2014

# Flytrafikk over Hessdalen (BO14-102)

## Gruppe BO14-G12

Bacheloroppgave, Avdeling for informasjonsteknologi, Høgskolen I Østfold

Peter Christopher Bach
Robin Holm
Daniel Dohrmann Nilsen

HiØ
18. mai 2014

# HØGSKOLEN I ØSTFOLD

*Avdeling for Informasjonsteknologi*
*Remmen*
*1757 Halden*
*Telefon: 69 21 50 00*
*URL: www.hiof.no*


# BACHELOROPPGAVE

| | | |
|---|---|---|
| Prosjektkategori:<br>Bachelor | X | Fritt tilgjengelig |
| Omfang i studiepoeng:<br>20 Poeng | | Fritt tilgjengelig etter |
| Fagområdet:<br>Dataingeniør | | Tilgjengelig etter avtale med oppdragsgiver |

| | |
|---|---|
| Tittel:<br>Flytrafikk over Hessdalen | Dato (leveranse):<br>22.05.14 |
| Forfattere:<br>Daniel Dohrmann Nilsen<br>Peter Christopher Bach<br>Robin Holm | Veileder:<br>Børre Stenseth |
| Avdeling / Program:<br>Avdeling for informasjonsteknologi | Prosjektnummer:<br>BO14-102 |
| Oppdragsgiver:<br>Erling P. Strand | Kontaktperson hos oppdragsgiver: |

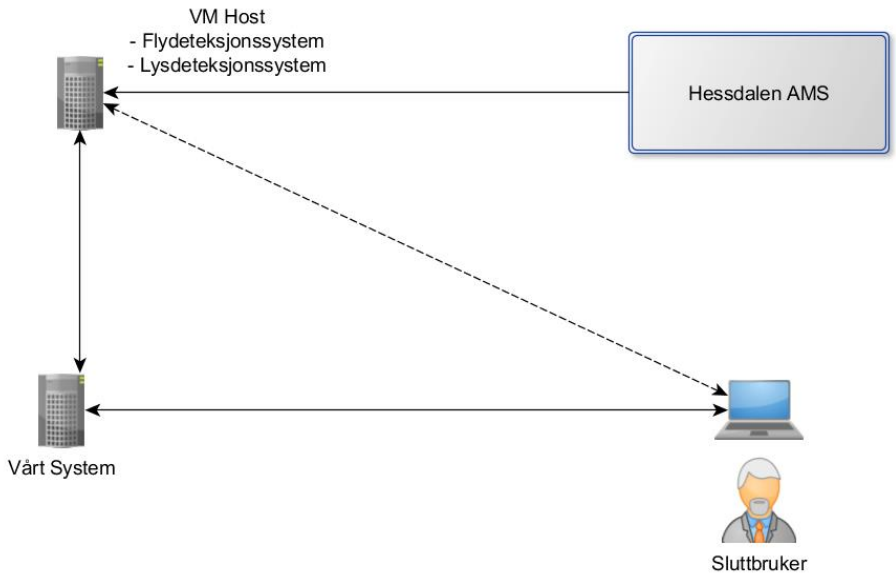

Ekstrakt:
I Hessdalen kan det til tider observeres lysfenomen som ingen hittil har funnet en forklaring på. Derfor er det satt opp utstyr som prøver å fange opp og analysere disse hendelsene. En potensiell kilde kan være flytrafikken over området. En tidligere prosjektgruppe ved HiØ har satt opp to separate systemer, ett lysdeteksjonssystem og et flydeteksjonssystem. Sistnevnte samler inn data om fly i området og logger dette til en database, slik at det kan avklares om et detektert lysfenomen var et fly eller ikke.

Vår oppgave går ut på å hente ut denne informasjonen og fremstille det på en webside. Her finnes det mange mulige løsninger, noen mer gunstige enn andre. Derfor dreier rapporten seg i hovedsak om hvilke ulike teknologier som kan tas i bruk for å løse de ulike problemstillingene, samt en redegjørelse for hvordan *vi* har valgt å gjøre det, og hvorfor.

# Forord

Vår oppgave gikk ut på å sette opp en webside som lot brukeren se flytrafikk over Hessdalen i et valgt tidsrom. Dette var basert på et system satt opp av en tidligere prosjektgruppe, som logger data om flytrafikken til en database. En enkel oversikt over system ser du i Figur 1.1: Enkel systemoversikt.

**Figur 1.1: Enkel systemoversikt**




Hessdalen AMS (Automatic Measurement Station) samler inn data om flytrafikken i området via en sensor. Dette sendes videre til en dedikert maskin plassert på oppdragsgivers kontor, her kalt VM Host. Denne inneholder både fly- og lysdeteksjonssystemene. Vårt system presenterer dataene produsert av flydeteksjonssystemet på en webside. Sluttbruker kan så bruke data fra lysdeteksjonssystemet og sammenlikne dette med dataene presentert av vårt system. Slik er det mulig å avgjøre om et lysfenomen var et fly. Vår oppgave forholdt seg i hovedsak til flydeteksjonssystemet, fordi lysdeteksjon og detaljer rundt dette ikke var direkte relevante for vår løsning, selv om det til syvende og sist er det som er målet med oppgaven.

# Sammendrag

Forskning på lysfenomenet i Hessdalen har foregått i mange år. Flere bacheloroppgaver har vært gjennomført opp gjennom årene, alle med hensikt å få en dypere forståelse for hva dette fenomenet forårsakes av. Sist vår ble det kjørt et prosjekt med hensikt å bruke kameraet plassert i Hessdalen til å analysere bildene tatt av lysfenomenet. Som et grenprosjekt til dette ble deteksjon av fly innført, med den hennsikt å filtrere ut flytrafikk som en potensiell kilde (/feilkilde). Dette prosjektet ble aldri helt fullført, men var vellykket i å sette opp en flydetektor som logger data til en database. Det er her vårt prosjekt kommer inn i bildet.

Vi bygger videre på systemet de satt opp, for å gjøre det mulig å effektivt hente ut og analysere disse dataene. Flytrafikk over området er betydelig. Dette får store konsekvenser for utviklingen av vårt system, som ikke bare må gjøre det enkelt og effektivt for brukeren å finne frem i havet av informasjon, men som også må forholde seg til tekniske begrensninger i alle delene av systemet. Databasen, webserveren og klientapplikasjonen som alle kjører på hvert sitt system må tåle påkjenningene av å behandle relativt store datamengder. I tillegg vil også prosjektet naturlig nok bli sterkt påvirket av mulighetene og begrensningene til flydeteksjonssyetemet og databasen, som vårt produkt bygger direkte videre på. Dette fører til at vi i vår oppgave må ta hensyn til begrensningene i alle de involverte systemene, og hvilke krav som er rimelige å stille til disse. Slik forebygger vi eventuelle problemer som kan oppstå. Dette er med målet å minimere innvirkningen problemer i andre systemer kan påføre vårt eget.

Etter at produktet var ferdig utarbeidet ble det integrert med hovedsiden for Hessdalenprosjektet. Rapporten var omtrent ferdigstilt da dette skjedde, slik at hoveddelene av rapporten er utarbeidet uten hensyn til dette. Derfor har vi inkludert en kort notis om denne flyttingen der det er relevant, selv om det hadde liten direkte innvirkning på produktet og rapporten.

# Takk Til

Takker Dick Olausson fra den tidligere prosjektgruppen for å ha vært hjelpsom og lett tilgjengelig når vi hadde spørsmål angående fly- og lysdeteksjonssystemene, samt databasen fra fjorårets bacheloroppgave, og for å ha stilt til hjelp når det gjaldt modifikasjoner til dette systemet.

# Innhold

# Figurliste

**Figurer:**



**Tabeller:**



**Eksempler og kodelister:**



**Formler:**

# Ordbeskrivelse

| Ord | Beskrivelse |
|---|---|
| Flight,<br>Overflyvning,<br>Rute | Et fly som foretar en enkelt reise over det aktuelle området. Hvis det samme flyet kommer tilbake ved en senere annledning vil dette bli klassifisert som en ny overflyvning. |
| Hessdalen AMS,<br>AMS,<br>Automatisk Målestasjon,<br>Automatic Measurement Station | Målestasjonen plassert i Hessdalen. Denne inneholder kameraer og ulike sensorer, blant annet flysensoren som er kilden til dataene vi bruker i vår oppgave. |
| Sighting,<br>Punkt,<br>Observasjon,<br>Flyobservasjon | Et enkelt punkt der flyet er blitt "observert" og loggført. Flere slike punkter utgjør til sammen *en* overflyvning. |
| Marker | Et objekt som vises på kartet i form av et bilde. Dette kan for eksempel representere en enkelt flyobservasjon. |
| Heatmap | En visningsmodus for kartet. Individuelle fly vil ikke vises i denne modusen, i stedet vises et farget lag over kartet, der fargen samsvarer med sammenlagt flyintensitet over området. |
| Hardkodet | Oppførsel eller verdier som er en fast del av programkoden og ikke lett kan endres. |
| Issue,<br>Issuetracker | I vår sammenheng bruker vi dette om formelle gjøremål og et program for å behandle disse. Hovedsakelig en enkeltstående del som skal implementeres i koden, eller noe som skal forandres: "Gjøre det mulig for brukeren å bestemme hvilken ikon som brukes for en Sighting", er et eksempel på et Issue. |
| Repositorium ("Repo"),<br>Commit,<br>Changeset,<br>Push | Kildekodekontroll som gjør det enklere å organisere og ta vare på eldre versjoner av kodebasen.<br>En commit/changeset er en formell gruppering av forandringer som er gjort på kodebasen.<br>Changesets kan pushes til det sentrale repositoriet og bli allment tilgjengelig. Før dette skjer, er forandringen lokal hos den aktuelle utvikleren. |
| Fly- og lysdeteksjonssystemer | Dette henviser spesifikt til de to systemene utviklet av den tidligere prosjektgruppen, de systemene vi forholder oss til. Det er altså ikke snakk om potensielle andre lysdeteksjonssystemer eller sensorer i Hessdalen. Disse er begge programmer som kjører på samme maskin. |
| VM-Host | Et begrep brukt om den dedikerte maskinen på oppdragsgivers kontor som fly- og lysdeteksjonssystemene kjører på. Denne kjører også Windows på en virtuell maskin(VM), som inneholder en dekoder for dataene fra flytrafikken, derav navnet. |

# 1. Introduksjon

I dette kapittelet gir vi en oversikt over oppgaven vår, samt strukturen for resten av rapporten.

## 1.1 Prosjektgruppen

Gruppen består av tre personer: Daniel Dohrmann Nilsen, Peter Christopher Bach og Robin Holm. Vi har jobbet sammen med de fleste fag helt fra vi tok forkurs for ingeniørutdanning i 2010/2011, frem til denne bacheloroppgaven. Vi har alle den samme kompetansen som inkluderer blant annet behandling av databaser samt generell programmering i flere språk.

## 1.2 Oppdragsgiver

Oppdragsgiveren til prosjektet er Erling P. Strand. Han er høgskolelektor ved HiØ og har vært involvert i lysfenomenet i Hessdalen i mange år. Han har også vært veileder for en rekke ulike bacheloroppgaver som er blitt gitt i forbindelse med dette.

## 1.3 Oppdraget

En tidligere prosjektgruppe ved HiØ har satt opp både et fly- og et lysdeteksjonssystem, der førstnevnte tar i bruk en flydetektor satt opp ved den automatiske målestasjonen i Hessdalen. Denne tar inn informasjon om forbipasserende fly og logger dette til en database. Vår oppgave går ut på å hente ut dataene fra flydeteksjonssystemet og presentere dem på en webside på en gunstig måte, slik at det kan avklares om en lysobservasjon i lysdeteksjonssystemet var et fly.

Sluttproduktet blir en webside med et kart som viser flytrafikken i et gitt tidsrom, j.fr. Flightradar24 (Flightradar24 AB, 2014). Det skal være mulig for brukeren å sette et start- og sluttidspunkt, og deretter skal ikoner som representerer flytrafikken i dette tidsrommet plasseres ut på kartet. Disse ikonene skal kunne klikkes på for å gi mer utfyllende informasjon. Alt dette hentes direkte fra informasjonen lagret i databasen.

