppSettings.h	Definere settingsfact: currentDate, currentTime, inputDateTime, userAgent, flightTimeEstimationStepSize
src\Settings\	AppSettings.cc	Deklarere settingsfact: currentDate, currentTime, inputDateTime, userAgent, flightTimeEstimationStepSize
src\Settings\	App.SettingsGroup.json	Definere FactMetaData: currentDate, currentTime, inputDateTime, userAgent, flightTimeEstimationStepSize

Tabell 8.3: Endringslogg for modul 3

8.5 Analyse og diskusjon

8.5.1 Kartesiske koordinater til beregning

Vinddata fra Yr kommer som en vektor på polar form, men da det ble ansett som mer hensiktsmessig å bruke kartesiske koordinater i deler av programmet konverteres vektorene til kartesisk form. Den største grunnen til at dette valget ble tatt er interpolering av vindvektorene må gjøres komponentvis, som er utfordrende med vinkelen i vektorer på polar form. Vinkelen til vektoren var vanskelig å interpolere fordi å representere kontinuiteten til vinkler ville trengt mye ekstra logikk og beregninger. Eksempelvis ville to vektorer med vinkel 357° og 3° hatt en differanse på 354° i beregningene heller enn den korrekte differansen på 6°. Det ble derfor valgt at når de polare vektorene leses inn i programmet, konverteres de til kartesiske koordinater slik at interpolering kan gjøres enklere. Når andre deler av programmet ber om hva vindvektoren er for en spesifikk lokasjon gis det i polare koordinater. Konverteringen tilbake til polare koordinater gjøres før høydejusteringen av vinddata, da det kun er lengdekomponentet av polar vektoren som skaleres med høyden og ikke retningen. Å måtte konvertere mellom disse koordinatsystemene hver gang en vektor skal leses inn fra YR og hver gang en vindvektor skal brukes til simulering av rute introduserer en del ekstra beregninger, men dette veies opp for i forenklingen av de mer komplekse interpoleringsfunksjonene. Grunnen til at når andre deler av programmet ber om vindvektorer så gis det på polarform er at det henger igjen fra når program modulen ble planlagt. Det ble tidlig i planleggingen tenkt at fordi vindvektorene fra Yr kommer på polarform og mange av beregningene som skulle gjøres i kulekoordinater var det praktisk å ha det på denne formen. Senere i utviklingsprosessen ble interne beregninger i begge modulene endret til å være på kartesisk form siden dette var enklere å bruke. Programmet kan dermed effektiviseres en del ved å fjerne noen av disse beregningene assosiert med konverteringene, men det kommer til å kreve å endre noe av arkitekturen til intern kommunikasjon i programmet.

8.5.2 Valg av spline

Splines ble valgt istedenfor lineær interpolering fordi det er bedre egnet til datasett med store variasjoner. Vind varierer stort sett lite fra sted til sted, men kan endres drastisk over få timer. Videre ble den spesifikke typen Catmull-Rom spline valgt fordi den opp-

fylte alle kravene som var satt til som det var nødvendig at algoritmen måtte oppfylle. Splines løste dermed problemet på en god måte, da den var mer nøyaktig enn lineær interpolering med en minimal økning i beregningskostnaden siden koeffisientene til polynomene regnes ut når data leses inn heller enn hver gang det hentes ut en vindvektor for en gitt tidsindeks.

8.5.3 Nøyaktighet av Hellmaneksponent

Yr disponerer vindata kun ved 10m. Droner flyr ofte langt høyere over bakken og derfor må data fra Yr høydejusteres. Høydejusterte vindvektorer påvirkes av usikkerhet i beregning og datakilder. Dette slår ut på nøyaktigheten av vindvektorestimatet. Derfor gjennomføres en analyse som bruker kjente datakilder som målestokk for å verifisere om estimatet er presist eller ikke.

I prosjektet er Hellmaneksponenten anslått ved korrelasjonen mellom terrengtypene fra Statkart og tabell 8.1, hvor tabellen i seg selv allerede er et estimat(Francisco mfl., 2011).

Data fra den tyske værtjenesten, Deutschen Wetterdienstes (DWD), med deres modell Icosahedral Nonhydrostatic Model (ICON) brukes som målestokk, siden den blant annet tilbyr vinddata ved 10m og 80m. Analysen består av data fra fire punkter i Trondheimsområdet med kjent terrengtype. Ved å bruke data fra de fire punktene og løse for Hellmaneksponenten fra Hellmaneksponent-loven, fra Figur 8.26, er det mulig å estimere en anbefalt eksponent for hver time, i hver lokasjon.

$$\alpha_{anbefalt} = \frac{\ln(H) - \ln(H_0)}{\ln(V) - \ln(V_0)} \tag{8.27}$$

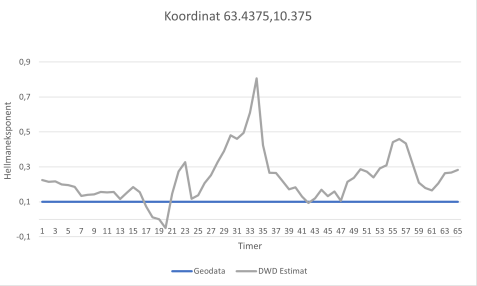
Figur 8.26: Formel som bruker vinddata fra to kjente høyder for å estimere hva Hellmaneksponeten burde være.

Terrengtypen brukes til sammenligne den anbefalte eksponenten med tilsvarende eksponent fra tabell 8.2. På grafene i figur 8.27 viser analysen stor variasjon mellom anbefalt eksponent og eksponenten som faktisk blir brukt for høydejustert vindestimering. Dersom man ser på resultatene for analysen i tabell 8.4 så stemmer median verdiene og

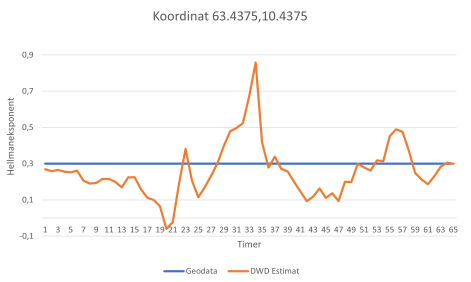
gjennomsnittene overens i enkelte tilfeller, men standardavviket mellom eksponentene er stort og indikerer at vindestimatet er upresist. Siden Hellmaneksponetens ekstremverdier i tabell 8.1 er 0, 10 og 0, 40, er et standardavvik på 0, 15 for stort. Dette betyr at Hellmaneksponent-loven med tilgjengelige eksponenter ikke produserer gode estimat av høydejustert vind ved korte tidsintervall.

Punkt	Statkart konstant	Gjennomsnitt	Median	Standardavvik
Nordvest	0,1	0,24	0,21	0,14
Nordøst	0,3	0,26	0,23	0,15
Sydvest	0,25	0,32	0,28	0,16
Sydøst	0,2	0,31	0,27	0,16

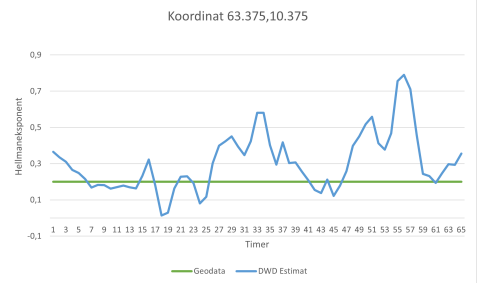
Tabell 8.4



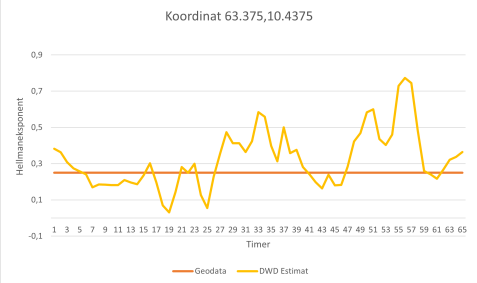
(a) Nordvestlig punkt, med desimal koordinater N63.4375 Ø10.375.



(b) Nordøstlig punkt, med desimal koordinater N63.4375 Ø10.4375.



(c) Sydvestlig punkt, med desimal koordinater N63.375 Ø10.375.



(d) Sydøstlig punkt, med desimal koordinater N63.375 Ø10.4375.

Figur 8.27: Grafer som sammenligner et estimat av Hellmaneksponenten fra DWDs værdata, med verdien som hentes fra oppslags tabellen. For å se nærmere på data og analyse se vedlegg D

8.5.4 Nedlastingsprosess for værdata

Den største grunnen til at simuleringen av ruten ofte tar lang tid er at programmet må laste ned værdata for hvert beregningstidspunkt sekvensielt. Årsaken til at dette tar så lang tid er todelt. Den første er at verken funksjonalitet for å sende forespørsler i parallell eller funksjonalitet for at programmet kunne fortsette å jobbe med andre ting mens det venter på svar fra en forespørsel ble implementert. Den andre grunnen er at Locationforecast ikke tillater å be om mer enn ett datapunkt per forespørsler som sendes. Begge disse har stor påvirkning, men den første er den som enklest kan gjøres noe med. Dersom disse endringene hadde blitt implementert hadde det fjernet den største flaskehalsen til å få flytidsestimeringen til å være betydelig raskere.

8.6 Kapittel konklusjon

Den nye funksjonen for tidsestimering regner ut estimert tidsforbruk til ruten på en beregningseffektiv måte tiltross for store tidsforsinkelser i nedlasting av værdata. Bilineær og Catmull-Rom spline ser ut til å interpolere vinddata på en god måte, men høydejustering med Hellmaneksponent-loven er muligens for upresis til å brukes til korte tidsperioder. Løsningen for nedlasting av værdata kunne løses ved å parallellisere nedlastingsprosessen.

Kapittel 9

Testing

9.1 Introduksjon

Testing er en viktig del av prosjektet. Testing gjennomføres for å verifisere at funksjonalitet fungerer som planlagt. Waterfall SDLC legger opp til tre ulike test-typer: α-,β- og Acceptance testing (W3schools, u.å.).

9.2 Teoretisk rammeverk

9.2.1 α test

α testing er innledende validering om ny funksjon fungerer som forventet. Dette gjennomføres tidlig i- og kontinuerlig gjennom prosessen av utviklerne. Testene er små, og har som hensikt å gjøre debugging enklere. α-testing er etterfulgt av β-testing.(Semilof, 2021)

9.2.2 β test

β-testing innebærer å eksponere produktet for brukergruppen (W3schools, u.å.). Utviklere utarbeider et testskjema som skal veilede brukerne gjennom programmet. Brukere som ikke har innsyn i utviklingsarbeidet oppdager ofte problemer som ikke er tydelige for utviklerne. Dette bidrar til å ytterligere forbedre programvaren.

9.2.3 Acceptance test

Når brukergruppen ikke har flere tilbakemeldinger er det klart for en avsluttende test. Oppdragsgiver veiledes gjennom programvaren, og beslutter om resultatet samsvarer med forventninger og om det er klart for ut utrulling (W3schools, u.å.).

9.3 Metode og utstyr

9.3.1 Testgjennomføring

I programvareutvikling jobber gruppemedlemmer ofte individuelt med sine bidrag. Dermed gjennomføres også α testing kontinuerlig og individuelt. I praksis betyr det at når kode er testklar, kjører utvikleren programmet og sjekker om funksjonen fungerer som den skal.