## 1.4 Formål, leveranser og metode

I delkapitlene under gir vi en kort presentasjon av målet med oppgaven og arbeidsmetodene vi tok i bruk for å oppnå dette.

### 1.4.1 Formål

Formålet med oppgaven er å detektere og forstå lysfenomenene som forekommer i Hessdalen, der vår oppgave spesifikt går ut på å gjøre det lettere å avklare om detekterte lysfenomen bare var et forbipasserende fly.

Dette oppnås ved å sette opp en webside som lar brukeren få oversikt over flytrafikken i et valgt tidsrom. Det skal være lett for brukeren å hente ut de aktuelle dataene, og få all relevant informasjon for å så kunne sammenlikne dette med et detektert lysfenomen som foregikk på et gitt tidspunkt.

All relevant informasjon fra databasen skal knyttes opp mot flyobservasjoner som blir representert som ikoner på kartet. Dette kan for eksempel være flyhøyde, vinklelen fra basestasjonen og opp til flyet, fart, osv. Disse er for å videre kunne spesifisere hvor og i hvilken tilstand flyet befant seg i på det aktuelle tidspunktet, slik at det blir lettere å bruke dette som en filtreringsmetode.

Merk også at disse dataene kan ha en annen funksjon en bare bortfiltrering av feilkilder, for det er ikke utenkelig at det kan være en direkte sammenheng mellom flytrafikken og disse lysfenomenene. Kanskje fly som flyr i en spesifikk høyde ofte forekommer på samme tid som et lysfenomen? Kanskje data om flytrafikken pleier å være mangelfull rundt tidspunktet der et lysfenomen har oppstått? Dataene kan ha mange bruksområder, derfor er det viktig at vi presenterer så mye data som mulig, da vi ikke har noen forutsetninger for å bestemme hva som er relevant med tanke på bruksområdet.

### 1.4.2 Leveranser

Hovedresultatet av prosjektet vil være en webside som gjør det mulig for brukeren å analysere flytrafikken over Hessdalen i tidsrom brukeren selv bestemmer. Selve rapporten som beskriver systemet, og hvordan vi kom frem til vår løsning er også en stor del av leveransen.

### 1.4.3 Metode

Vi har en relativt håndfast problemstilling med mange mulige løsninger på det samme problemet. Derfor blir en av våre største oppgaver å ta gode valg når det gjelder de ulike teknologiene vi har til rådighet. Vår "metode" blir således å argumentere grundig for og mot de ulike løsningene vi har, da selve implementeringen av disse er relativt uproblematisk.

Det skal også nevnes at vårt sluttprodukt er et stort program bestående av mange komponenter som alle skal kommunisere med hverandre. Det kan være problematisk å delegere ut arbeidsoppgaver når man snakker om programmering av denne typen, derfor har vi hovedsakelig *en* utnevnt person som tar for seg den overordnede programmeringsjobben. Løsningene fremkommer så i fellesskap, men det er hovedsakelig en person som tar seg av implementasjonen. Dette er med unntak av mindre arbeidsoppgaver som for eksempel utarbeiding av algoritmer for å løse spesifike problemer, som godt kan behandles som en separat arbeidsoppgave uten at dette hindrer fremgangen i prosjektet.

## 1.5 Rapportstruktur

**Kapittel 2:** Analyse av problemstillinger

Her gir vi en grundig oversikt over oppdragsgivers ønsker og kravene til oppgaven generelt, og hvilke muligheter vi har for å implementere disse. Her er det mange alternative teknologier og metoder for å løse en gitt problemstilling, derfor går vi grundig til verks med å se på de ulike alternativene. Merk at direkte valg av teknologier og løsninger vil bli dekket i senere kapitler, her presenteres bare alternativene med sine fordeler og ulemper. Bakgrunnen for denne oppdelingen er at kapittel 2 er ment å gi en overordnet oversikt over en gitt problemstilling, med alle muligheter vi har til rådighet. Slik får man en bedre kontekst og forståelse for valgene vi tok, og hvorfor. Disse blir så dekket i detalj i kapittel 3, med begrunnelse og forklaring av valgene vi tok. Merk at det er direkte sammenheng

mellom nummerering av delkapitlene i kapittel 2 og de i kapittel 3. Det vil imidlertid være flere tilskudd i kapittel 3.

En stor del av analysen går også på vårt forhold til den tidligere prosjektgruppen og deres system. Merk også at oppgaven gitt i prosjektbeskrivelsen er relativt kortfattet, men at mange utvidelser av vår oppgave, samt forandringer i det tidligere systemet, kan gjøres. Derfor tar vi i dette kapittelet også opp hvordan vi begrenser omfanget av vår oppgave med tanke på slike potensielle utvidelser.

**Kapittel 3:** Planlegging og utforming

Etter å ha gitt en oversikt over mulige teknologiske og konseptuelle løsninger, med fordeler og ulemper, i kapittel 2, tar vi i kapittel 3 opp de faktiske løsningene vi valgte å bruke, samt begrunnelsen bak disse. Som oftest vil et delkapittel i kapittel 3 ha et direkte relatert delkapittel i kapittel 2, som gir en overordnet oversikt over problemstillingen. Ut ifra dette utarbeider vi så en plan for hvordan løsningen skal gjennomføres.

**Kapittel 4:** Implementasjon

I dette kapitelet ser vi på hvordan løsningene er implementert og hvordan disse er knyttet sammen. Altså en full bekrivelse av systemet og dets komponenter.

**Kapittel 5:** Testing og evaluering

Kapittel 5 tar vi for seg de ulike testfasene prosjektet har gjennomgått, og resultatene av disse. Deretter konkluderes kapitlet med en evaluering av hvor godt produktet lever opp til oppdragsgivers og våre egne krav.

**Kapittel 6:** Diskusjon

I dette kapittelet ser vi på sluttproduktet, og stiller spørsmål ved om det inneholder den funksjonalitet som er nødvendig for å løse problemstillingen det er laget for. Altså om systemet ble utformet hensiktsmessig til analyse av flytrafikk, med bakgrunn i lysfenomenene i Hessdalen.

Her sammenlikner vi sluttproduktet med målene satt i 1.4, og ser om kravene vi satt blir oppfylt.

Vi ser også på om oppdragsgiver er fornøyd med sluttproduktet, og om det er noe som kunne ha blitt forbedret.

Til sist snur vi også dialogen mot oss selv, og ser om vi er fornøyd med produktet vi har levert, hva vi har lært, og eventuelt hva vi hadde gjort annerledes hvis vi skulle arbeidet på et liknende prosjekt igjen.

**Kapittel 7:** Konklusjon

Vi ser på et kort sammendrag av kapittel 6, diskusjonen, og trekker en konklusjon.

# 2. Analyse av problemstillinger

I dette kapittelet tar vi for oss en grundig analyse av oppdragsgivers ønsker, og ser på de ulike teknologiene og mulighetene vi har til rådighet for å løse de relaterte problemstillingene. Problemstillingene er generelt sett presentert i rekkefølgen vi møtte på dem, men merk at selve konklusjonen og implementasjonen til en gitt problemstilling først blir dekket i kapittel 3. Det er også en direkte sammenheng mellom kapittelnummereringen i kapittel 2 og kapittel 3.

## 2.1 Fly- og Lysdeteksjonssystemet

I 2013 ble det kjørt et bachelorprosjekt ved Høgskolen i Østfold, med oppgave å detektere og analysere lysfenomenet som forekommer i Hessdalen. Oppgaven gikk ut på å analysere bilder av lysfenomenene som blir tatt av et kamera plassert ved den automatiske målestasjonen i Hessdalen. For å kunne filtrere ut potensielle feilkilder med tanke på forbipasserende fly, ble et gren-prosjekt innført i oppgaven. De satte opp en sensor som tok inn data fra forbipasserende fly, og deretter logget disse til en database. Datamaskinen som bearbeider dataene og logger dem til databasen er en dedikert maskin som er plassert på oppdragsgivers kontor. Selve lysdeteksjonssystemet kjører også på samme maskin.

Dette prosjektet kom aldri helt i mål. Systemet som ble satt opp er relativt ustabilt, og kunne ha vært forbedret. Oppdragsgiver spesifiserte at det var ønskelig å få dette systemet oppgradert, hvis vi fikk mulighet til dette. Dette er beskrevet i 2.1.1. Denne oppgaven ble også oppført som en egen bacheloroppgave på anbefaling av prosjektgruppen som satt opp dette systemet, da de var klar over problemet. Denne oppgaven ble aldri tildelt en gruppe, slik at problemet var et faktum da vi begynte med vår oppgave. Vi måtte tidlig bestemme oss for i hvor stor grad vi ville involvere oss i dette arbeidet, da det ville få store konsekvenser for vårt eget prosjekt og hvor mye vi kunne forvente å få utrettet.

Da vi var helt avhengig av dette systemet i arbeidet med å bygge vår egen løsning, måtte vi sette oss tilstrekkelig inn i deres system, slik at vi var klar over hvordan det fungerte, og eventuelle feil og mangler det måtte ha.

### 2.1.1 Oppgradering av lys- og flydeteksjonssystem

Det ble tidlig klart at lys- og flydeteksjonssystemene ikke var optimale. Disse måtte oppgraderes. Hvis vi bestemte oss for å ta på oss jobben å forbedre disse systemene, ville dette føre til at vi måtte dedikere mye tid fra vårt eget prosjekt for å sette oss grundig inn i deres system, samt å utforme de nødvendige løsningene. På den annen side ville dette bety at vi hadde full kontroll på hvordan systemene arbeidet, og som en følge av dette ville det bli betraktelig enklere å gjøre grensesnittet mellom flydeteksjonssystemet og vårt eget optimalt.

Et av problemene med den eksisterende løsningen var en minnelekkasje i lysdeteksjonssystemet, som førte til at vertsmaskinen sluttet å fungere. Dette førte til "hull" i dekningen, der informasjonen ikke ble lagret til databasen.

Systemet var også ganske utdatert, både hardware- og software-messig, noe oppdragsgiver også ville forbedre dersom vi fikk tid. Hvis vi valgte å ta på oss denne jobben ville det bety at vi hadde full kontroll på når systemet gikk offline og liknende, slik at dette ikke fikk noen konsekvenser for vårt prosjekt.

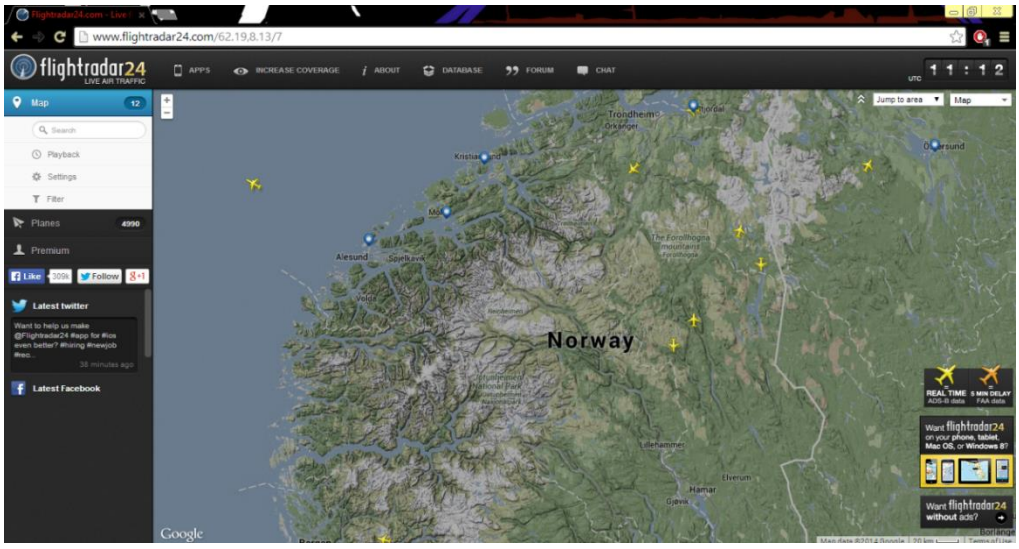
## 2.2 Brukergrensesnittet

Oppdragsgivers ønske var at dataene fra databasen skulle presenteres på en webside i form av flyikoner på et kart. Her skulle brukeren selv kunne velge hvordan dataene skulle fremstilles ved hjelp av diverse verktøy og hjelpefunksjoner (som å begrense tidsintervallet for uthenting av data, hvor mye data som ble vist frem på en gang, og liknende). Dette medførte at vi måtte ta stilling til en hel rekke teknologiske valg, med formålet å gjøre brukeropplevelsen så smidig og effektiv som mulig.