På grunn av rollen til MR som både oppdragiver og brukergruppe, legges β-tester under Acceptance-test. Slik lages for hver modul og har som hensikt at oppdragsgiver gir klarsignal til oppstart på neste modul. I starten av hver test introduseres funksjon og dens hensikt. Deretter begynner testskjemaet, som er en tabell med tre kolonner. Første kolonne inneholder instruksjon som bruker skal gjennomføre. Neste kolonne beskriver forventet resultat av instruksjon. Siste kolonne inneholder bilde av instruksjon eller resultat. Bilde er betryggende for den som utfører testen da det kan verifisere om instruksen er riktig gjennomført.

9.3.2 Utstyr

MS Word bukes for å lage Acceptance-test.

9.4 Resultater og empiriske funn

Interne α-tester ga hyppig diagnostikk rundt progresjon eller eventuelle feil under utvikling. Testingen for til god arbeidstuktur som videre kvalitetsikrer programvaren som presenteres for oppdragsgiver.

Test 1 ble gjennomført digitalt, og både veileder og oppdragsgiver fikk testen tydelig presentert over opptak. Tilbakemeldinger ble sendt over mail innen én virkedag. Se vedlegg B for kopi av korrespondanse.

Test 2 ble gjennomført fysisk i MRs lokaler. Denne gang var modul 2 og 3 i fokus, men endringer fra tilbakemeldingen gjort i modul 1 ble også presentert. Oppdragsgiver følgte testskjema, og kom med positive tilbakemeldinger og godkjente programvaren. Se vedlegg C for testskjema.

9.5 Analyse og diskusjon

Interne α-tester tillot gruppen å raskt evaluere arbeidets kvalitet. En uformell tilnærming til dette bidro til hurtig utvikling, dermed rask fremgang med arbeidspakkene. Samtidig, gjorde manglene på formalitet at det oppsto feil senere i utvikling. Interne tester foregikk fortløpende uten standardisert gjennomføring og dokumentasjon, og det er uvisst hvordan dette påvirket arbeidet. I henseende er det ingen tydelige negative konsekvenser med gjennomføringen. På en annen side ville nøye planlagte interne tester krevd mer tid av utviklingsfasen. Likevel, ville det la arbeidet skje i fast rekkefølge, uten behov for å rette opp i feil fra tidligere arbeidspakker.

Gjennom prosjektarbeidet ble det ikke direkte gjennomført β-tester. Fordelen med slike tester kan blant annet være å kunne tilpasse brukergrensesnitt til folk utenfor prosjektgruppen. Perspektiver utenfra kan være mer nyanserte og representere en generell be-

folkning bedre enn utvalgte gruppemedlemmer. Samtidig, ved designvalg, vil det være tekniske aspekter til stede som gjør gruppemedlemmer partiske, da ytterligere endringer til grensesnittet kan bety mer arbeid for dem.

Under prosjektet ble det gjennomført to acceptance-tester. Disse spilte sentrale roller i utvikling-tilbakemelding-syklusen under prosjektarbeidet. Testene ble holdt med én måneds mellomrom. Dette ga gruppen kort tid til å utvikle ny funksjonalitet, men hadde stor fremgangen og skape god funksjonalitet som oppdragsgiver kunne teste. Både gruppens og oppdragsgivers tid ble godt utnyttet. Samtidig, ble det tatt mange tekniske avgjørelser i det tidsrommet, og med lite tid til utvikling etter testen, hadde ikke gruppen tatt hensyn til mengden tilbakemelding oppdragsgiver kunne komme med. Gruppen risikerte å måtte justere på store deler av arbeidet, siden det hadde skjedd såpass mye i tiden mellom testene. Takket være god kommunikasjon med oppdragsgiver før og under denne perioden, var designkravene tydelige, og gruppen kunne drive med selvstendig utvikling uten behov for store endringer i etterkant av den siste testen.

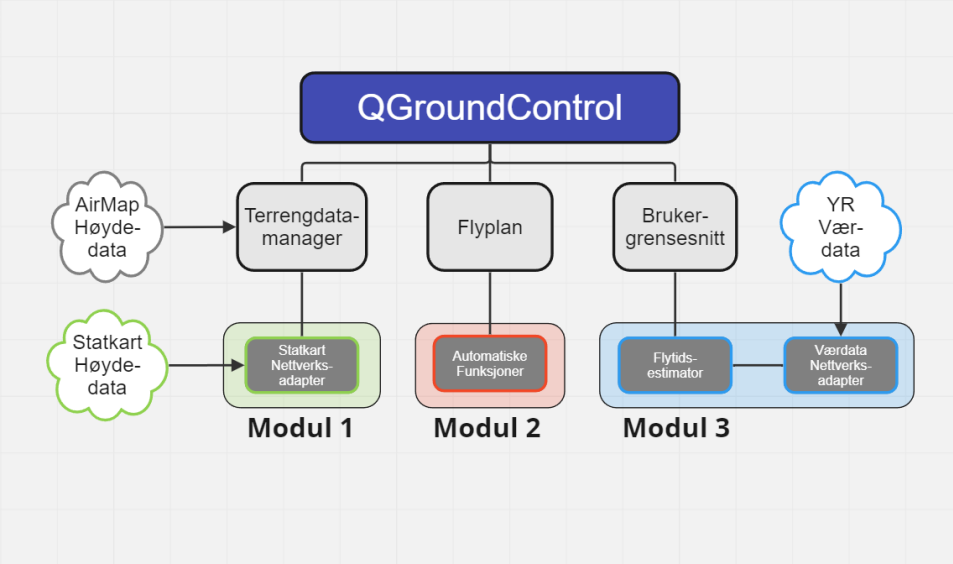
9.6 Kapittel konklusjon

Det ble gjennomført *alpha*-tester underveis i prosjektarbeidet samt to acceptance-tester. Testing internt foregikk hyppig og uformelt, noe som la grunnlag for rask fremgang. Videre, ble det holdt to formelle tester for oppdragsgiver som begge gikk etter planen. Tilbakemelding ble gitt tydelig og oversiktlig, noe som lot gruppen effektivt adressere ønskede forbedringspotensialer. Testingen har vært med på å holde arbeidet tett tilknyttet oppdragsgivers behov, samt å verifisere kvaliteten ved arbeidet underveis.

Kapittel 10

Resultat

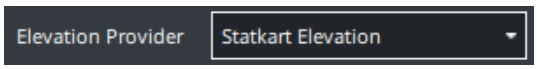
Alle modulenes resultat sammenstilles i dette kapittelet. Hensikten er å presentere nye funksjonaliteter som brukeren har tilgjengelig i denne versjonen av QGC. Figur 10.1 viser hvordan de forskjellige delene av programutvidelsen henger sammen.



Figur 10.1: Figuren viser hvordan de tre programmodulene kobler seg på forskjellige deler av QGC.

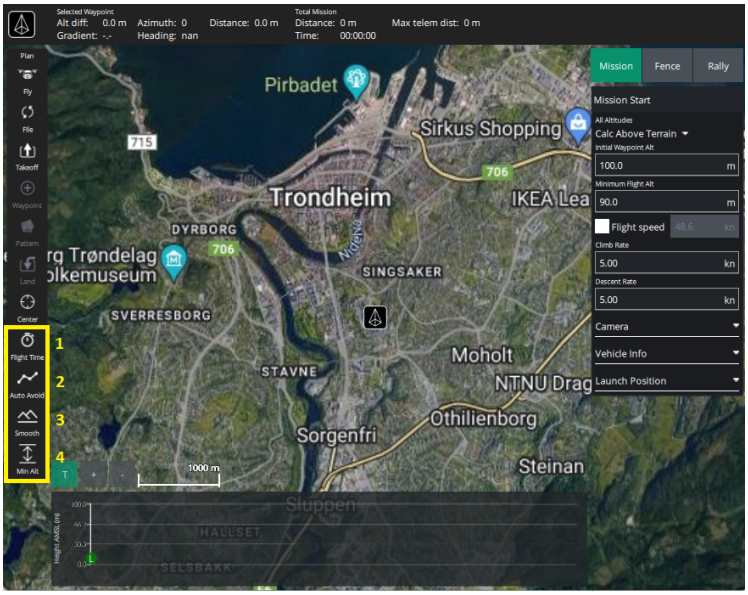
Terrengfølgende ruteplanlegging er nå mulig å gjennomføre i hele Norge. Brukeren har

valget om å bruke Airmap eller Kartverket som distributør av terrenghøyde. Figur 10.2 viser en menyen som brukeren kan bestemme distributør av terrenghøyde. Menyen ligger i innstillinger i QGC.



Figur 10.2: Valgmeny; distributør av terrenghøyde

I tillegg har brukeren fire nye knapper som på hver sin måte assisterer terrengfølgende ruteplanlegging. Figur 10.3 viser alle nye knapper markert i gult.



Figur 10.3: Fire nye knapper som effektiviserer ruteplanlegging på hver sin måte.

I figur 10.3 vil knapp:

1 estimerer flytid med hensyn til vind.
2 legger til nye waypoints dersom planlagt rute kolliderer med terrenget.
3 justerer stigning til å være innenfor grenser bestemt av operatør.
4 bestemmer minimum høyde over terrenget for hele flyruten.

Kapittel 11

Konklusjon

Denne delen har som hensikt å konkludere om problemstillingen ble løst på en tilfredsstillende måte. Og komme med anbefalt videre arbeid.

Rapportens problemstilling lyder som følgende:

> *Kan norsk høydedata integreres i ruteplanlegging for MRs UAV for å oppnå optimal flyhøyde? Videre, kan utvikling av ekstra funksjonalitet effektivisere ruteplanlegging i QGroundControl, med fokus på å fly nært terrenget?*

Norsk høydedata er integrert i programmet slik at både funksjoner som eksisterte før og nye kan bruke data optimalt.

Funksjonaliteten bak nye knapper effektiviserer ruteplanleggingen, fordi det sparer brukeren for manuelt og repetitivt arbeid.

Derfor kan problemstillingen sies å være løst på en tilfredsstillende måte

11.1 Videre arbeid

Å laste ned data for lengre ruter tar lang tid. Dette er fordi det sendes 28 forespørsler på en gang for så å prosessere data til en tile før den laster ned neste. Her er det mye forbedringspotensiale med hensyn til tidseffektivitet

Slik det er nå er det tre knapper for å kjøre de tre optimaliseringsfunksjonene. Dette kan forbedres ved å samle alt i en knapp, gjerne med mulighet til å velge hvilke funksjoner som skal kjøres.

Nevnt i kapittel 7.5.4 kan oppdatering av UI bli bedre. Metoden som brukes nå er ikke optimal, og skulle gjerne vært forbedret.

Hvis man prøver å bruke estimeringen ved negative bredde- eller lengdegrader vil det oppstå problemer med avrunding av koordinater. Det er ikke tatt hensyn til dette slik funksjonene er nå.