### 2.2.1 Valg av kartprogramvare

Oppdragsgiver ønsker et system liknende websiden FlightRadar24 (Flightradar24 AB, 2014) som vist i Figur 2.1.

**Figur 2.1: FlightRadar24**




Her hadde vi mange muligheter for valg av kartprogramvare. Oppdragsgiver spesifiserer at det eneste som trengtes av kartprogramvaren var en oversikt over selve Hessdalen, derfor hadde statiske kart uten zooming og "panning" ikke vært et problem. Likevel mente vi at det ville være bedre å ha disse verktøyene tilgjengelige hvis dette lot seg gjøre.

En mulighet vi hadde var å lage kartprogramvaren manuelt. Vi kunne altså ha brukt et statisk kart-bilde av Hessdalen og omegn, og spesifisert koordinatene for kartets plassering, slik at alle flylokasjoner ble riktige i forhold til dette. Dette ville ha ført til at vi fikk full kontroll over alle aspekter ved kartet, slik at vi ikke ville møte på situasjoner der en gitt kartprogramløsning hadde uløselige mangler.

Likevel ville også et slikt valg ha medført enormt mye arbeid fra vår side, ikke minst med tanke på all testing som måtte ha blitt gjort. Hvis vi brukte en ferdig kartløsning kunne vi være relativt trygge på at eventuelle feil ville bli rettet opp i fremtiden.

Når det gjalder ferdige kartløsninger finnes det flere leverandører: amCharts, OpenLayers, arcGIS, Google Maps og mange flere. Bare dette lille utvalget av kartprogramvare har meget liknende funksjonalitet, og det var klart tidlig at de fleste kartløsninger ville innfri de fundamentale kravene for vårt produkt. Problemstillingen vi sto ovenfor var at det var nesten umulig å få full oversikt over alle krav vi potensielt kunne stille i fremtiden. Vi måtte velge en kartløsning der vi kunne forvente at det meste allerede var dekket, selv om vi ikke hadde de spesifikke kravene klare fra starten av.

En stor faktor i beslutningen var populariteten til de forskjellige løsningene. Dersom det fantes et enormt antall brukere for et produkt, var det sannsynlig at alle tenkelige former for funksjonalitet allerede var implementert, da andre sannsynligvis hadde stilt krav til dette tidligere. Samt at systemet ville bli vedlikeholdt i fremtiden av leverandøren.

Noe av det viktigste for oss når det gjaldt valget av kartprogramvare var at kartene var rikt på detaljer når det ble zoomet inn på småsteder, spesielt rundt Hessdalen. Vi hadde også tenkt til å legge ut et stort antall ikoner på kartet, kanskje også animere disse, derfor var det best hvis programvaren gjorde slike operasjoner lett tilgjengelige og effektive.

Når det gjaldt verktøy for manipulasjon av kartet for sluttbrukeren (zooming, panning, etc.), ville det være en fordel hvis dette allerede var inkludert i kartprogramvaren. At kartprogramvaren starter opp og oppdaterer seg raskt var også meget relevant.

Til sist var prisen for løsningen en betydelig faktor. Open Source løsninger var altså å foretrekke.

### 2.2.2 Animasjon av fly

En potensielt sentral del av oppgaven var animasjon av fly, selv om dette ikke var en del av oppdragsbeskrivelsen. Animasjon ville gjøre det lettere for sluttbrukeren å få oversikt over et hendelsesforløp med store mengder data. Oppdragsgiver gjorde det klart tidlig i prosjektgangen at dette var ønskelig, men ikke kritisk. Derfor ble prosjektet utarbeidet med dette som et aktuelt tilskudd, etter at det grunnleggende var på plass.

Når det gjaldt mulighetene for animasjon av flyikonene, måtte vi se på om vår valgte kartløsning hadde dette innebygd. Hvis dette ikke var tilfellet måtte vi se på hvordan vi kunne løse det "manuelt". Dette kunne potensielt bli en tidskrevende prosess, derfor hadde innebygget animasjon vært å foretrekke, men dette kunne vi ikke regne med at var tilgjengelig i en gitt kartløsning. Alternativt kunne det være en situasjon der de innebyggede mulighetene for animasjon ikke var gunstige for vårt bruksområde. Det var altså ikke utenkelig at vi måtte implementere dette manuelt, uansett tilfelle.

### 2.2.3 Hvordan vise frem data fra databasen

Relevante data for flyene skal være lett tilgjengelige. Vi hadde flere muligheter når det gjaldt å vise frem data for et gitt fly. Vi kunne for eksempel ha en "popup" på flyikonet som viste disse dataene når brukeren spesifikt spurte om mer informasjon (ved å trykke på et flyikon).

Det var også mulig å gjøre det slik at all data alltid var tilgjengelig, slik at brukeren kunne se all relevant informasjon umiddelbart. Dette ville vært en meget bra løsning hvis det var mulig å presentere dette på en oversiktlig måte. Problemet var at vi hadde store mengder data, som raskt kunne bli overveldende for brukeren. Det skal også nevnes at bruksområdet for løsningen er å analysere flytrafikken i et spesifikt tidsrom, gjerne også i et spesifikt geografisk område. Derfor var det mer naturlig å la brukeren selv gjøre en "forespørsel" om mer utdypende informasjon der dette var relevant.

En annen mulighet vi også hadde tilgjengelig var å vise frem deler av informasjonen ved hjelp av smart bruk av ikonene. Man kunne for eksempel vise frem hvilken rotasjon flyet hadde på et gitt tidspunkt ved å faktisk rotere flyikonet på kartet. Høyden til flyet kunne representeres ved størrelsen på flyikonet, og så videre. Hvis vi valgte å gå for en slik løsning måtte vi være forsiktige så ikke dette endte opp med å være rotete og uoversiktelig på det endelige kartet.

Til sist må det nevnes at arbeidsgivers ønsker selvsagt var kritiske når det gjaldt presentasjonen av data, da dette var en av de viktigste faktorene for brukbarheten til vår løsning.

### 2.2.4 Verktøy og hjelpefunksjoner

Når det gjaldt uthenting og manipulering av websiden, måtte vi gjøre det mulig for brukeren å hente ut de dataene som var mest relevante på en rask og effektiv måte. Her måtte bruksområdet for vårt produkt tas med i betraktningen. Hvis et lysfenomen ble detektert i Hessdalen ville dette bli registrert i en database sammen med et bilde av det faktiske fenomenet. Deretter var det så tenkt at sluttbrukeren av vårt system skulle kunne gå inn og sammenlikne tidspunktet for lysfenomenet med flytrafikken som forekom rundt denne tiden. På denne måten ville det være mulig å se om det aktuelle lysfenomenet var et fly eller ikke. Brukeren går altså inn i vårt system og setter relevante parametere for å begrense mengden data, som så vises på kartet.

Det er også flere fly som ikke inneholder senderen som flydeteksjonssystemet bruker for å ta inn data om flyet, altså er det er helt usynlig for systemet. På grunn av dette ville muligheten for å la brukeren se *sannsynligheten* for fly være av interesse. Spesielt i områder (både geografisk og tidsmessig) der databasen ikke inneholdt data. Et slikt problem kan angripes på flere ulike måter. Hvis avanserte sannsynlighetsberegninger skulle utføres, var det meget viktig å ta med i betraktningen at dette ikke måtte bli for krevende operasjoner for de systemene det eventuelt skulle kjøre på. Se 2.2.4.1.

Altså måtte vi veie kontroll brukeren skulle ha over datautvalget opp mot hvor brukervennlig systemet ville bli som følge av dette. Hvis brukeren hadde full kontroll over datauthentingen, ville dette samtidig ført til at systemet ble tungvint å bruke. Alternativt ville for dårlige verktøy og valgmuligheter ha ført til at brukeren kanskje ikke hadde mulighet til å få tak i den ønskede informasjonen.

#### *2.2.4.1 Sannsynlighet for fly*

Et problem med systemer som logger flydata er at ikke alle fly inneholder senderen som slike systemer baserer seg på. Derfor vil visse fly ikke kunne registreres i slike systemer. Det kunne altså være nyttig å ha en oversikt over hvor stor sannsynlighet det var for at et fly hadde passert et område, basert på annen trafikk som er blitt registrert der.

Man kunne ha sett på regelmessigheten av ruter. For eksempel kunne det hende at et gitt fly alltid passerte Hessdalen ca. klokken 14.00 hver dag. Dette kunne så blitt satt sammen for alle fly-ruter for å gi en oversikt over flyene i området. Problemet med denne tilnærmingen var at fly som ikke var en del av en standard rute aldri ville få noe utslag, og da lysfenomenet kan dukke opp hvor som helst, ville denne type analyse typisk ikke ha vært spesielt nyttig.

En annen mulighet ville være å la brukeren spesifisere et punkt på kartet, for så å få en indikasjon på trafikken der. Dette ville ha krevd relativt tunge regneoperasjoner på store mengder data, og gitt et resultat som i beste fall var noe upålitelig, i verste fall fullstendig unyttig. En annen vinkling av denne problemstillingen var en såkalt "heatmap". Dette er et slags filter man legger over kartet som angir intensiteten av (i vårt tilfelle) flytrafikk. Ved hjelp av dette er det mulig for bruker å få en generell oversikt over hvor det er sannsynlig at fly har passert, basert på andre fly i et relativt stort tidsrom.

## 2.3 Database og backend

Som nevnt i 2.1, var databasen en kritisk faktor i vårt arbeid. En stor del av oppgaven gikk ut på å sette oss inn i hvordan denne var definert, og deretter bygge vår applikasjon basert på kravene dette stilte til grensesnittet mellom de to systemene.

Her må det nevnes at databasen allerede hadde logget data for omtrent et halvt år før vår oppgave begynte. Dette førte til at eventuelle problemer som kunne oppstå med dataene, måtte løses på andre måter enn å forandre på databasens struktur. Vi måtte heller bygge et robust system som tålte potensielle feil og mangler i disse dataene.

### 2.3.1 Relevante kolonner

Databasen inneholdt mange ulike typer data for en gitt flyobservasjon (altså en "punkt-prøve" av flyet, med dataene det hadde i løpet av et gitt øyeblikk). Hver observasjon er relatert til et sett med data. I databasen representeres dette som en rad med flere kolonner, slik vi ser i Tabell 2.1: Databaselayout:

**Tabell 2.1: Databaselayout**

| Hexident | Postime | Flightid | Latpos | Longpos | Track | Altitude | Osv. |
|---|---|---|---|---|---|---|---|
| 06A052 | 1370085739 | QTR991 | 61.48540 | 11.30859 | 295 | 32000 | |
| 06A052 | 1370085774 | QTR991 | 61.51859 | 11.16329 | 295 | 32000 | |
| 06A052 | 1370085796 | QTR991 | 61.53936 | 11.07198 | 295 | 31975 | |

En rad består av flere datafelter, bare et mindre utvalg er tatt med i denne tabellen. **Hexident** er en unik identifikator på flyet, **postime** er tidspunktet da flyobservasjonen fant sted og **flightid** er et slags rutenummer for flyet. De andre feltene gir data om flyets tilstand, slik som koodinater, rotasjon, høyde, osv. Disse blir dekket i større detalj senere.

Til sammen vil så flere slike rader forme en "flight/overflyvning", ett fly på én tur i nærheten av Hessdalen.

Her måtte vi ta et valg angående hvilke kolonner fra databasen vi skulle vise brukeren, og hvilke som var nødvendige for systemet selv.

Her måtte vi være nøye slik at vi ikke tok bort data/kolonner som kunne være av interesse for sluttbruker. På den annen side kunne det også være problematisk hvis vi tok med kolonner som virket forvirrende, eller som ga et feilaktig inntrykk.

### 2.3.2 Feil og mangler i data

Det kan oppstå situasjoner der noen av dataene i databasen er feil, eller rett og slett mangler. Vi hadde flere forskjellige måter å takle slike situasjoner på.

Vi kunne gå inn i databasen ved hjelp av SQL-spørringer og "rydde opp" ved å ta bort feilaktige eller mangelfulle felter, eller korrigere data hvis disse var mangelfulle (ved hjelp av for eksempel interpolasjon). Merk at denne type forandring ville krevd konstant opprettholdelse i ettertid, samt at det ville tatt bort noe av kredibiliteten til dataene. Dataene kunne ha vært manipulert, og beregnet informasjon kunne ha avvik og mangler.

Vi kunne også la databasen være slik den var, men prosessere dataene under uthenting, slik at vi gjorde en aktiv filtrering og opprydding før dataene ble vist frem til sluttbruker. På denne måten ville det også vært mulig å informert bruker om data som hadde blitt behandlet før fremvisning, slik at brukeren var klar over situasjonen.

En annen mulighet ville vært å gått inn i flydeteksjonssystemet og korrigert mangelfulle data før de ble lagret til databasen. Merk at i denne situasjonen måtte vi ha reflektert eventuelle forandringer på de dataene som allerede var blitt logget til databasen fra før.

En siste mulighet var å *ikke* behandle slike feil, og heller varsle brukeren hvis dataverdier så feilaktige ut. Denne løsningen kunne være ideell med tanke på effektmålet for oppgaven (å få klarhet i lysfenomenet), da vi ikke hadde noen reell forutsetning for å bestemme hva som var "feil" eller ikke, slik nevnt i 1.4.1. Hvis et fly hadde rare verdier kunne nettopp dette være relatert til lysfenomenet. I en slik situasjon ville bortfiltrering eller korrigering av slike "feil"-data være katastrofalt.

Dataene i databasen kunne også være mangelfulle eller overflødige med tanke på tettheten i data. Relatert til dette var problemstillingen som oppsto hvis bruker valgte å hente ut for store mengder data. Dette er beskrevet i 2.3.2.1.

#### *2.3.2.1 Uthenting av for store mengder data*

Selv om vi løste problemer med datamengder i databasen kom vi likevel til et punkt der brukeren hadde mulighet til å velge å hente ut så mye data at grensene ble sprengt. Dette var uunngåelig da data som ble aggregert over tid, der all data skulle være tilgjengelig for brukeren, i teorien hadde uendelig stor mengde på sikt. Derfor måtte vi ta stilling til hva vi skulle gjøre hvis denne grensen ble nådd.

En løsning vi kunne ty til for å løse dette problemet var å sette en fast øvre grense for hvor mye data en bruker kunne hente ut. For eksempel en uke eller liknende. Men hvis vi gjorde dette ville vi redusert fleksibiliteten til systemet vårt kraftig, derfor ønsket vi å heller finne en annen løsning.

En annen ting vi kunne ha gjort var å hente ut dataene i flere separate "pakker". For eksempel ved at vi først hentet ut 10 000 rader, deretter 10 000 rader til, osv. På denne måten ville det til enhver tid være plass til allokering på webserveren, og det ble slutt-brukeren som måtte ta hensyn til plass på sin egen maskin. Dette var rimelig, da brukeren selv valgte hvor mye data som skulle uthentes.

Uansett løsning var det et faktum at vi måtte gi brukeren tilbakemelding på hvor mye data som ble uthentet, og om brukeren ville gå videre med dette. Dette ble gjort ved å sette en grense på 24 timer, hvis et større utvalg enn dette ble utført måtte så brukeren få beskjed om at dette kunne vise seg å være problematisk. Denne grensen la vi så til i en konfigurasjonsfil, slik at denne kunne forandres i ettertid hvis den viste seg problematisk.

Selv med dette ville uthentingen av store mengder data være et faktum. Systemet kunne ikke uten videre transportere store mengder data til sluttbrukerens maskin. Det var hovedsakelig webserveren som ville få problemer. Skulle vi teste oss frem, og sette en hardkodet grense for hvor mye som kunne hentes ut? Problemet med denne fremgangsmåten var at denne grensen potensielt sett vil kunne variere, samt at det ville begrense fleksibiliteten til systemet, slik nevnt tidligere. Slik beskrevet i 3.4.3, så fikk vi senere mulighet for å øke grensen på hvor mye data som kunne allokeres på webserveren. Likevel var det ikke utenkelig at denne måtte senkes igjen ved et senere tidspunkt, derfor var det best hvis vår løsning på dette var fleksibel.

### 2.3.3 Dataflyt

Vi hadde en database med alle data vi trengte. Disse dataene skulle presenteres på en webside. For å løse dette hadde vi mange teknologier og metoder tilgjengelige i alle ledd av prosjektet.

For det første måtte brukeren ha et brukergrensesnitt i en nettleser, her var det ikke spesielt mange alternativer å snakke om. Vi måtte ha en webside med HTML (og CSS) som fundament. Utover dette var det utallige muligheter. Dataene måtte hentes ut fra en database og sendes til klientmasinen. Formatet på denne dataoverføringen er diskutert i 2.3.3.1. På webserveren som tok seg av denne uthentingen og videresendingen måtte vi kjøre et skript som tok seg av databasetilkoplingen og diverse dataprosesseringer. En utbredt løsning på dette er PHP, men det var også flere andre alternativer, dette er beskrevet i 2.3.3.2.

#### *2.3.3.1 Dataoverføring, JSON vs. XML*

Når data skal overføres fra en webserver til en klient-maskin er det flere måter dette kan skje på. Data kan overføres ved at det skrives i et format vi selv bestemmer, som vi så behandler på klient-maskinen. Dette er en liknende problemstilling til den vi så i 2.2.1, der vi diskuterte mulighetene for å bygge kartprogramvare selv, samt hvilke fordeler og ulemper dette hadde i forhold til å bruke en ferdig utformet løsning.

Det ville teoretisk sett vært mulig å la webserveren ta seg av all prosesseringen, og så servere et ferdig "kart-bilde" til brukermaskinen, noe som ville ført til tap av all direkte manipulasjon av kartet. Det ville også ført til at alle små forandringer som brukeren gjorde ville krevd en ny runde med uthenting og prosessering av data. Likevel hadde dette hatt den fordelen at brukermaskinen hadde sluppet all arbeidsbyrde, og vårt program kunne ha blitt brukt på helt enkle enheter med et minimum av prosesseringskraft. Vårt produkt var imidlertid tiltenkt fullverdige datamaskiner som ikke burde ha noe problem med arbeidsbyrden. Det skal også nevnes at det potensielt kunne blitt mye arbeid for webserveren hvis denne skulle bearbeide og presentere data for mange brukere samtidig.

Den beste og mest utbredte løsningen på denne type problem er å overføre alle relevante data til brukeren, som så tar disse i bruk og får arbeidsbyrden på sin maskin. Denne overføringen skjer så i form av et utbredt og velprøvd dataformat, der hovedsakelig to valgmuligheter finnes, XML og JSON.

Disse er relativt jevnverdige med tanke på funksjonalitet, men forskjellen er at XML er betraktelig mer fleksibel, da det brukes i mange flere sammenhenger enn bare overføring av dataobjekter slik som fly. Dette kommer på bekostning av at det kreves en del ekstra "lim" i XML formatet for at alt skal være så tydelig som mulig. JSON på den annen side er mer spesifisert for vårt type bruksområde, og er således mer kompakt enn XML. Da vi har behov for å overføre potensielt enorme mengder data, er det selvsagt mer gunstig at det kan gjøres så kompakt som mulig.

En kortfattet sammenlikning av de to formatene kan ses i Eksempel 2.1: Sammenlikning av XML og JSON

**Eksempel 2.1: Sammenlikning av XML og JSON**

**XML: 102 tegn uten whitespace**

```
<document>
  <by navn="Oslo" befolkning="77591" />
  <by navn="Fredrikstad" befolkning="77591" />
</document>
```

**JSON: 88 tegn uten whitespace (14% reduksjon)**

```
{
  "byer": [
    {
      "navn": "Oslo",
      "befolkning": 634463
    },
    {
      "navn": "Fredrikstad",
      "befolkning": 77591
    }
  ]
}
```

Vi ser tydelig av dette eksempelet at XML bruker betraktelig mer plass (i antall tegn), da "by" må spesifiseres for hver enkelt by-objekt, mens JSON sier at "her kommer en liste med byer", og deretter listes alle by-objektene opp i rask rekkefølge. Merk også at den prosentvise forskjellen i antall tegn vil øke etterhvert som flere by-objekter legges til.

En annen problemstilling med XML er at det også generelt sett tar lenger tid å prosessere, dette er også en kritisk faktor da vi vil at vårt program skal være så responsivt og raskt som mulig. Det er også betraktelig enklere og raskere å behandle JSON formatet i JavaScript enn det er å arbeide med XML.

Til sist må det nevnes at det eksisterer mange andre løsninger på denne type problem, slik som kommaseparerte lister eller muligheter gjort tilgjengelige i HTML5. Det finnes for eksempel løsninger som er enda mer kompakte enn JSON. Vi valgte likevel å begrense oss til disse to, da det var disse vi hadde hatt mest erfaring med tidligere. Vi var også trygge på at disse ville dekke alle behovene i vårt produkt.

#### *2.3.3.2 Alternativer til PHP*

Når det gjelder skript som skal kjøre på en webserver, er en av de mest utbredte måtene å gjøre dette på PHP. PHP er et programmeringsspråk som vi alle har hatt erfaring med tidligere, da det har vært en sentral del av opplæringen i programmering ved HiØ. Likevel er det ikke enerådende, det finnes mange andre muligheter som dekker behovet vårt. Eksempler på disse er:

- ASP.NET

- Python (med Django-rammeverket)
- Ruby on Rails
- Java Server Pages (JSP)
- Node.js

Alle disse har fordeler og ulemper, men de er alle gyldige alternativer til PHP. Likevel var det tidlig klart at PHP var det vi alle hadde hatt mest erfaring med. Derfor hadde vi den problemstillingen at vi kunne gå for noe vi visste at ville fungere, og som alle hadde akseptabel erfaring med, men at det senere kunne vise seg at andre løsninger var bedre for vårt bruksområde. Kanskje vår type applikasjon hadde vært mest gunstig å skrive i Python?

Problemstillingen var imidlertid at hvis vi skulle gå for en av disse andre løsningene, måtte vi først ha satt oss grundigere inn i denne løsningen, og utviklingsprosessen kunne potensielt ha blitt utsatt betraktelig.

### 2.3.4 Prosessering av data, server vs. klient

Vårt sluttprodukt skal hente ut data fra en database og videresende dette til sluttbrukeren via en webserver. På veien vil vi behandle dataene, blant annet ved å gjøre spesifikke utvalg, filtrere ut kolonner, legge til nye kolonner, etc. Disse dataene kunne hovedsakelig bearbeides på 3 ulike steder:

- I databasen ved hjelp av SQL-spørringer
- På webserveren som henter ut og videresender dataene
- På klientmaskinen, maskinen til slutt-brukerne til vårt system

Da vi begynte på oppgaven kjørte databaseserveren og webserveren på skolens server, frigg. Likevel hadde vi mulighet for å flytte disse til andre maskiner hvis dette var gunstig for oss, dette detaljeres i 2.4. Likevel er prinsippene de samme for denne problemstillingen.

En databaseserver som for eksempel skal filtrere data gjør dette ekstremt effektivt i forhold til det vi kan få til på både maskinen til sluttbrukeren og webserveren. Men ikke alt er mulig å gjøre på en gunstig måte med SQL. Derfor måtte vi kanskje ty til mer "manuelle" løsninger på enten webserveren eller klienten der vi ikke fikk løst problemene med SQL. Her måtte lasten som eventuelt ville oppstå på webserveren ved stor pågang tas med i betraktning.