Bibliografi

Airmap. (2023). *Airmap Elevation API*. https://developers.airmap.com/docs/elevation-api (Hentet: 26. april 2023)

Amazon. (u.å.). *What Is An API?* https://aws.amazon.com/what-is/api/ (Hentet: 27. april 2023)

Bratbergsengen, K., & Nätt, T. H. (2022). *Cache*. https://snl.no/cache (Hentet: 27. april 2023)

Francisco, B.-R., Ángeles-Camacho, C., & Rios-Marcuello, S. (2011). *Methodologies Used in the Extrapolation of Wind Speed Data at Different Heights and Its Impact in the Wind Energy Resource Assessment in a Region. 10.5772/20669.* https://www.researchgate.net/publication/221912731_Methodologies_Used_in_the_Extrapolation_of_Wind_Speed_Data_at_Different_Heights_and_Its_Impact_in_the_Wind_Energy_Resource_Assessment_in_a_Region (Hentet 21. mai 2023)

Freya Holmér. (2022). *The Continuity of Splines*. https://www.youtube.com/watch?v=jvPPXbo87ds&ab_channel=FreyaHolm%5C%C3%5C%A9r (Hentet 9. mai 2023)

Git. (u.å.). *About*. https://git-scm.com/about (Hentet 21. mai 2023)

GitHub. (u.å.). *GitHub*. https://github.com/ (Hentet 21. mai 2023)

Griffins-Gadgets. (2020). *Modifying QGroundControl #1 - Setup and First Build*. https://www.youtube.com/watch?v=6SGl5S6GpNk&t=308s (Hentet: 22. februar 2023)

Kartverket. (2023). *Åpent API for høyde- og dybdedata fra Kartverket*. https://ws.geonorge.no/hoydedata/v1/#/ (Hentet: 26. april 2023)

Luftfartstilsynet. (u.å.). *Sikkerhetsavstander og maksimal flygehøyde*. https://luftfartstilsynet.no/droner/kommersiell-bruk-av-drone/sikkerhetsavstander-og-maksimal-flygehoyde/ (Hentet 21. mai 2023)

Maritime Robotics. (2023). *MR Quality Policy*. https://www.maritimerobotics.com/quality-policy (Hentet 18. mai 2023)

Nygård, S. L., Skjolden, T. S., Jensen, J. W., & Klüver, J. H. (2023). Terrengfølgende ruteplanlegging for drone: Forprosjekt.

Ogniewski, Jens. (2019). *Spline Interpolation in Real-Time Applications using Three Control Points. 10.24132/CSRN.2019.2901.1.1.* https://www.researchgate.net/publication/336308276_Spline_Interpolation_in_Real-Time_Applications_using_Three_Control_Points (Hentet 21. mai 2023)

QGroundControl. (u.å.-a). *Getting Started*. https://dev.qgroundcontrol.com/master/en/getting_started/ (Hentet: 22. februar 2023)

QGroundControl. (u.å.-b). *Getting Started*. https://dev.qgroundcontrol.com/master/en/getting_started/ (Hentet 21. mai 2023)

QGroundControl. (u.å.-c). *Initial Repository Setup For Custom Build*. https://dev.qgroundcontrol.com/master/en/custom_build/CreateRepos.html (Hentet 21. mai 2023)

QGroundControl. (u.å.-d). *QGroundControl User Guide*. https://docs.qgroundcontrol.com/master/en/ (Hentet: 20. februar 2023)

Qt-Group. (u.å.-a). *A Cross-platform IDE for sofware development*. https://www.qt.io/product/development-tools (Hentet 11. mai 2023)

Qt-Group. (u.å.-b). *Qt Framework*. https://www.qt.io/product/framework (Hentet 18. mai 2023)

Qt-Group. (u.å.-c). *Signals and slots*. https://doc.qt.io/qt-6/signalsandslots.html (Hentet 11. mai 2023)

Semilof, M. (2021). *alpha testing*. https://www.techtarget.com/searchsoftwarequality/definition/alpha-testing (Hentet 13.mai 2023)

Twigg, C. (2003). *Catmull-Rom splines*. https://www.cs.cmu.edu/~fp/courses/graphics/asst5/catmullRom.pdf (Hentet 9. april 2023)

W3schools. (u.å.). *SDLC Waterfall method*. https://www.w3schools.in/sdlc/waterfall-model (Hentet 13.mai 2023)

Yr. (2023). *Terms of Service*. https://developer.yr.no/doc/TermsOfService/ (Hentet: 10. mai 2023)

Yr. (u.å.). *Locationforecast data model*. https://developer.yr.no/doc/locationforecast/datamodel/ (Hentet 21. mai 2023)

Zhenov, P. (2021). *Hash What? Understanding Hash Indexes*. https://codingsight.com/hash-index-understanding-hash-indexes/ (Hentet: 27. april 2023)

Vedlegg

Vedlegg A

Bacheloroppgave

NTNU

Institutt for elektroniske systemer
Institutt for elkraftteknikk
Institutt for teknisk kybernetikk

Oppgaveforslag bacheloroppgave elektroingeniør (BIELEKTRO) i Trondheim, vårsemester 2023

Navn bedrift: Maritime Robotics AS	**Kontaktperson:** Torbjørn Houge **Epost:** torbjorn.houge@maritimerobotics.com **Telefon/mobil:**	
Tittel på oppgave: Terrengfølgende ruteplanlegging for drone		
Hvilke studieretninger passer oppgaven for? (kryss av for alle aktuelle retninger; flervalg er mulig):	**Automatisering og robotikk**	☑
	Elektronikk og sensorsystemer	☐
	Elkraft og bærekraftig energi	☐
Er oppgaven reservert for noen bestemte studenter? (skriv navnene på studentene inn til høyre)		
Har dere arbeidsplass for studentene	☐ ja ☐ nei ☒ usikker?	
Er dette en lukket oppgave? Dvs. at sluttrapporten ikke kan publiseres fordi den inneholder sensitiv informasjon.	☐ ja ☐ nei ☒ ikke enda bestemt	

Kort beskrivelse av oppgaven med problemstilling.

Oppgaven omhandler ruteplanlegging for UAV.

Oppgavens innhold:

- Identifikasjon av mulig løsning (foreløpig tenkt løsning ved å modifisere QGroundControl og lage en "modul").

- Integrering av høydedata fra åpne kilder (i Norge: Statens kartverks løsning).

- Algoritme for å legge ruta for å følge terrenget, basert på høydemodell og flyets fysikk, samt konfigureringsmuligheter.

- Mulighet til å teste i simulator (X-Plane er tilgjengelig).

Ytterligere detaljer om denne oppgaven spesifiseres sammen med interesserte studenter i begynnelsen av prosjektet.

Vedlegg B

Mailkorrespondanse for test 1

Re: Fremvisning av hovedmodul 1

Torbjørn Houge <torbjorn.houge@maritimerobotics.com>
ti. 28.03.2023 13:41

Til: Tormod Skjolden <tormod.skjolden@ntnu.no>

Kopi: Sigurd Gosse <sigurd.gosse@gmail.com>;Sander Løvdahl Nygård <sanderln@stud.ntnu.no>;Jonas Hoel Klüver <jonas.h.kluver@ntnu.no>;Jesper Wille Jensen <jesperwj@stud.ntnu.no>

Takk for det

Jeg har sett på videoen, og det ser jo enkelt og greit ut. Automatikk kan jo forbedre ting, men sammenlignet med enkelte andre løsninger jeg har prøvd er det ikke ille.
Er import av høydemodell mer eller mindre hardkodet inn, eller bruker man QGC sitt grensesnitt?

Torbjørn

man. 27. mar. 2023 kl. 20:48 skrev Tormod Skjolden <tormod.skjolden@ntnu.no>:

> Supert!
>
> Her er testskjema for hovedmodul 1
>
> Tormod
>
> ---
>
> **Fra:** Torbjørn Houge <torbjorn.houge@maritimerobotics.com>
> **Sendt:** mandag 27. mars 2023 11:17
> **Til:** Tormod Skjolden <tormod.skjolden@ntnu.no>
> **Kopi:** Sigurd Gosse <sigurd.gosse@gmail.com>; Sander Løvdahl Nygård <sanderln@stud.ntnu.no>; Jonas Hoel Klüver <jonas.h.kluver@ntnu.no>; Jesper Wille Jensen <jesperwj@stud.ntnu.no>
> **Emne:** Re: Fremvisning av hovedmodul 1
>
> Hei
>
> Det blir spennende. Jeg har foreløpig tid 12-16 torsdag, men erfaringsmessig er siste uka før påske noe uoversiktlig så det mest praktiske for meg er nok en video. Da kan jeg og forsøke å få samlet et par stykker til her så vi kan være litt flere som gir tilbakemelding.
>
> hilsen Torbjørn
>
> man. 27. mar. 2023 kl. 11:02 skrev Tormod Skjolden <tormod.skjolden@ntnu.no>:
>
> > Hei Torbjørn,
> >
> > I fremdriftsplanen er det ført opp en test denne uken. Vi har god progresjon i utviklingen og vil gjerne ta deg gjennom en kort test som illustrerer hvor vi ligger i løpet.
> >
> > Testbeskrivelse i korte trekk:
> > 1. Generering av høydegraf fra norsk terrengdata
> > 2. Varsling ved kollisjon med bakken
> > 3. Varslingen går vekk ved å legge om ruten
> >
> > Testen kan fremvises over et teamsmøte, eller gjennom en forhåndsinnspilt video som du får tilsendt. Dersom du ønsker å ta det over teams, er vi tilgjengelige hele uken unntatt hele onsdag, og fredag etter 14:00. Videoen kan vi sende til deg i løpet av torsdag.

Kom gjerne med tilbakemelding om hvilket alternativ du finner mest gunstig. Uansett vil du få et testskjema slik at du kan følge testgjennomgangen.

Vennlig hilsen,
Bachelorgruppe 11

Vedlegg C

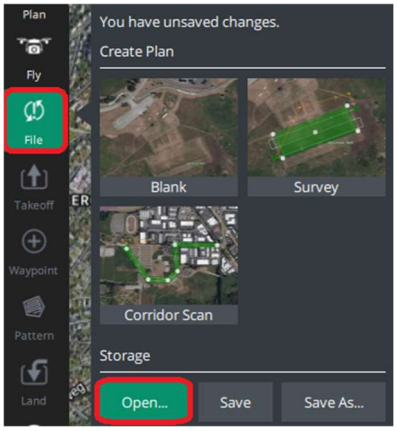
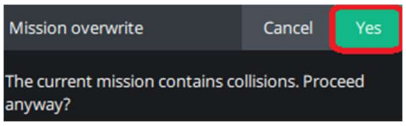
Testskjema for modul 2 og 3

Test hovedmodul 2&3

Dato:			08.05.2023
Gjennomføringsdato:			09.05.2023
Test gjennomføres av:			Torbjørn Houge
Mottaker:			Torbjørn Houge, Sigurd Gossé
Hva skal testes?			Hovedmålet med testen er å vise implementasjon av modul 2 og modul 3.
Godkjent test? (Kryss av)	Godkjent 	Ikke godkjent	
Signatur mottaker		 Dato: 9/5-23	

Varsling av ugyldig rute

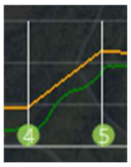
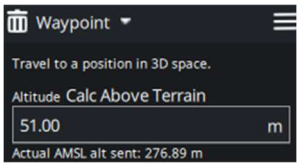
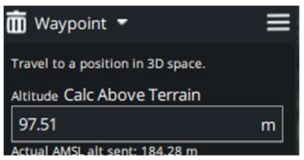
Beskrivelse:
Viser implementasjon av vindu for å bekrefte at brukeren ønsker å lagre ruten selv dersom ruten inneholder kollisjoner.