Til slutt måtte vi også tenke på at ikke alle data og muligheter var tilgjengelige i alle ledd i prosessen. På databasen var det for eksempel problematisk å gruppere data på en gunstig måte, slik at de ble lettere å ta i bruk for webserveren og klientmaskinen (mer om dette i 2.3.6). Merk at dette er på grunn av dårlig databaselayout, og ikke begrensninger i selve programvaren. Direkte forbindelse til databasen fra klientsiden utelukkes naturligvis av sikkerhetshensyn.

### 2.3.5 Analyse av data

Databasen inneholder en mengde informasjon om flytrafikken rundt Hessdalen. Problemet vi sto ovenfor var i hvor stor grad vi skulle ta på oss oppgaven å analysere disse dataene. Her er det i hovedsak snakk om to ulike typer analyse. For det første vil analyse av flytrafikken for å få en bedre forståelse av lysfenomenet være svært relevant for effektmålet, dette er detaljert i 2.3.5.1. En annen type analyse som også er relevant er den som gir en bedre forståelse for flytrafikken, og som gjør dataene i databasen lettere tilgjengelig. Dette er beskrevet i 2.3.5.2.

#### 2.3.5.1 Dataanalyse for effektmålet, forståelse av lysfenomenet

Formålet med oppgaven var, som nevnt i 1.4.1, ikke bare å detektere flytrafikk, men også potensielt å bruke disse dataene for å bedre forstå lysfenomenet i seg selv. Det var ikke utenkelig at dataene kunne gi oss et hint om hva dette lysfenomenet faktisk var. Derfor kunne en grundig analyse av disse dataene være av interesse. For eksempel kunne det være en direkte sammenheng mellom lavtflyvende fly over Hessdalen og detekterte lysfenomener?

Problemet med denne type analyse var at det først og fremst bygde på antakelser. Det var vanskelig å utarbeide funksjonalitet for generell analyse, og hvis man ønsket mer spesifikke typer analyse, var det uendelig mange metoder/teorier man kunne ha kommet med, som så måtte implementeres. På den annen side hadde vi en unik mulighet for å lage smarte løsninger mens vi utarbeidet vårt produkt. I ettertid ville det vært mer problematisk å implementere disse, uten først å sette seg godt inn i vårt system, og deretter utvide dette. Derfor hadde det vært gunstig hvis vi inkluderte relevante analysemetoder mens vi arbeidet med oppgaven.

Vi måtte altså ta en avgjørelse om hvilke typer analyse vi skulle inkludere i vårt produkt, og hvilke vi skulle overlate til manuell analyse, eller eventuelle utvidelser av systemet i fremtiden.

#### 2.3.5.2 Dataanalyse for resultatmålet, bedre oversikt over flytrafikken

Den andre typen analyse er den som gjør at brukere av vårt system lettere får den oversikten de trenger. Her er det også snakk om potensielt uendelige muligheter for analyse, men her er hensikten med analysen mer spesifik, vi ønsker å gjøre data lettere tilgjengelig for brukeren.

Problemet med så store datamengder som vår oppgave tar for seg, er at det er vanskelig å presentere disse dataene på en måte som gir brukeren oversikt over det som er interessant i en gitt situasjon.

Det er umulig for oss å imøtekomme alle mulige problemstillinger som fremtidige brukere kan ville løse ved hjelp av våre data, men det er visse ting som vi kan gjøre mye lettere for brukere ved å implementere relativt enkel funksjonalitet. For eksempel se på trender i dataene, hyppighet av fly, gjentagende fly-avganger og liknende.

En av de mest relevante formene for analyse i denne sammenhengen er noe som har blitt nevnt flere ganger tidligere, sannsynligheten for fly i et område.

Flydeteksjonssystemet har den svakheten at fly som ikke inneholder den senderen som systemet bruker for å hente inn relevante data vil være helt usynlige for flydeteksjonssystemet, og som følge vårt system. Derfor hadde det vært gunstig for sluttbrukeren å hvertfall få en generell oversikt over sannsynligheten for fly i et gitt område. Selv om vårt system viser at det ikke var flytrafikk i et område på det nøyaktige tidspunktet da et lysfenomen ble registrert, er det fortsatt fullt mulig at det var et fly der, men det bare manglet den påkrevde senderen som flydeteksjonssystemet bruker. Dette kan løses på flere måter.

En mulighet som hadde vært meget gunstig for oppdragsgiver og andre brukere av systemet, hadde vært hvis brukeren kunne plassere ut en markør, og deretter fått opp en prosentvis sannsynlighet for om det var et fly i dette område ved det angitte tidspunktet basert på trender i flydataene. Dette ville ha krevd relativt mye arbeid for å få til, imidlertid, derfor måtte vi forholde oss til hvor vanskelig

dette hadde vært å implementere, hvor nøyaktig og effektivt dette hadde vært, samt hvilke andre alternativer vi hadde.

Et direkte alternativ til dette som gir en mer generell oversikt, men som dekker et liknende behov, er en såkalt "heatmap" (se 3.2.4.1, spesifikt Figur 3.5: Eksempel på heatmap). Dette er et slags bilde som baserer seg på et sett med punkter plassert ut på et kart. Plotter man all flytrafikk (eller hva som helst annet) på et kart, vil man umiddelbart se områder med høy trafikk. Det vil være større ansamlinger av fly i noen regioner enn i andre. En heatmap gir en generell oversikt over dette ved å markere områder på kartet der intensiteten av objekter er størst med en farge, som så blir "svakere" jo mindre intensiteten blir. Ved hjelp av en heatmap kan man få en estimert sannsynlighet for om det finnes fly i et gitt område.

Forskjellen på å sette ut en markør slik nevnt over, og det å generere en heatmap, er at heatmap-en er så generell at den potensielt kan bli ubrukelig for å finne sannsynligheten for fly. Dette er blant annet fordi en heatmap gjerne vil ta med flydata for et stort tidsintervall, for eksempel all flytrafikk for en måned. Dette kan løses ved at dataene som brukes til heatmap-en kan spesifiseres nærmere av brukeren. For eksempel kunne man bruke all data for en måned, men bare for data mellom klokken 17.00 og 18.00 på søndager. Dette ville imidlertid tatt betraktelig lenger tid å implementere, da vi måtte ha tatt hensyn til alle potensielle ønsker en bruker kunne ha anngående heatmap-en.

Til sist må det også nevnes at denne type analyse vil kreve store mengder data, problemstillingen var så hvor analysen skulle finne sted, og hvordan den skulle foregå slik at dette ble gjort mest mulig effektivt. Selve heatmap kunne ha blitt generert både på webserveren og på klientmaskinen.

Den mest "økonomiske" måten å generere heatmaps på, ville være å gjøre dette på forhånd og ha mellomlagrede kart klare. Dette hadde vært meget kostnadseffektivt, da all prosessering kunne skje på forhånd, slik at ingen deler av systemet fikk noen signifikant arbeidsbyrde. Dette hadde imidlertid den svakheten at brukeren ikke lenger kunne spesifisere detaljer rundt hvordan heatmap-en ble generert. Tapet av fleksibilitet ville antakelig gjort funksjonen relativt unyttig, da det ville vært lang ventetid (opp til en måned/uke avhengig av konfigurasjon) og umulig å få heatmaps for noe annet enn hele måneder.

En annen problemstilling var også at det hele tiden blir lagret mer data av flydeteksjonssystemet. Derfor måtte en prosess kjøres regelmessig, som regnet ut heatmaps for den nye flytrafikken.

### 2.3.6 Gruppering av fly i en overflyvning

Behovet for å gruppere fly rundt en overflyvning ble umiddelbart klart for oss da tynning av fly i en overflyvning, og behandling av overflyvninger som enheter (spesielt på kartet) ble nødvendige. Hvilke muligheter vi hadde for dette er beskrevet i delkapitlene under.

For å spesifisere nøyaktig hvilken type gruppering det er snakk om, så mener vi altså at et fly som flyr i nærheten av Hessdalen vil etterlate seg flere "punkter" i databasen. Alle disse tilhører en logisk gruppe, en "overflyvning". Hvis dette flyet kommer tilbake senere vil dette representere en ny overflyvning som danner sin egen gruppe. Målet er så å bruke dataene og finne en måte å gruppere disse på, slik at en overflyvning kan behandles som *ett* objekt. Dette er nødvendig av flere grunner, hovedsakelig for å kunne tynne ut databasen, samt for å kunne behandle grupper når stier (paths) skal tegnes opp på kartet, osv (se 3.2.4.4).

Grupperinger basert på de eksisterende kolonnene er beskrevet i delkapitlene under, men det var også andre fremgangsmåter vi kunne ha gått for, disse skal vi først gi en oversikt over.

Problemet med gruppering oppsto hovedsakelig fordi dataene produsert av flydeteksjonssystemet er separate punkter for en vilkårlig overflyvning, uten noen direkte definerende kolonne som identifiserer **en** overflyvning. Slik beskrevet under var likevel dette mulig ved å bruke en kombinasjon av kolonner, men denne problemstillingen kunne vært helt avverget hvis flydeteksjonssystemet ble modifisert.

Flydeteksjonssystemet tar inn data for en overflyvning, og er til alle tider klar over hvilket fly det har kontakt med, derfor hadde det vært mulig for systemet å generere en separat kolonne (i form av et løpetall) som skilte på overflyvninger. Ved hjelp av dette ville det vært enkelt å gruppere alle fly som hadde det samme løpetallet i en overflyvning.

En annen mulighet ville vært å strukturert databasen på en slik måte at flere tabeller ble tatt i bruk. En tabell kunne inneholdt overflyvninger, og en annen kunne så inneholdt punktene som oppgjorde denne overflyvningen, og som så refererte til hvilken overflyvning de tilhørte. Dette ville krevd en identifiserende kolonne slik nevnt i forrige paragraf.

Slik nevnt i 2.3.6.3 er Regtime en meget relevant kolonne med tanke på gruppering, men det har den svakheten at det potensielt kan oppstå små variasjoner under lagring, som gjør at en overflyvning kan få opp til flere ulike regtime verdier. Dette kunne vært løst ved at denne kolonnen ikke ble bestemt av databasefunksjonen, NOW(), men isteden valgte det nåværende tidspunktet i flydeteksjonssystemet. Denne verdien kunne så bli satt for alle punkter til overflyvningen. Da ville det vært garantert at denne alltid var lik for alle punkter i en overflyvning.

Alle disse metodene hadde imidlertid en stor svakhet, det krevde at vi utførte fundamentale forandringer i flydeteksjonssystemet, som så måtte bli reflektert for alle data allerede logget til databasen. Dette ville vært umulig for de to første metodene, da dette ville krevd at data ble gruppert slik at de kunne tildeles en felles identifikator. Altså en gruppering for å oppnå gruppering.

#### *2.3.6.1 Gruppering ved hjelp av hexident og postime*

Den første og mest åpenbare løsningen var å se på en gitt hexident (en unik identifikator for et gitt fly, som kunne være lik for flere ulike overflyvninger), og dermed se på tidspunktene for denne hexidenten. For å forklare dette tar vi for oss et eksempel der en hexident ble registrert i databasen ved følgende tidspunkter:

- 3. mars 2014 klokken 14:43
- 3. mars 2014 klokken 14:45
- 3. mars 2014 klokken 14.50
- 7. mars 2014 klokken 9.45
- 7. mars 2014 klokken 9.46

Vi ser at det er logisk å gruppere disse slik at de 3 første punktene inneholder en overflyvning, deretter kommer de neste 2 punktene flere dager etter, slik at disse blir gruppert i sin egen overflyvning. Desverre er det en stor problemstilling med denne fremgangsmåten. Hvordan skal man sette grensene for om et fly tilhører samme overflyvning eller ikke? Vi ser på et nytt eksempel:

- 5. januar 2014 klokken 19:11
- 5. januar 2014 klokken 19:17
- 5. januar 2014 klokken 19.18
- 5. januar 2014 klokken 19.20

Spørsmålet er om dette er en overflyvning. Det er det vanskelig å vite. Kanskje mellomrommet på ca. 5 minutter fra det første punktet og til de 3 andre faktisk betyr at flyet nå har begynt på en ny overflyvning? Til å begynne med, tenkte vi på det slik at hvis punktene var så nær hverandre, så behandlet vi det likevel bare som en overflyvning, da dette var logisk. Dette ville imidlertid ha ført til problemet hvis flyet byttet FlightID mens dette skjedde. Tynningen av data ville også bli ujevn hvis to overflyvninger ble slått sammen, og deretter tynnet ut.

#### *2.3.6.2 Gruppering ved hjelp av FlightID*

FlightID fungerer som et slags rutenummer for et fly. Der Hexident er en unik identifikator på et fly, som kunne gå igjen for flere overflyvninger, så ville Flightid være unik og konstant for en overflyvning. Derfor ble det raskt klart at dette potensielt kunne brukes for å gruppere fly.

#### *2.3.6.3 Gruppering ved hjelp av hexident og regtime*

En annen mulighet for gruppering var å se på datafeltene hexident og regtime.

Først må det forklares at vårt syn på "regtime"-kolonnen i databasen var at dette var klokkeslettet data ble lagret til databasen, altså noe i nærheten av "postime" (tidspunktet som flyet sendte ut en melding om dens nåværende status). Det viste seg imidlertid at utover å være klokkeslettet som data ble lagret til databasen, så ble også all data for en gitt overflyvning lagret SAMTIDIG.

Systemet tok inn informasjon fra passerende fly kontinuerlig, og aggregerte det mens flyet var i luften. Etter at flyet hadde forlatt rekkevidden til sensoren ble så all data lagret til databasen samtidig. Dette betydde at regtime ville være lik for alle data i en overflyvning. Altså kunne vi tilsynelatende bruke regtime for å nøyaktig kunne gruppere data. Hexident ble brukt hvis to ulike fly ble registrert med samme regtime (de fløy ut av rekkevidden til sensoren samtidig).

## 2.4 Plassering og drifting av systemer

Vi hadde hovedsakelig tre ulike systemer å forholde oss til. Hvor disse skulle hostes (plasseres og driftes) var viktig for stabiliteten til vårt system, og hvordan det ville tåle å kjøre konstant i flere år fremover.

### 2.4.1 Database

Vi vurderte kort om vi skulle la databasen være hostet på skolens server Frigg, slik den var da oppgaven vår begynte, eller om det var mer gunstig å flytte den et annet sted på grunn av annen trafikk på Frigg.

### 2.4.2 Flydeteksjonssystem

Slik nevnt i 2.1 så kjørte flydeteksjonssystemet (sammen med lysdeteksjonssystemet) på en dedikert datamaskin på oppdragsgivers kontor. Denne løsningen var ikke helt optimal, spesielt med tanke på at denne maskinen fikk lite tilsyn. Derfor var det viktig å utforske eventuelle alternativer til dette.

Det var flere ting som måtte tas hensyn til når det gjaldt plasseringen av dette systemet, men hovedsakelig var det ustabiliteten til systemet som var den største faktoren. Et slikt system krever regelmessig tilsyn for å være sikker på at det fungerer som det skal, men det er få som vil ta på seg en slik oppgave. Derfor kunne det blant anne bli vanskelig å få dette driftet på skolens servere.

### 2.4.3 Webserver

Når det gjaldt webserveren var den største faktoren hvor mye minne som kunne allokeres på webserveren på en gang. Problemet var at store mengder data skulle overføres til sluttbrukeren, og dermed kunne grensene raskt sprenges. En stor del av prosesseringen gikk også på webserveren, derfor var det viktig å forsikre oss om at serveren som hostet denne hadde tilstrekkelig med ressurser for å utføre alle prosesseringer for flere samtidige klienter.

En annen problemstilling var at lysfenomenet i Hessdalen var, og er, av internasjonal interesse. Derfor kunne vårt system potensielt få perioder med stor pågang. Altså måte systemet ha en så stor grad av tilsyn og vedlikehold at dette ikke ville føre til problemer.

Webserveren er, i likhet med databasen, hostet på Frigg. Denne brukes i en rekke forskjellige sammenhenger, men vi har liten kontroll over den. Hvis det skulle oppstå problemer med at serveren ikke kunne allokere nok minne til programmet vårt, kunne dette bli vanskelig å løse.

Vi var altså i en situasjon der webserveren var ideell å drifte på frigg så lenge vi ikke nådde noen grenser. Derfor måtte vi utforme vår applikasjon med dette i tankene hvis vi bestemte oss for å plassere webserveren der.

## 2.5 Arbeidsmetode

Vårt prosjekt hadde som mål å produsere en webside som inneholdt mye funksjonalitet, og som til tider behandlet relativt store datamengder. Programmering var en stor del av prosjektet, og dette måtte det tas hensyn til med tanke på vår arbeidsmetode.

### 2.5.1 Versjonshåndtering

En av de største problemstillingene vi sto overfor med prosjektet var hvordan vi skulle håndtere backup og administrering av programkoden vi skrev, eller med andre ord *versjonshåndtering*. Idéen bak versjonshåndtering er at brukere skal kunne gjøre forandringer, og deretter "committe" disse (altså en navngitt forandring som legges inn i systemet). Slik at det er lett å holde oversikt over forandringer, og hva hver enkelt forandring innebærer. Dette inneholder også muligheten for å gå tilbake til en tidligere commit, altså å tilbakestille endringer hvis noe skulle være feil. Alle disse forandringene samles i et såkalt "repository" (en sentral lagringsplass som holder oversikt over alle forandringer, samt nåværende versjon av dokumentet, etc.).

Til prosjektet ble vi tildelt en brukerkonto på "Trac", som kunne brukes for å kommunisere med prosjektveileder, dokumentere prosjektet og holde oversikt over Issues. Trac inneholdt også et versjonshåndteringssystem, SVN. Dette var altså en mulighet for versjonshåndtering.

SVN er bygget på en slik måte at alt er sentralisert i et repository, eller en "ekte" sentral lagringsplass om du vil. Altså vil forandringer som vi gjør på prosjektet, når de committes, automatisk appliseres til det sentrale repository-et, for alle brukere. Dette kunne være problematisk, da det kanskje viste seg i

ettertid at en feil hadde forekommet, som så ville føre til at dette måtte tilbakestilles. Det var bedre hvis hvert enkelt gruppemedlem hadde sitt eget lokale repository, og deretter opplaste dette til det sentrale reposity når alt var klart. Hver bruker kunne så velge når sykronisering med dette hoved-repositoriet skulle skje.

En løsning foreslått av prosjektveileder var Git (GitHub, 2014), men dette hadde den problemstillingen at det var betraktelig mer komplisert og problematisk i bruk i forhold til hva vi krevde til vårt bruksområde. Da vi bare var tre som jobbet sammen, samt at vi unngikk å dele opp programmeringsarbeidet så langt dette lot seg gjøre (beskrevet i 2.5.2), så var denne type løsning relativt omfattende i forhold til våre behov.

Versjonshåndtering gjør at backup og reversering av forandringer alltid er lett tilgjengelig. Dette gjør også at alle involverte alltid har oversikt over hvilke forandringer som skjer i et prosjekt. Spørsmålet var så hvilket versjonshåndteringssystem som var mest gunstig for vår arbeidssituasjon.

### 2.5.2 Delegering av arbeidsoppgaver

Sluttproduktet i vårt prosjekt var **ett** sammenhengende system som omhandlet **en** enkelt webside. Dette førte til at en enhet i systemet til enhver tid var avhengig av alle andre enheter. Hvis for eksempel en forandring ble gjort på webserveren, ville dette mest sannsynlig få direkte følger for selve websiden også. Denne type løsning gjorde det problematisk å delegere ut større arbeidsoppgaver til ulike gruppemedlemmer, da disse delene til enhver tid måtte holdes samstemte. Små misforståelser kunne bety store utsettelser.

Vi måtte altså ta en avgjørelse på hvordan vi hadde tenkt til å arbeide, slik at vi kunne jobbe så effektivt som mulig gjennom hele prosjektperioden.

Den første muligheten var å prøve å dele opp programmeringsjobber til hver enkelt, og "sy" dette sammen etterhvert som ulike deler ble ferdig. Problemet var at misforståelser kunne føre til store utsettelser. En annen problemstilling var å teste systemet for feil, da det ofte var vanskelig å vite hvor eventuelle feil ligger før alle deler settes sammen, særlig dersom ingen har en komplett oversikt. Dette ville imidlertid bety rask fremgang i prosjektet, da utviklingen kunne foregå parallelt.

Vi kunne også arbeide hver for oss med ulike arbeidsoppgaver, men dette har den problemstillingen at enkeltmedlemmer kan miste helhetsperspektivet i løpet av arbeidet. Et annet problem er at det til tider kan være vanskelig å dele opp arbeidet slik at alle har noe å gjøre.

På den annen side, ville konstant gruppearbeid ført til mye bortkastet tid for andre gruppemedlemmer, da det ikke var utenkelig at man til tider måtte vente på at en løsning skulle implementeres. Dette har imidlertid den positive effekten at alle gruppemedlemmer alltid har oversikt over hele prosjektgangen, og problemer kan raskt løses da alle er samlet når de oppstår.

### 2.5.3 Loggføring av arbeid

Arbeidet foregikk fortløpende med programmering av produktet, rapportskriving og møter med oppdragsgiver og veileder. I løpet av denne tiden måtte vi ha en klart definert metode for å loggføre alle viktige hendelser slik at vi hadde dette tilgjengelig når vi skulle ta dette med i rapporten senere. Derfor var det viktig å tidlig finne en en måte å raskt og effektivt loggføre arbeid. Her var det uendelig mange måter å gå frem på, men i prinsippet var det likegyldig hvordan vi gikk frem, bare vi fikk med all relevant informasjon.

Det må også nevnes at vi selvsagt holdt regelmessige møter med tilhørende møtereferater, men det var hovedsakelig hendelsene utenfor disse som var problematiske å finne gode måter å loggføre.

En av de viktigste formene for loggføring i denne type prosjekt var *issue tracking*, altså en oversikt oversikt over arbeidsoppgaver som måtte utføres i programkoden, og resultatet av disse.

#### *2.5.3.1 Issue tracking*

Issuetracking er rett og slett en måte å holde oversikt over hva om skal implementeres i koden, og resultatet av slike forandringer. For eksempel kunne et gruppemedlem lage et issue for å gjøre det mulig å gi en beskjed til brukeren når store datamengder hentes ut. Dette kunne så implementeres og merkes som "løst".

Det var mange ulike løsninger for issue tracking. Vi kunne blant annet bruke løsningen innebygd i Trac, eller alternativt en liknende løsning innebygget i Bitbucket (nettstedet for prosjektstyring som hoster Mercurial, vår løsning på versjonering, se 3.5.1). Den største problemstillingen vi måtte tenke på var hvor brukervennlig løsningen var, og hvor lett det var å eksportere løsningen nær slutten av prosjektet, hvis dette eventuelt skulle inkluderes i hovedrapporten.

### 2.5.4 Programmeringsfilosofi

Det er hovedsakelig to hovedparadigmer å forholde seg til når det gjelder programmeringsfilosofier. Det er om koden skal utarbeides etter fossefall-prinsippet eller om den skal utarbeides iterativt. Fossefall-prinsippet vil si at hvert "nivå" av prosessen utarbeides for å være basis for det neste. Man går aldri tilbake til et tidligere nivå da hvert nivå utarbeides for å være endelig. For eksempel vil dette si at testing av systemet foregår i ett nivå, etter at systemet er fullt utarbeidet.

Alternativet er å arbeide utifra en iterativ prosess, der man ofte går tilbake og forandrer på funksjoner og metoder som allerede er ferdigstilt, hvis dette er gunstig for helhetsløsningen. Det vil si at i løpet av prosessen vil det alltid være rom for å gå tilbake og omarbeide deler av systemet, selv om det kan medføre at andre deler også må tilpasses dette.