	Instruksjon	Forventet resultat	Bilde
1	Trykk på *Plan* til venstre i vinduet, deretter trykk på *Open* og velg testplan.plan.	Åpner filmappe.	
2	Trykk på *File* og deretter *Save*. Observer varsling om at ruten inneholder kollisjonselementer. Trykk *Cancel.*	Popupmeny som ber bruker bekrefte lagring av rute med terrengkollisjon.	

Interpolering av nye z-verdier

Beskrivelse:

Viser implementasjon av mulighet for automatisk justering av hele banen. Funksjonaliteten som er lagt til er; automatisk korrigering av kolliderende ruter, automatisk korrigering av for bratte seksjoner, automatisk korrigering av minimum høyde over terreng for hele ruten.

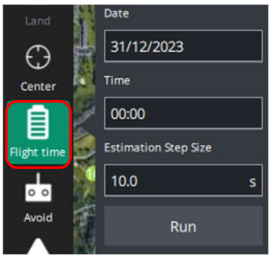
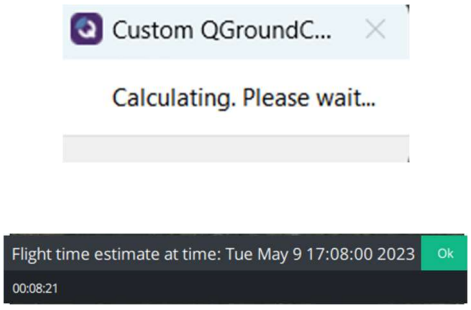
	Instruksjon	Forventet resultat	Bilde
3	Observer strekning mellom waypoint **7** og **8**. Strekket kolliderer med terrenget. Trykk på *Avoid* til venstre i brukergrensesnittet	Nye waypoints legges til i strekket som hadde kollisjon.	 
4	Observer strekning mellom waypoint **4** og **5** i terrengprofilen. Strekket har brå stigning, operatør er usikker på om stigningen er brattere enn flyets climb rate. Trykk på *Smooth*, sett climb rate og decent rate til 13.64mph (6.09m/s). Trykk *Run.*	Høyden til waypoint 4 justeres opp.	 
5	Verifiser dette ved å undersøke missionitems til høyre i brukergrensesnittet. Trykk på waypoint 4, og 5.	Waypoint 5 er 51m over bakken, mens waypoint 4 er 97.51m over bakken.	 
6	Til høyre i brukergrensesnittet, i *Mission Start* skal du nå endre M*inimum Flight Alt* til 60m og trykk enter. Observer strekket mellom waypoint **2** og **3**. Her er flyruten nær å kollidere med terrenget.	Waypoints justerer høyden opp, slik at strekket i mellom aldri er lavere enn 60m over bakken.	 

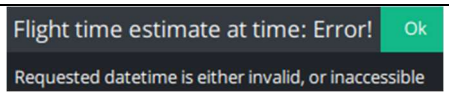
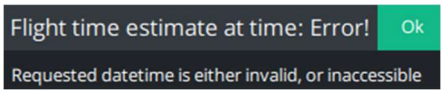
	Trykk på *Keep dist* til venstre i brukergrensesnittet for å sette minimumshøyde for flyruten.		

Estimering av flytid

Beskrivelse:

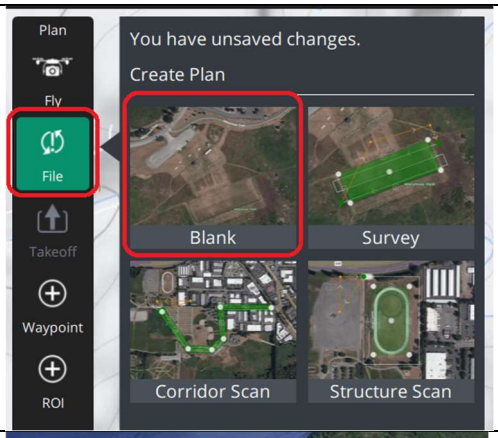
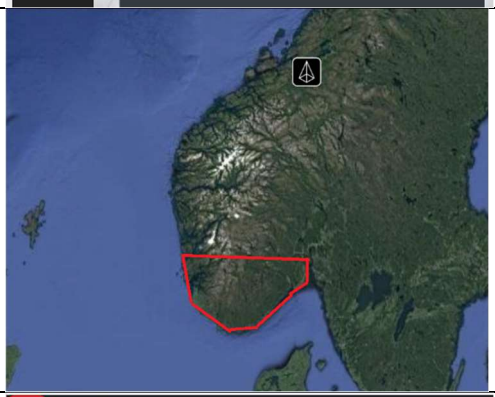
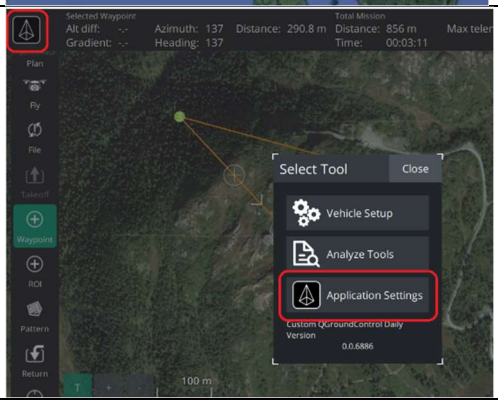
Siden droner opererer ved kun noen titalls meter per sekund i hastighet. Ved disse hastighetene kan vind slå ut mye på tiden det tar en drone å reise en vis strekning. Dermed kan det være ønskelig å få et mer nøyaktig estimat for reisetid som også tar hensyn til påvirkningen til vind på fartøyet.

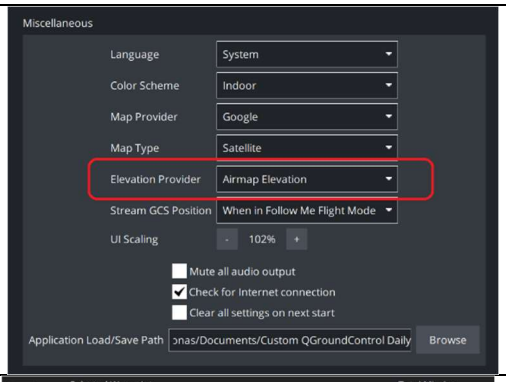
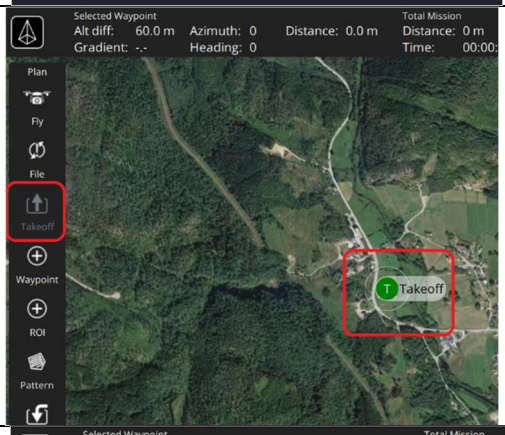
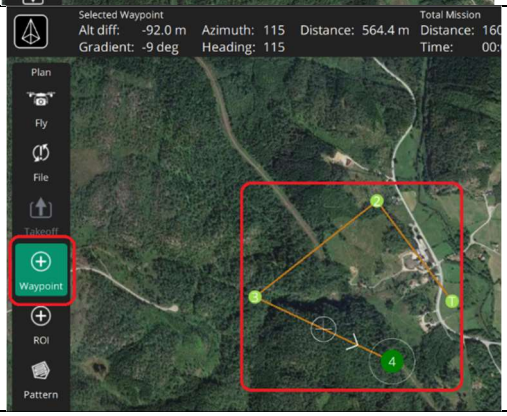
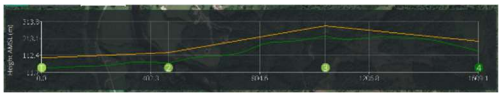
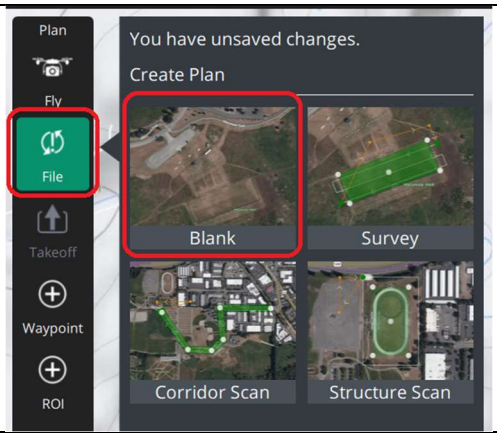
	Instruksjon	Forventet resultat	Bilde
7	Trykk på *Flight time* til venstre i GUI.	Menyen åpnes.	
8	Under *Date,* skriv inn 09/05/2023 og trykk enter på tastaturet for å bekrefte valgt dato. Under *Time,* skriv inn nårværende tidspunkt og trykk enter. Under *Estimation Step Size,* skriv et tall mellom 1-10 i Estimation Step Size. Trykk enter. Trykk på *Run.*	Popup-vindu som viser estimert flytid for ruten.	
9	Trykk på *Flight time.*	Menyen åpner.	
10	Velg et tidspunkt som er innenfor 36 timer frem i tid. Trykk *Run.*	Popup-vindu som viser estimert flytid for ruten. Betydelig raskere fordi data er cached lokalt i programminnet.	
11	Trykk på *Flight time.*	Menyen åpner.	
12	Velg en dato som er 3 dager frem i tid.	Popup-vindu med feilmelding.	