I tillegg til dette er også generelle "kjøreregler" for programmering kjekt å få definert tidlig, hvis ulike medlemmer skal arbeide på det samme systemet. Det er imidlertid viktig at disse ikke blir for begrensende, da dette kan kjøre selve kodearbeidet vanskeligere enn hva som er hensiktsmessig.

# 3. Planlegging og utforming

I kapittel 2 presenterte vi en oversikt over problemstillinger som er relevant i vår oppgave. I dette kapittelet skal vi ta for oss de løsningene vi valgte og hvorfor, samt hvordan vi implementerer disse. Dette skaper så fundamentet for det fulle systemet vi skal levere, som blir beskrevet i sin helhet i kapittel 4. Merk at det er en direkte sammenheng mellom nummereringen av delkapitler i kapittel 2 og 3.

## 3.1 Fly- og Lysdeteksjonssystemet

Vår oppgave var å utarbeide en webside som presenterer dataene fra databasen satt opp av den tidligere prosjektgruppen. Likevel ble det spesifisert av oppdragsgiver at vi burde se nærmere på deres løsning, og forbedre systemet hvis vi så muligheter for dette. Vårt prosjekt bygget direkte på deres system, derfor var det helt kritisk at dette fungerte tilfredsstillende.

Slik nevnt i 2.1 var selve flydeteksjonen bare en liten del av deres system, og dette produserte data som ble lagret i en database driftet på skolens egen server. Så lenge det var data i databasen, kunne vi utarbeide vårt produkt, og dette ville ha fungert (kanskje med litt rare resultater) uansett hva som hendte med fly- og lysdeteksjonssystemene.

Diverse problemer med lys- og flydeteksjonssystemene gjorde at disse måtte oppgraderes, dette er detaljert i 3.1.1. Med dette som utgangspunkt bestemte vi oss for å ikke involvere oss i dette systemet annet enn de dataene som lå i databasen. Vi kunne så konsentrere oss om å utarbeide vårt eget produkt først og fremst, og hvis det skulle bli tid mot slutten av prosjektet kunne vi heller ta for oss dette systemet da.

Merk at dette var med forbehold om at databasen de hadde satt opp var tilfresstillende for behovene til vårt prosjekt. Eventuelle problemer som oppsto med deres system løste vi så heller ved å ta kontakt med den tidligere prosjektgruppen direkte, istedenfor å prøve å løse det selv. På denne måten fikk vi konsentrert oss om vårt prosjekt i første rekke, uten i tillegg å måtte bruke tid på å sette oss inn i deres system. Det ville imidlertid vise seg at forandringer måtte gjøres i deres system for at vi skulle kunne gjøre vår jobb, dette er beskrevet i 3.1.2.

### 3.1.1 Oppgradering av lys- og flydeteksjonssystem

Systemet viste seg å være relativt ustabilt på grunn av flere faktorer. Blant annet hadde systemet et problem med en minnelekasje. Det var også nødvendig med generell oppdatering av programvaren på maskinen. Vi besluttet oss imidlertid tidlig i prosjektet for å ikke ta på oss denne jobben, da vi prioriterte arbeid på vårt eget system. Disse problemstillingene ble senere løst av den tidligere prosjektgruppen, som tok på seg jobben med å oppdatere og forbedre systemene. De utførte dette parallelt med, men uavhengig av, vårt prosjekt.

Selv etter disse forandringene var systemet fortsatt relativt ustabilt. Dette var noe vi måtte ta hensyn til i arbeidet fremover.

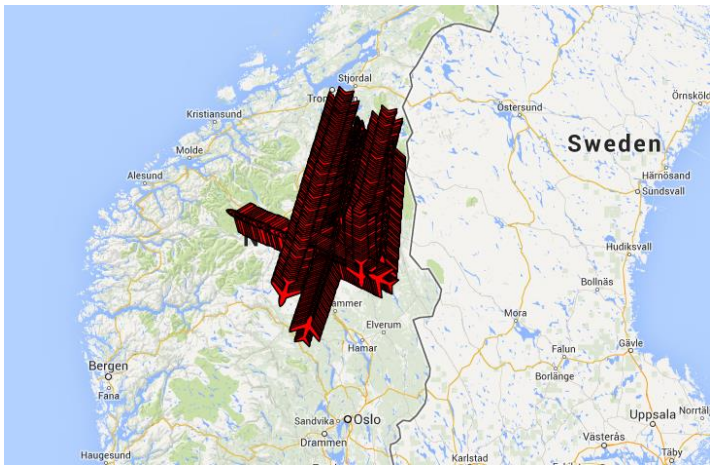
For det første kjørte hovedsakelig programmene på en Linux-maskin, men det viste seg nødvendig med en proprietær løsning som tolket meldingene sendt ut fra passerende fly. Uten dette var det umulig å dekode meldingene. Dette var et Windows-program, og derfor ble Windows satt opp på en virtuell maskin på Linux-maskinen. En svikt i et av leddene kunne føre til at hele systemet sluttet å fungere. Dette blir beskrevet nærmere i kapittel 4.

Med alle disse usikkerhetsmomentene tatt med i betraktning, konkluderte vi med at selv om systemet ble forbedret, så kunne vi ikke nødvendigvis stole på at det ville kjøre stabilt i årene fremover. Dermed tok vi forbehold om dette mens vi utarbeidet vår løsning, slik at eventuelle feil som kunne forekomme ikke ville få konsekvenser for vårt system.

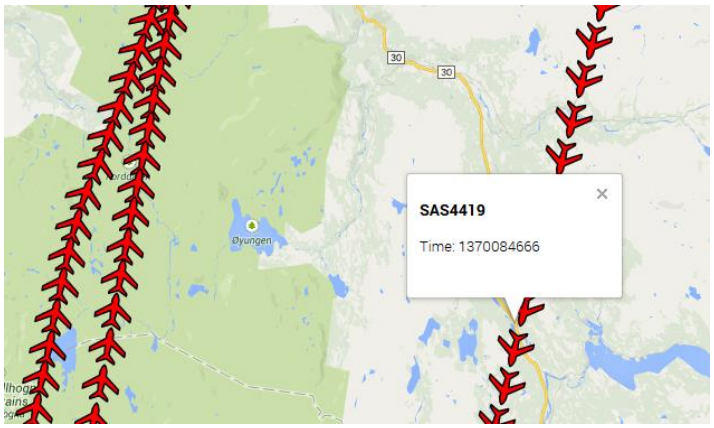
### 3.1.2 Samplingsfrekvens og datamengde

Et kritisk problem vi møtte på kort tid etter at vi begynte å hente ut data og kjøre tester på vårt system, var at samplingsfrekvensen til systemet var alt for stor. Det ble registrert mer data enn nødvendig for å representere en overflyvning. Flere steder var det over hundre punkter per overflyvning. Dette var alt for mye data, spesielt med tanke på at de fleste flyene bare beveget seg i rette linjer, med liten variasjon i dataverdiene. Problemet er illustrert i Figur 3.1 og Figur 3.2, som viser en tidlig utgave av systemet der hver observasjon har et eget flyikon.

**Figur 3.1: Datamengde**




**Figur 3.2: Punktdensitet**

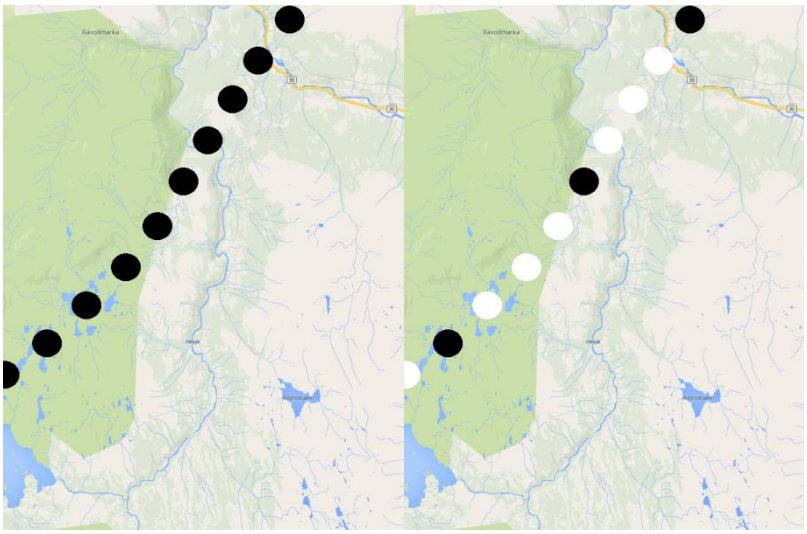


Måten vi valgte å løse problemet på var å gjøre forandringer i flydeteksjonssystemet, slik at dette begrenset hvor mange punkter som kunne bli registrert for et forbipasserende fly. Denne grensen satt vi til 25 (der vi tidligere hadde over 100 punkter for de fleste overflyvninger). Dette var den minste mulige grensen som systemet tillot uten å måtte skrives helt om. Denne forandringen måtte også reflekteres i de dataene som allerede hadde blitt logget til databasen, dette er dekket i 3.3.2.3.

Denne forandringen ble utarbeidet i samarbeid med den tidligere prosjektgruppen, da vi hadde valgt å ikke sette oss inn i deres system i større grad, slik nevnt tidligere. Forandringen er beskrevet i større detalj i kapittel 4.

Vi ønsket altså å oppnå en effekt som likner den representert i Figur 3.3, altså en jevn uttynning slik at vi står igjen med et mer gunstig antall punkter. Slik vi ser på den andre figuren er det fortsatt fullt mulig å se flyets bane. Merk at figuren ikke er basert på faktiske data, og svingningen er noe overdrevet.

Figur 3.3: Forbedret utvalg av punkter



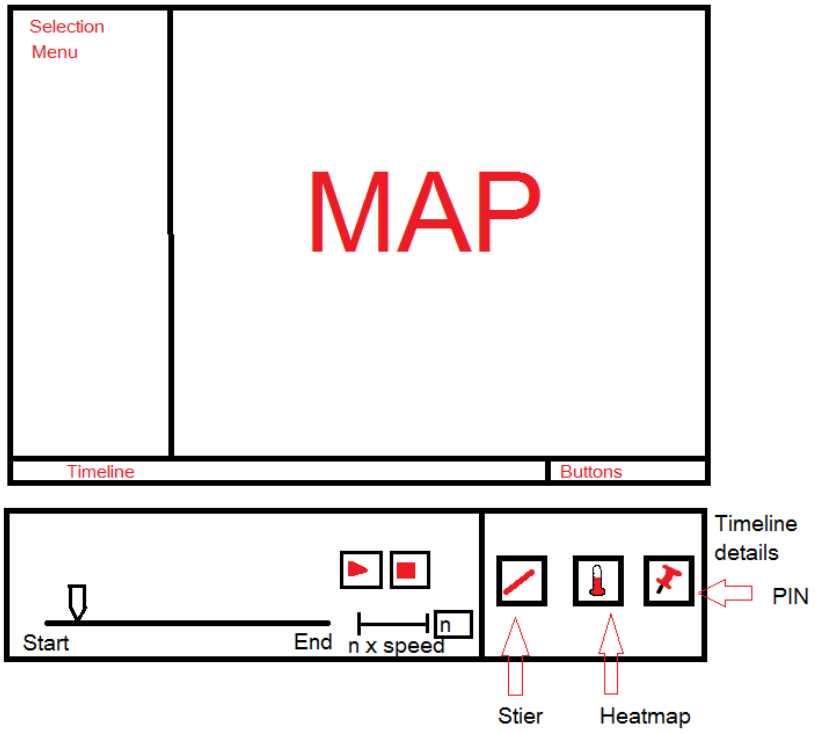
Merk at selv om databasen ble tynnet ut til å ha maksimalt 25 punkter, la vi i tillegg til funksjonalitet i vårt program som gjorde et enda tynner utvalg. Vi satt grensen til 10 punkter for hver overflyvning. Dette ga et tilstrekkelig bra bilde av overflyningen. Denne grensen var så tilgjengelig i konfigurasjonsfilen, slik at dette kunne justeres i ettertid (opp til 25 punkter).