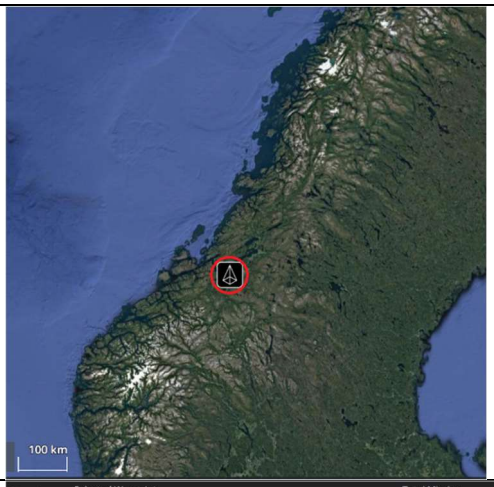
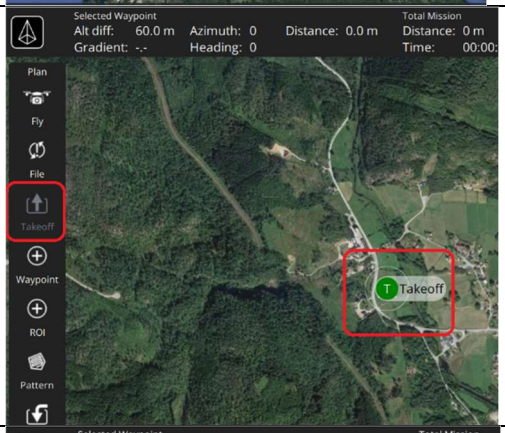
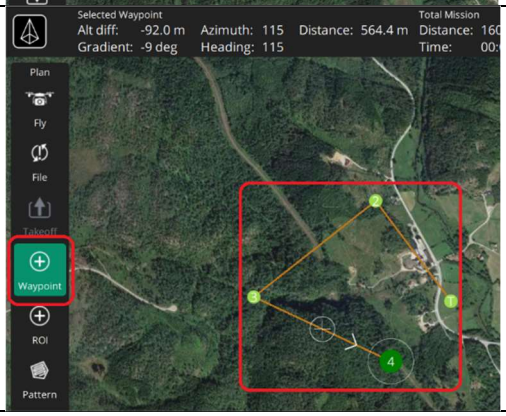
	Trykk *Run.*	Data kan bare hentes inntil 36 timer frem i tid.	
13	Trykk på *Flight time.*	Menyen åpner.	
14	Sett *Date til* 09/05/2023. Velg et ugyldig tidspunkt, f.eks. 37:11. Trykk *Run.*	Popup-vindu med feilmelding. Klokkeslett må være innenfor 00:00-23:59	

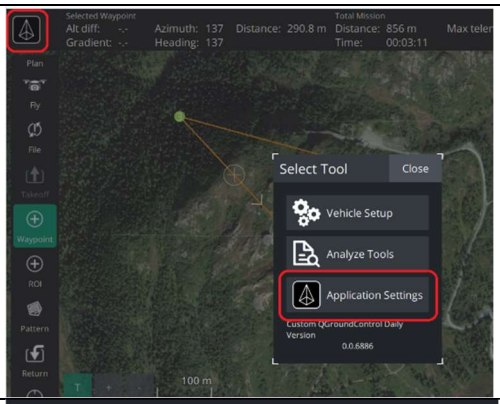
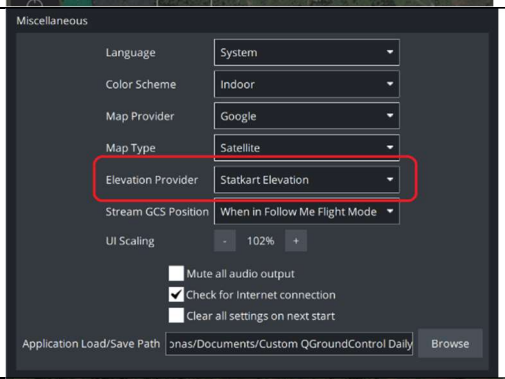
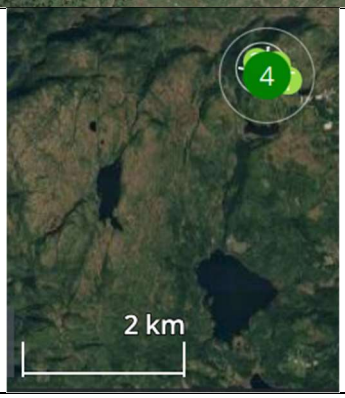
Korrigert test 1 (Norsk kartdata)

Beskrivelse: Det er allerede vist at vi kan hente inn data fra Kartverket. Denne testen har som hensikt å vise at den korrigerte versjonen kan benytte både Airmap og Kartverket for høydemodell, og at den lagrer data i database og runtime-minne for hurtigere tilgang.

15	Instruksjon	Forventet resultat	Bilde
16	Trykk på *File* og velg *Blank* og deretter *yes.*	Startet et nytt tomt mission	
17	Zoom ut slik at du ser hele Norge på kartet. Zoom deretter inn på et vilkårlig sted sør for Oslo, men i Norge. Pass på at kartskala er 250m	Kommer til et sted i Norge under 60 grader nord.	
18	Trykk på Maritime Robotics logoen øverst til venstre og velg Application settings i popup-vinduet.	Komme til innstillinger for programmet	

19	Scroll ned til M*iscellaneous* og sett E*levation Provider* til AirMap Elevation. Trykk på *Back*.	Endret hvilken distributør sin høydemodell som brukes	
20	Trykk på *Takeoff* og sett det på et passende sted.	Startet et mission med en takeoff.	
21	Trykk på *Waypoint* og lag en vilkårlig flyrute med tre Waypoints.	Laget en rute som genererer en terrengprofil.	
22	Observer nederst i brukergrensesnittet at terrengprofilen er synlig.	Ruten skal vise en grønn strek som indikerer høyden til terrenget gitt fra Airmap. Dette viser at Airmap fungerer under 60 grader nord	
23	Trykk på *File* og velg Blank, og *yes*.	Startet et nytt tomt mission	

24	Zoom ut slik at du ser hele Norge på kartet. Zoom deretter inn på et vilkårlig sted i Trondheim. Pass på at kartskala er 250m.	Kommer til et sted i Norge over 60 grader nord.	
25	Trykk på Takeoff og sett det på et passende sted.	Startet et mission med en takeoff.	
26	Trykk på Waypoint og lag en vilkårlig flyrute med tre Waypoints.	Laget en rute som generere en terrengprofil	
27	Observer nederst i brukergrensesnittet at terrengprofilen ikke er synlig.	Gul linje viser at Airmap ikke klarer å henter inn høydedata for dette området.	

28	Trykk på Maritime Robotics logoen øverst til venstre og velg Application settings i popup-vinduet.	Komme til innstillinger for programmet	
29	Scroll ned til miscellaneous og sett elevation provider til Statkart Elevation. Trykk på Back.	Endret hvilken distributør sin høydemodell som brukes.	
30	Observer nederst i brukergrensesnittet at terrengprofilen er synlig.	En grønn strek som indikerer terrenghøyden vises istedenfor kun den gule linjen. Dette	
31	Flytt på et waypoint til et vilkårlig sted i nærheten av ruten.	Observer at terrengprofilen nederst oppdaterer seg kontinuerlig. Dette skjer fordi høydemodellen i området rundt allerede er lagret i både cache og runtime.	
32	Zoom ut slik at kartskala er på 2km.	Et større område skal være synlig for å kunne sette et waypoint langt unna	

33	Sett et waypoint til et vilkårlig sted langt borte fra ruten.	Observer at terrengprofilen bruker tid på å oppdatere seg. Det nye punktet er såpass langt unna at høydedataen må lastes ned for det nye området.		

Vedlegg D

Hellmanneksponent estimat dataanalyse

latitude	longitude	elevation	utc_offset	timezone
63,4375	10,4375	18	0	GMT

time	windspeed	windspeed	winddirec	winddirection_80m (
########	2,06	3,61	61	56
########	2,08	3,56	55	52
########	2,3	4	56	53
########	2,3	3,92	56	52
########	2,33	3,94	59	54
########	2,15	3,71	68	63
########	2,28	3,5	61	59
########	2,78	4,13	60	58
########	3,19	4,77	58	57
########	3,88	6,08	55	54
########	4,11	6,44	49	48
########	4,17	6,3	44	42
########	3,98	5,66	51	42
########	3,42	5,45	52	46
########	3,25	5,2	45	38
########	3,12	4,34	40	29
########	3,02	3,81	34	30
########	2,66	3,28	34	31
########	2,08	2,38	35	22
########	1,7	1,5	28	360
########	1,49	1,42	20	309
########	1,43	2,14	12	281
########	1,14	2,52	345	277
########	1,63	2,51	317	275
########	1,97	2,5	294	270
########	1,92	2,73	279	262
########	1,73	2,81	280	266
########	1,6	3,01	274	264
########	1,43	3,28	258	258
########	1,39	3,77	249	259
########	1,39	3,91	240	266
########	1,28	3,8	231	268
########	0,89	3,61	243	273
########	0,57	3,4	225	284
########	1,39	3,31	330	299
########	1,77	3,16	344	305
########	1,51	3,05	352	311
########	1,97	3,47	336	314
########	2,25	3,84	328	309
########	2,91	4,41	319	303
########	3,47	4,7	319	299
########	3,98	4,83	321	304
########	3,89	4,98	318	308
########	3,69	5,19	311	304
########	3,56	4,49	308	302
########	2,86	3,8	295	297
########	2,92	3,55	301	292
########	2,82	4,27	297	286
########	3	4,53	296	292
########	2,97	5,54	290	287
########	3,4	6,07	284	287
########	3,79	6,53	288	285
########	3,55	6,87	280	283
########	3,57	6,87	281	278
########	3,67	9,4	281	291
########	3,42	9,47	277	292
########	3,82	10,29	276	302
########	5,06	10,93	288	305
########	5,99	10,06	296	295
########	6,31	9,84	297	295
########	6,44	9,48	298	295
########	6,41	10,36	301	292
########	6,47	11,65	302	291
########	5,99	11,31	303	292
########	5,41	10,08	298	293

latitude	longitude	elevation	utc_offset	timezone
63,4375	10,375	0	0	GMT

time	windspeed	windspeed	winddirec	winddirection_80m (
########	2,24	3,58	63	60
########	2,25	3,52	58	55
########	2,47	3,88	58	55
########	2,47	3,74	58	56
########	2,51	3,77	61	58
########	2,53	3,72	72	66
########	2,6	3,44	67	64
########	3	4,02	64	63
########	3,49	4,7	63	61
########	4,22	5,85	59	57
########	4,5	6,2	53	52
########	4,39	6,08	47	44
########	4,28	5,45	53	43
########	3,66	5,02	55	46
########	3,19	4,68	49	42
########	3,05	4,22	41	31
########	2,86	3,33	36	33
########	2,5	2,56	37	31
########	1,84	1,84	41	13
########	1,34	1,21	27	336
########	1,12	1,52	10	293
########	1,2	2,12	355	278
########	1,22	2,41	325	275
########	1,8	2,3	304	272
########	1,8	2,4	289	268
########	1,71	2,63	277	261
########	1,61	2,72	277	264
########	1,5	2,96	270	258
########	1,46	3,29	254	250
########	1,34	3,64	243	254
########	1,44	3,75	236	261
########	1,36	3,81	234	267
########	1,04	3,7	253	272
########	0,63	3,37	252	282
########	1,39	3,36	330	300
########	1,84	3,2	338	309
########	1,88	3,26	335	313
########	2,42	3,82	330	313
########	2,88	4,12	326	309
########	3,2	4,69	321	304
########	3,55	4,66	320	301
########	4,06	4,94	322	306
########	3,98	5,11	321	310
########	3,68	5,24	313	305
########	3,44	4,53	306	301
########	2,73	3,8	294	297
########	2,87	3,58	299	293
########	2,69	4,21	292	288
########	2,82	4,62	293	292
########	3,01	5,47	285	288
########	3,37	5,95	282	287
########	3,92	6,46	289	286
########	3,67	6,74	281	285
########	3,69	7,02	283	281
########	3,91	9,81	283	293
########	3,67	9,55	281	293
########	4,18	10,32	281	302
########	5,65	11,07	293	305
########	6,4	9,89	297	294
########	6,8	9,88	298	296
########	6,85	9,66	299	296
########	6,83	10,49	302	292
########	6,8	11,77	303	291
########	6,3	10,99	303	292
########	5,68	10,23	296	292

latitude	longitude	elevation	utc_offset	timezone
63,375	10,4375	113	0	GMT

time	windspeed	windspeed	winddirec	winddirection_80m (
########	1,13	2,5	45	53
########	1,2	2,55	42	45
########	1,49	2,83	42	48
########	1,64	2,9	38	44
########	1,78	3,04	38	44
########	1,63	2,69	43	45
########	1,8	2,56	34	39
########	1,98	2,91	41	41
########	2,42	3,55	38	40
########	3,25	4,74	45	44
########	3,4	4,96	43	42
########	3,48	5,38	39	35
########	3,54	5,33	43	34
########	3,18	4,69	44	40
########	2,64	4,3	37	31
########	2,28	4,27	29	21
########	2,34	3,52	20	15
########	2,06	2,38	14	15
########	1,51	1,61	8	353
########	0,95	1,28	342	309
########	0,78	1,4	310	274
########	0,85	1,43	315	258
########	0,82	1,53	284	259
########	1,12	1,46	280	254
########	1,4	1,57	270	243
########	1,12	1,8	260	236
########	0,89	1,86	243	234
########	0,86	2,3	234	236
########	1,06	2,5	221	233
########	1,14	2,69	218	239
########	1,22	2,6	215	247
########	1,14	2,75	218	251
########	0,92	3,1	221	255
########	0,85	2,72	225	264
########	1,27	2,91	315	297
########	1,7	3,26	320	310
########	1,35	3,82	312	315
########	2,05	4,32	313	312
########	2,05	4,48	317	309
########	2,69	4,83	315	304
########	2,9	4,8	314	301
########	3,26	4,91	320	303
########	3,26	4,58	320	306
########	3,04	4,99	314	303
########	2,8	4,07	305	298
########	1,97	2,88	294	290
########	1,33	2,4	283	287
########	1,2	2,89	275	284
########	1,22	3,23	279	286
########	1,3	4,37	274	286
########	1,5	5,22	266	287
########	2,24	5,55	280	284
########	2,3	5,32	272	282
########	1,92	5	261	272
########	1,87	8,51	254	288
########	1,77	8,83	254	290
########	2,12	9,98	262	298
########	3,98	10,88	288	302
########	5,55	9,53	293	295
########	5,82	9,62	294	297
########	5,64	8,86	297	294
########	5,64	9,82	300	292
########	5,52	10,76	302	290
########	4,84	9,77	300	293
########	4,35	9,27	293	288

latitude	longitude	elevation	utc_offset	timezone
63,375	10,375	113	0	GMT

time	windspeed	windspeed	winddirec	winddirection_80m (°)
########	1,14	2,44	52	55
########	1,27	2,55	45	48
########	1,49	2,84	48	51
########	1,63	2,83	43	45
########	1,77	2,97	43	45
########	1,63	2,55	47	48
########	1,7	2,41	40	42
########	1,84	2,69	45	42
########	2,34	3,41	40	40
########	3,18	4,46	46	44
########	3,47	4,95	46	43
########	3,61	5,24	42	35
########	3,61	5,13	44	33
########	3,25	4,56	45	38
########	2,62	4,25	40	30
########	2,2	4,31	30	22
########	2,21	3,23	18	16
########	1,96	2,02	15	9
########	1,4	1,49	4	340
########	0,89	1,25	333	299
########	0,81	1,3	300	266
########	0,86	1,39	306	249
########	0,91	1,36	276	253
########	1,1	1,3	275	247
########	1,2	1,53	265	238
########	0,95	1,78	252	232
########	0,81	1,86	240	234
########	0,92	2,22	229	234
########	1	2,55	217	228
########	1,17	2,66	211	236
########	1,22	2,51	215	247
########	1,14	2,75	218	251
########	0,92	3,08	221	257
########	0,78	2,61	230	266
########	1,28	2,95	309	298
########	1,84	3,4	315	313
########	1,72	4,1	306	317
########	2,41	4,53	312	313
########	2,41	4,56	318	308
########	2,91	4,96	319	304
########	3,04	4,69	316	304
########	3,48	4,8	321	306
########	3,42	4,56	322	308
########	3,19	4,96	319	304
########	2,92	3,76	308	299
########	1,79	2,6	297	293
########	1,24	2,12	284	289
########	1,22	2,79	279	285
########	1,14	2,91	285	286
########	1,4	4,11	270	288
########	1,61	5,15	263	288
########	2,24	5,29	280	285
########	2,31	5,07	275	285
########	1,9	5,02	267	275
########	1,84	8,86	257	294
########	1,75	9,05	257	293
########	2,3	10,08	268	300
########	4,35	11,23	293	304
########	5,77	9,57	296	297
########	6,08	9,85	295	299
########	6,04	9,03	299	296
########	5,97	9,98	301	294
########	5,85	10,86	303	292
########	5,2	9,58	300	295
########	4,52	9,46	295	290

Estimat av Hellman konstant for 63.4375,10.4375

Hellman konstant:	0,3	
Gjennomsr	Median	Standard avvik
0,26	0,23	0,15

YR	DWD Estimat
0,3	0,269785
0,3	0,258431
0,3	0,266122
0,3	0,256407
0,3	0,252622
0,3	0,262361
0,3	0,206107
0,3	0,190352
0,3	0,193478
0,3	0,216005
0,3	0,215974
0,3	0,198435
0,3	0,169345
0,3	0,224087
0,3	0,226024
0,3	0,158716
0,3	0,111747
0,3	0,100757
0,3	0,064793
0,3	-0,06019
0,3	-0,02314
0,3	0,193865
0,3	0,381463
0,3	0,207605
0,3	0,114577
0,3	0,169265
0,3	0,233266
0,3	0,303897
0,3	0,399227
0,3	0,479827
0,3	0,497361
0,3	0,523285
0,3	0,673374
0,3	0,858834
0,3	0,417249
0,3	0,278725
0,3	0,338087
0,3	0,272247
0,3	0,25706
0,3	0,19992
0,3	0,145908
0,3	0,093085
0,3	0,118792
0,3	0,164038
0,3	0,111613
0,3	0,136661
0,3	0,09395
0,3	0,199514
0,3	0,198183
0,3	0,299808
0,3	0,278721
0,3	0,261628
0,3	0,317497
0,3	0,314795
0,3	0,452294
0,3	0,489789
0,3	0,476533
0,3	0,370361
0,3	0,249334
0,3	0,213673
0,3	0,185942
0,3	0,230876
0,3	0,282831
0,3	0,305657
0,3	0,299265

Estimat av Hellman konstant for 63.4375,10.375

Hellman konstant:	0,1	
Gjennomsr	Median	Standard avvik
0,24	0,21	0,14

YR	DWD Estimat
0,1	0,225487
0,1	0,215217
0,1	0,217182
0,1	0,199509
0,1	0,195626
0,1	0,185388
0,1	0,134632
0,1	0,140744
0,1	0,143145
0,1	0,157065
0,1	0,154114
0,1	0,156617
0,1	0,116215
0,1	0,151948
0,1	0,184317
0,1	0,156145
0,1	0,073169
0,1	0,011405
0,1	0
0,1	-0,04908
0,1	0,146858
0,1	0,273677
0,1	0,327384
0,1	0,117879
0,1	0,138346
0,1	0,207022
0,1	0,252182
0,1	0,326878
0,1	0,390706
0,1	0,480568
0,1	0,460274
0,1	0,495395
0,1	0,610314
0,1	0,806442
0,1	0,424459
0,1	0,266122
0,1	0,264713
0,1	0,219522
0,1	0,172192
0,1	0,183839
0,1	0,130837
0,1	0,094344
0,1	0,120185
0,1	0,169954
0,1	0,132367
0,1	0,159033
0,1	0,106303
0,1	0,215405
0,1	0,237399
0,1	0,287259
0,1	0,27338
0,1	0,240227
0,1	0,292323
0,1	0,309283
0,1	0,442362
0,1	0,459907
0,1	0,434623
0,1	0,323444
0,1	0,2093
0,1	0,179659
0,1	0,165306
0,1	0,206352
0,1	0,263836
0,1	0,267589
0,1	0,282948

Estimat av Hellman konstant for 63.375,10.4375

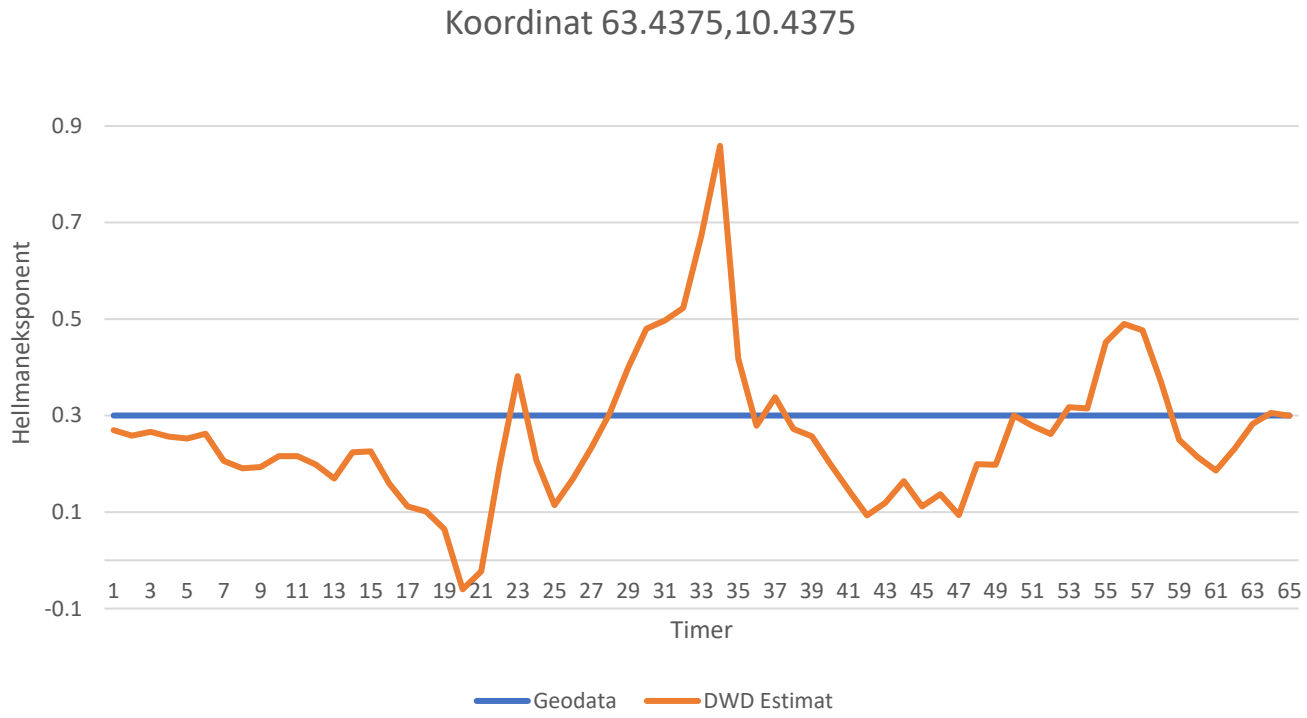
Hellman konstant:	0,25	
Gjennomsr	Median	Standard avvik
0,32	0,28	0,16

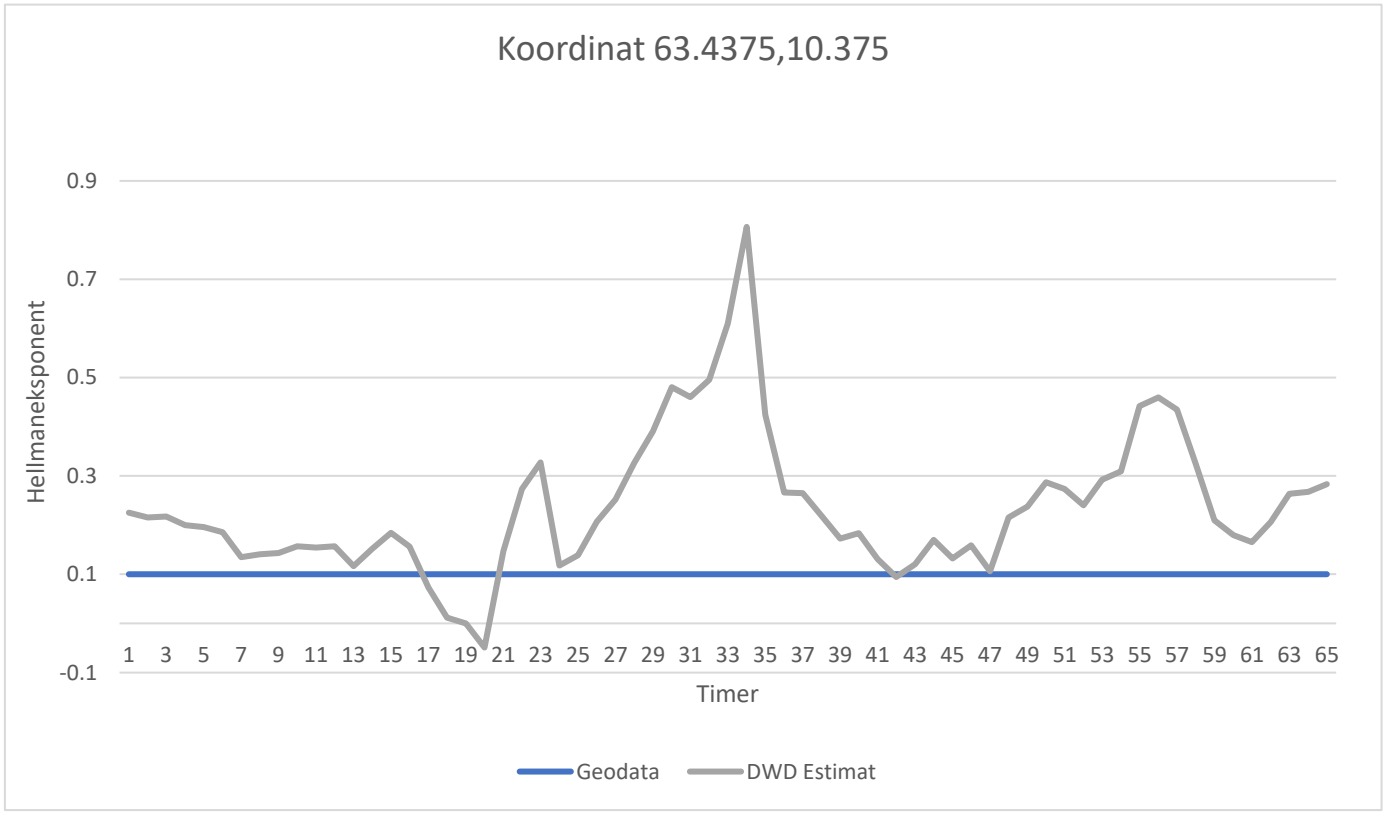
YR	DWD Estimat
0,25	0,381868
0,25	0,362488
0,25	0,308497
0,25	0,274119
0,25	0,257398
0,25	0,240911
0,25	0,169382
0,25	0,185173
0,25	0,184271
0,25	0,181482
0,25	0,181602
0,25	0,209506
0,25	0,196795
0,25	0,186854
0,25	0,2346
0,25	0,301734
0,25	0,196356
0,25	0,069439
0,25	0,030837
0,25	0,143381
0,25	0,281294
0,25	0,25016
0,25	0,299945
0,25	0,12749
0,25	0,055113
0,25	0,228166
0,25	0,354475
0,25	0,473075
0,25	0,412621
0,25	0,412857
0,25	0,363877
0,25	0,423466
0,25	0,584187
0,25	0,559357
0,25	0,39873
0,25	0,313112
0,25	0,500204
0,25	0,358469
0,25	0,375958
0,25	0,281472
0,25	0,242327
0,25	0,19695
0,25	0,163492
0,25	0,238323
0,25	0,179867
0,25	0,182624
0,25	0,283869
0,25	0,422678
0,25	0,468218
0,25	0,583041
0,25	0,599696
0,25	0,43633
0,25	0,403264
0,25	0,460274
0,25	0,728707
0,25	0,772888
0,25	0,744992
0,25	0,483613
0,25	0,259996
0,25	0,241673
0,25	0,217204
0,25	0,266676
0,25	0,320979
0,25	0,337784
0,25	0,363851

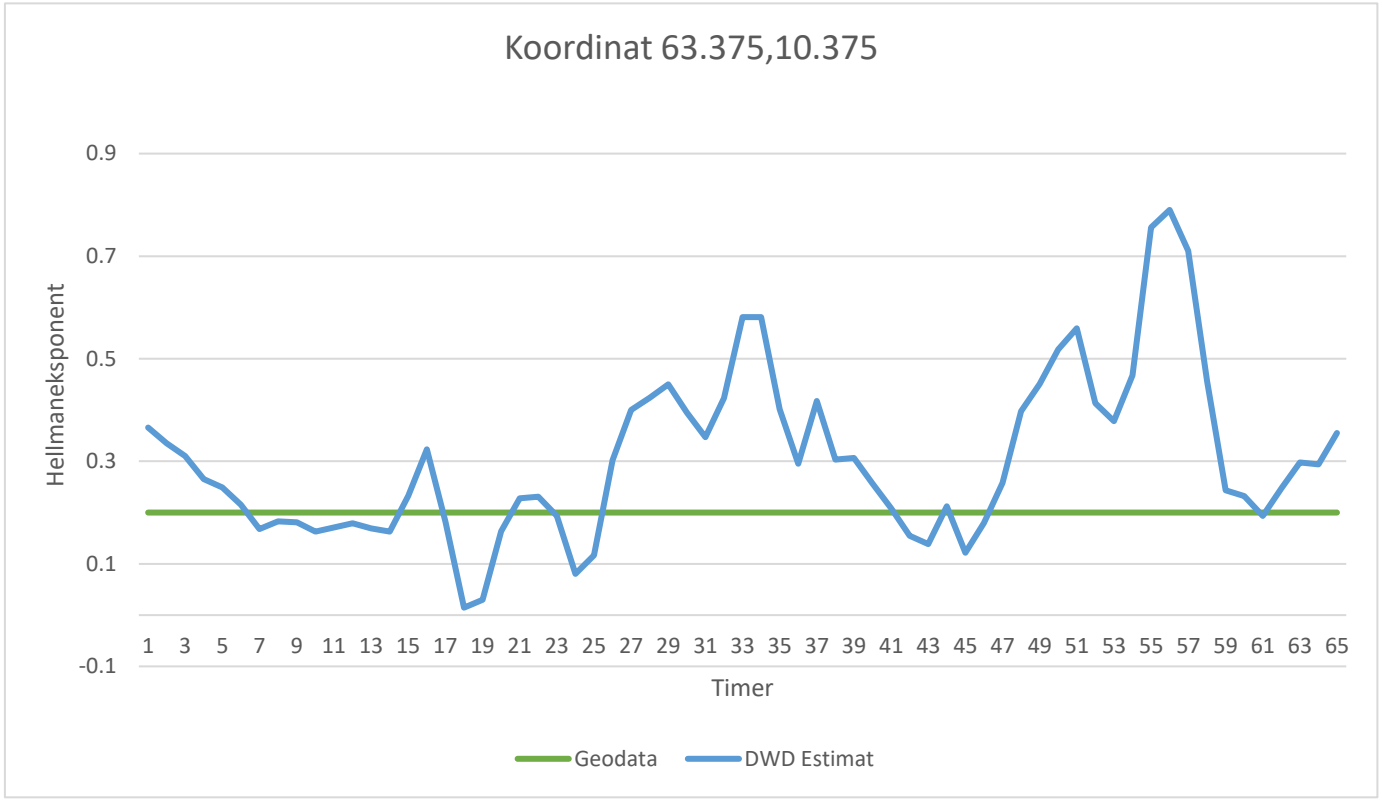
Estimat av Hellman konstant for 63.375,10.375

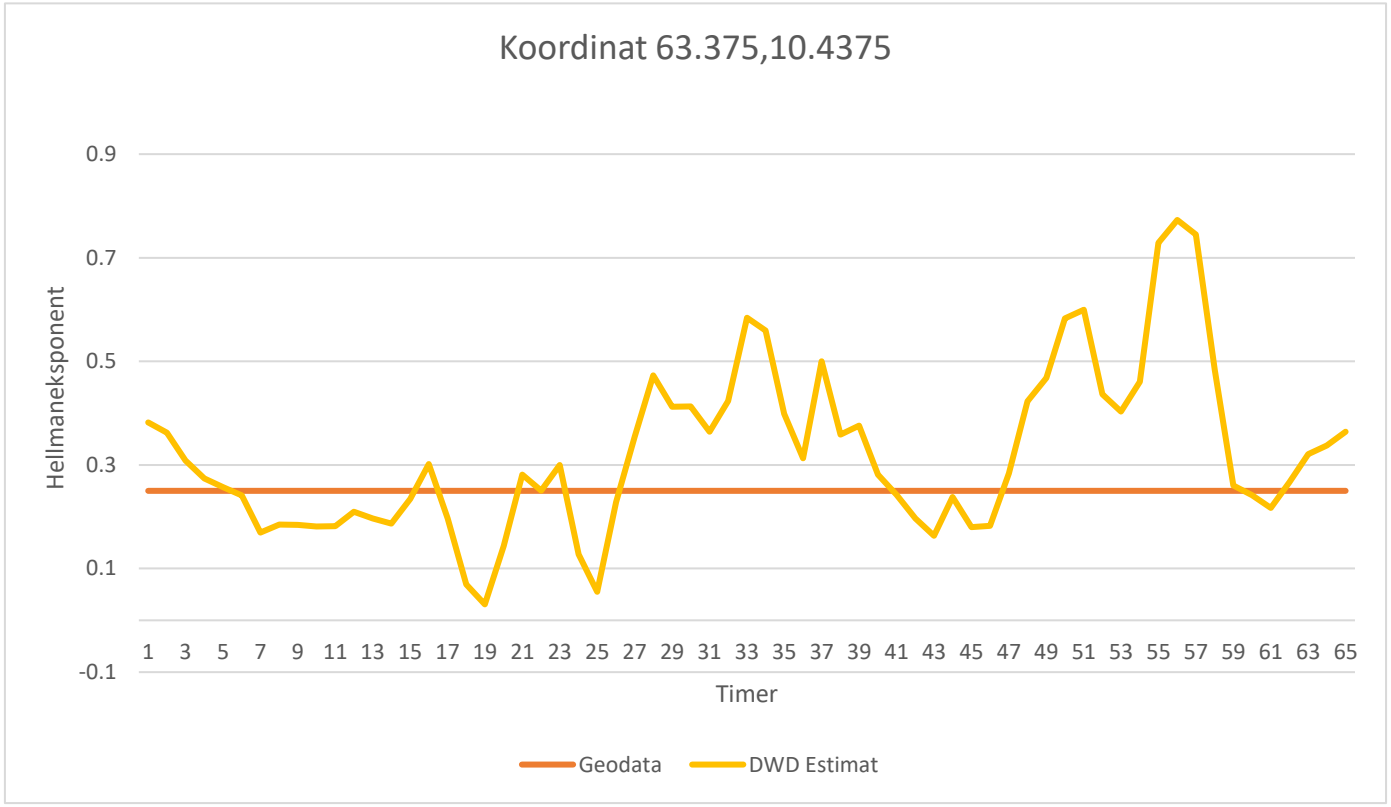
Hellman konstant:	0,2	
Gjennomsr	Median	Standard avvik
0,31	0,27	0,16

YR	DWD Estimat
0,2	0,365949
0,2	0,335223
0,2	0,310193
0,2	0,26531
0,2	0,248905
0,2	0,215208
0,2	0,167833
0,2	0,182633
0,2	0,181088
0,2	0,162672
0,2	0,170831
0,2	0,179189
0,2	0,168987
0,2	0,162865
0,2	0,232632
0,2	0,323395
0,2	0,182496
0,2	0,014501
0,2	0,029962
0,2	0,16335
0,2	0,227506
0,2	0,230892
0,2	0,193223
0,2	0,080336
0,2	0,116832
0,2	0,301959
0,2	0,39977
0,2	0,423618
0,2	0,450166
0,2	0,394973
0,2	0,346935
0,2	0,423466
0,2	0,581075
0,2	0,580835
0,2	0,401524
0,2	0,295276
0,2	0,417738
0,2	0,303493
0,2	0,306667
0,2	0,25644
0,2	0,208506
0,2	0,154649
0,2	0,138346
0,2	0,212261
0,2	0,121588
0,2	0,179517
0,2	0,257908
0,2	0,397795
0,2	0,450662
0,2	0,517904
0,2	0,559171
0,2	0,413256
0,2	0,378031
0,2	0,467229
0,2	0,755867
0,2	0,790188
0,2	0,710597
0,2	0,45609
0,2	0,243316
0,2	0,232017
0,2	0,193392
0,2	0,247103
0,2	0,297505
0,2	0,293838
0,2	0,355172


Koordinat 63.4375,10.4375
0.9
0.7
0.5
0.3
0.1
-0.1
Hellmaneksponent
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65
Timer
Geodata
DWD Estimat


Koordinat 63.4375,10.375
Hellmaneksponent
0.9
0.7
0.5
0.3
0.1
-0.1
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65
Timer
Geodata
DWD Estimat


Koordinat 63.375,10.375
0.9
0.7
0.5
0.3
0.1
-0.1
Hellmaneksponent
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65
Timer
Geodata
DWD Estimat


Koordinat 63.375,10.4375
Hellmaneksponent
0.9
0.7
0.5
0.3
0.1
-0.1
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65
Timer
Geodata
DWD Estimat

Vedlegg E

Poster

Terrengfølgende ruteplanlegging for drone

NTNU
Kunnskap for en bedre verden

Institutt for teknisk kybernetikk

Jesper Wille Jensen, Jonas Hoel Klüver, Sander Løvdahl Nygård, Tormod Skorpe Skjolden

MARITIME ROBOTICS

Sammendrag

QGroundControl, en ruteplanleggingsprogramvare, er modifisert for å bedre passe bruksområdene til Maritime Robotics. Dette innebærer å endre kilde for høydedata til Kartverkets geodata API, for å hente data som dekker hele Norge. Det er lagt til funksjonalitet for å korrigere ruter som kolliderer, endre ruter som er for bratte, og sørge for at ruten holder en gitt høyde over terrenget til en hver tid. Det er også lagt til funksjonalitet for estimering av flytid som tar høyde for vind.

QGroundControl

QGroundControl er gratis, har åpen kildekode, og kan dermed modifiseres etter behov. QGroundControl brukes til å planlegge ruter til autonome droner, og overvåke sensordata under flyging. Programmet kommuniserer med dronen over kommunikasjonsprotokollen MAV-Link.

Statens kartverk som datakilde for høydedata

Den opprinnelige kilden for høydedata til QGroundControl mangler data nord for 60 breddegrad. Dette er løst ved å legge inn muligheten å bytte til Kartverkets geodata API som kilde for høydedata. Det er integrert på en slik måte at alle funksjonaliteter avhengig av høydedata bevares.

Figure: Falk VTOL drone

Automatisering av repetitive oppgaver

For å forenkle arbeidsflyten ved bruk av QGroundControl er det automatisert flere repetitive oppgaver. Når programmet detekterer kollisjoner i ruten kan brukeren trykke "Auto-Avoid" knappen for å automatisk legge den over hindringen. Dersom et segment av ruten er for bratt er det mulig å trykke "Smooth" knappen for å korrigere dette. For tilfeller der ruten ikke holder en gitt minimumshøyde over bakken er det mulig å trykke "Kepp dist" knappen for å sikre dette.

(a) Opprinnelig rute

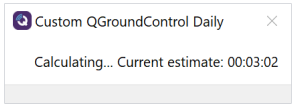
(b) Korrigert kollisjoner

(c) Justert høydeendrings vinkel og høyde over terreng

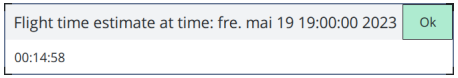
Avansert estimering av flytid

Som et supplement til den eksisterende funksjonaliteten er det lagt til en mer avansert estimat av flytid. Estimatet tar høyde for vind og vertikale endringer i ruten. Denne funksjonen sjekker også om vinden i alle rutesegment, og varsler dersom kraftig vind sammenlignet med dronens maksimale hastighet. Menyen for avansert estimering av flytid gir også mulighet for å legge til ønsket avreise tidspunkt inntil 36 timer fram i tid.

(a) Normal tidsestimering



(b) Beregner tidsestimering



(c) Avansert tidsestimering

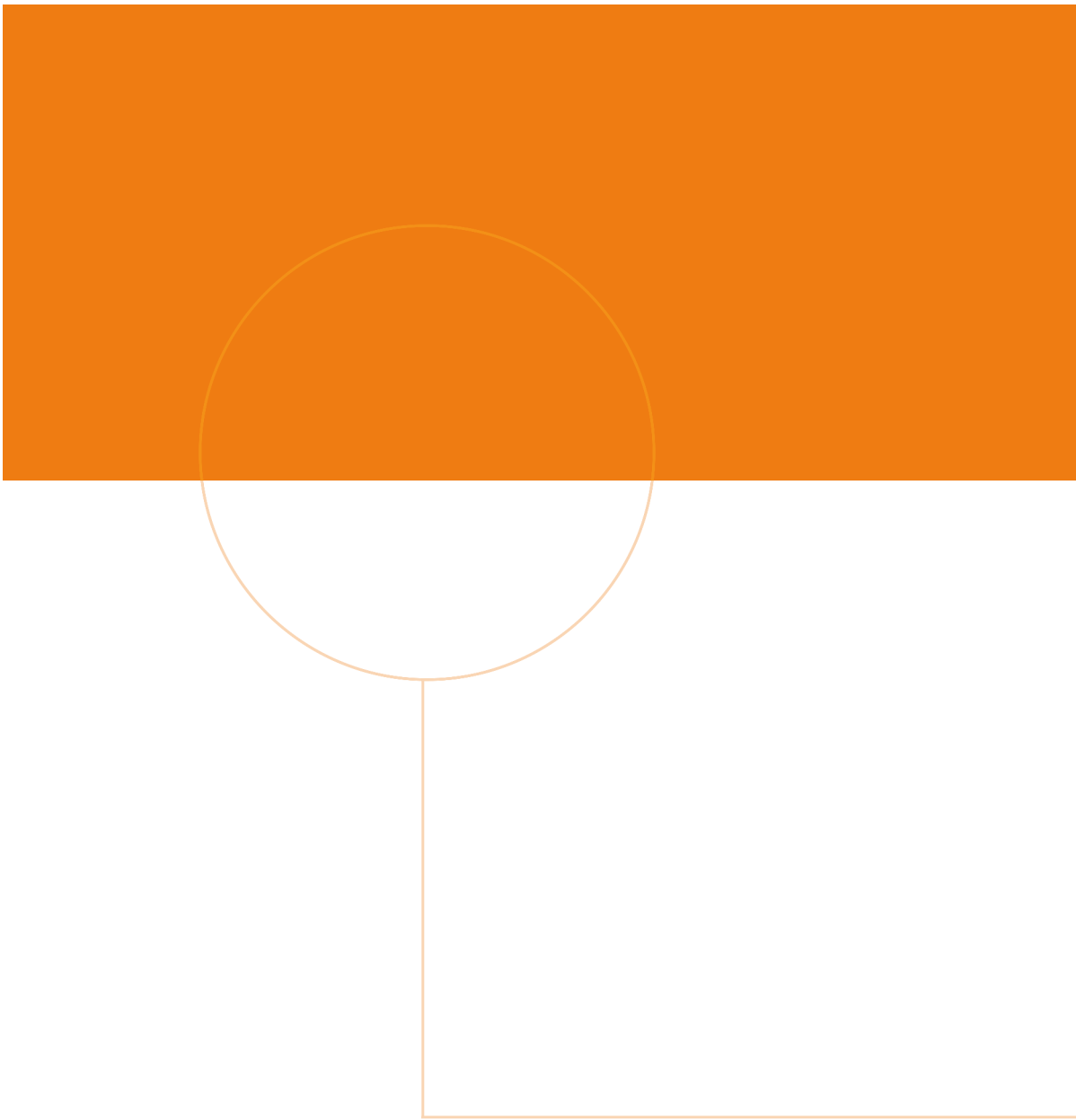
Konklusjon og videre arbeid

Arbeidsflyten i bruk av QGroundControl har blitt forbedret med nye funksjonaliteter som er implementert. Funksjonalitetene fjerner mange repetitive og tidkrevende oppgaver som oppstår under planlegging av flyruter. Tidsestimering gir mulighet til å velge et mer optimalt tidspunkt for flygingen, som forenkler planleggingsprosessen.

Ved videre arbeid anbefales å forbedre nettverksarkitetkur for nedlasting av vinddata. Nedlastingen gjøres sekvensielt, som er lite effektivt siden det kan kjøres parallelt. Funksjonalitet for fiksing av kollisjon, korrigering av stignings- og nedstigningrate for segmenter, og sørge for at ruten holder en gitt høyde over terrenget til en hver tid er nå delt opp i tre forskjellige knapper. De kan være hensiktsmessig å se på muligheten for å samle disse funksjonene i én knapp.

Kilder

- Jesper Wille Jensen, Jonas Hoel Klüver, Sander Løvdahl Nygård, Tormod Skorpe Skjolden. *Terrengfølgende ruteplanlegging for drone* 2023
- Jesper Wille Jensen, Jonas Hoel Klüver, Sander Løvdahl Nygård, Tormod Skorpe Skjolden. Repositoriet for tilpasset versjon av QGroundControl https://github.com/jesperwj/Maritime-Robotics-QGC-Bachelor-2023



NTNU
Kunnskap for en bedre verden

MARITIME ROBOTICS