Merk at alle (maks) 25 punkter likevel måtte overføres til webserveren. Deretter ble de tynnet ut og sendt videre til klientmaskinen.

## 3.2 Brukergrensesnittet

Brukergrensesnittet er den mest kritiske komponenten i vårt system. Dette måtte være utformet på en måte som ga brukeren best mulige verktøy og funksjoner for å kunne analysere flydataene så effektivt som mulig. Etter flere samtaler med arbeidsgiver for å få klarhet i hva som ble ønsket av systemet, samt hvilke metoder vi anså som mest gunstige for å produsere disse kravene, kom vi frem til en midlertidig skisse tidlig i prosjektet. Denne ga en oversikt over vår arbeidsplan som vi kom til å følge gjennom hele prosjektet, se Figur 3.4: Skisse av grensesnitt

Figur 3.4: Skisse av grensesnitt




Figuren gir en oversikt over generel layout og alle funksjoner vi ønsket inkludert i det ferdige produktet, detaljene om de ulike elementene er beskrevet i de påfølgende underkapitlene og kapittel 4, men først gir vi en kort oversikt:

Alt begynner med at brukeren velger et tidsrom ved hjelp av verktøyet øverst til venstre (ikke detaljert på denne figuren). Data hentes ut for det aktuelle tidsrommet og vise dette på kartet. Dette tidsrommet vil definere grensene for tidslinjen, vist nederst til venstre. Her kan brukeren velge ut et spesifikt "øyeblikk" i tid, som blir representert ved hjelp av ikoner på kartet.

Kartet inneholder hovedsakelig to ulike ikoner. Den første type ikon er datapunktene fra databasen, altså alle punktene som til sammen oppgjør en "overflyvning". Et fly på vei over Hessdalen ved et gitt tidspunkt. Disse kan skrus av eller på ved hjelp av knappen "Stier", som viser disse nodene samt en linje som knytter disse sammen.

Den andre type ikon er ikonet som representerer "øyeblikket" valgt av brukeren på tidslinjen. Dette vil altså plasseres på kartet på det stedet flyet befant seg ved dette tidspunktet, og data assosiert med ikonet vil også beskrive dette øyeblikksbildet av flyet.

Disse funksjonene vil gi brukeren mulighet for å se data for et ønsket **tidsrom**, samt oversikt over et spesifikt **tidspunkt** ved hjelp av tidslinjen. Dette er den grunnleggende funksjonaliteten brukeren trenger for å ta i bruk vårt system. De andre funksjonene, samt en mer detaljert beskrivelse av de funksjonene vi allerede har nevn her, finner du i delkapitlene under.

### 3.2.1 Valg av kartprogramvare

Etter å ha sammenliknet de ulike kartløsningene, kom vi frem til at Google Maps var det beste valget, hovedsakelig fordi Google Maps er utbredt og har mange brukere. En annen bonus er at Google Maps er godt dokumentert.

Google Maps hadde all funksjonalitet vi trengte. Det var enkelt å plassere og manipulere ikoner, kartet var detaljert nok for vårt behov (og merkbart bedre enn flere av konkurrentene), samt at zooming, panning og valg mellom satellitt- og kartbilder alle var inkludert direkte i kartet automatisk.

Det er selvfølgelig også en fordel at Google Maps er helt gratis.

### 3.2.2 Animasjon av fly

Animasjon var noe vi gjerne ønsket å få til. Dette ville gi brukeren en enkel oversikt over et hendelsesforløp med mange fly. Det viste seg imidlertid at oppdragsgiver ikke så på dette som viktig for sluttproduktet. Dermed valgte vi å ikke implementere dette, for heller å prioritere andre aspekter ved prosjektet.

### 3.2.3 Hvordan vise frem data fra databasen

Vi valgte å presentere flydataene hovedsakelig ved hjelp av popup-bobler/informasjonsvinduer. Disse dukker opp når man trykker på et flyikon eller en av nodene for overflyvningen (altså en flyobservasjon fra databasen). Dette var oppdragsgivers ønske etter å ha blitt presentert de ulike mulighetene.

Google Maps gjorde dette meget enkelt å legge inn, samt at informasjonsvinduer for flere fly uten problem kunne være oppe samtidig. De ville også følge flyikonene de tilhørte hvis disse ble flyttet på.

Vi introduserte en ny problemstilling når vi valgte å gjøre det på denne måten. Hvordan skulle brukeren få påvirke åpning og lukking av flere slike informasjonsvinduer, og hvilke verktøy vi skulle tilby for å gjøre dette lettvint? Dette er beskrevet i 3.2.4.3.

I tillegg til informasjonsvinduer valgte vi også å representere data om rotasjon direkte ved å rotere flyikonene på kartet. Posisjonen til flyene ble selvsagt representert ved plassering på kartet, og det valgte tidspunktet på tidslinjen ble tydelig fremstilt.

Tidslinjen oppgir altså tidspunktet da flyet meddelte sin status, slik beskrevet i 3.2.4.5. Idéen er at brukeren velger ut et tidspunkt på tidslinjen, og flyene vil bli plassert på riktig plass for å reflektere dette. Tidslinjen beskriver hele tiden "nåværende" tidspunkt og hvilke data som gjaldt på dette tidspunktet. Disse verdiene er interpolert ved hjelp av nærliggende noder, se 3.2.4.6.

Merk også at selv om diverse data blir representert ved hjelp av selve ikonene, er dataene også inkludert i informasjonsvinduene for å få dem i tekstlig format.

Resten av datafeltene gjøres om til et passende format og vises direkte i informasjonsvinduene.

### 3.2.4 Verktøy og hjelpefunksjoner

Det viktigste når det gjaldt å utforme grensesnittet var å bestemme hvilke verktøy som var mest hjelpsomme for oppdragsgiver og andre sluttbrukere når de skulle ta i bruk systemet . Målet med vårt system er at når et lysfenomen blir observert i Hessdalen, skal det være enkelt å gå inn i vårt system og finne flytrafikken for tidsrommet der dette forekom. Det skal også være enkelt å

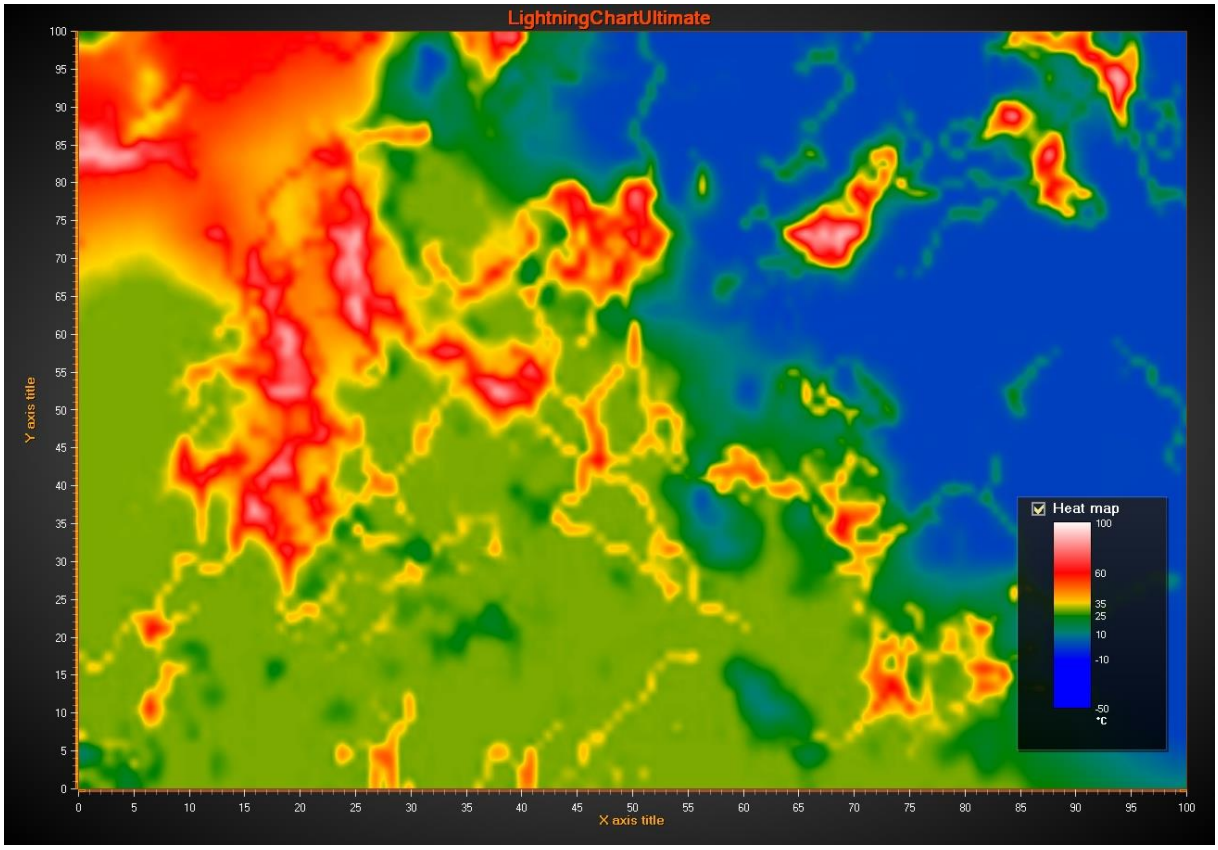
sammenlikne data "side-by-side" for å se om det detekterte lysfenomenet kunne ha vært et fly. For å oppnå dette måtte vi gi brukeren verktøy som viste den nødvendige informasjonen på en lett tilgjengelig måte, og helst også på en så detaljert måte at det var lett å ta en beslutning direkte på bakgrunn av dataene.

### *3.2.4.1 Sannsynlighet for fly (heatmap)*

Det kan være fly som mangler senderen som flydeteksjonssystemer baserer seg på. Dette gjør at slike fly vil være helt usynlige for slike systemer. På grunn av dette kunne det vært til stor hjelp for brukeren å se *sannsynligheten* for fly.

Løsningen vi endte opp med, var en såkalt "heatmap". Denne fargelegger kartet avhengig av forekomster, intensitet eller liknende – i dette tilfellet, den sammenlagte flytrafikken i en gitt periode. Områder der mange fly passerer vil ha en "varmere" farge enn områder der det går mindre trafikk. Et generelt eksempel på en heatmap kan ses i Figur 3.5. Dette gir brukeren en generell oversikt over sannsynligheten for fly i et større område.

**Figur 3.5: Eksempel på heatmap**




Måten vi implementerte dette på var ved å sette inn alle koordinatparene for de ulike flyene i et gitt tidsområdet i en heatmap-funksjon, som lager et gjennomsiktig lag og legger det på kartet. Fargen gir fort og intuitivt et fornuftig estimat på sannsynligheten for fly i et hvilket som helst område.

Hvis brukeren prøver å finne ut om det befant seg et fly på en bestemt plass på et bestemt tidspunkt, kan det skje at ingen fly er registrert i databasen for det aktuelle tidspunktet. Da kan brukeren velge å vise et heatmap med data fra for eksempel en uke tilbake i tid. Hvis man ser at området er sterkt

rødt (altså at det var høy intensitet av fly i det valgte intervallet), så kan man konkludere med at selv om ingen fly ble registrert, er det likevel relativt sannsynlig at det hadde vært et fly der.

Heatmap løsningen vi tok i bruk var en eksisterende funksjon fra `google.maps.visualization`. Denne tok imot en rekke punkter, og tegnet deretter en heatmap ut fra disse. Merk at denne løsningen ikke godtar linjer, man måtte oppgi punkter. Vi bestemte oss for å bruke punktene direkte fra databasen, uten å tilpasse disse ytterligere for å danne linjer (ved hjelp av for eksempel interpolasjon). Dette gjorde vi på grunnlaget at de valgte punktene allerede ga et tilstrekkelig godt bilde av flyets reise.

#### *3.2.4.2 Begrensning av tidsrom og område*

For det første var det klart at brukeren måtte få velge et ganske spesifikt tidsrom og geografisk område for dataene som skulle uthentes. Begrensning av området kunne brukeren gjøre manuelt ved å flytte og zoome på kartet, men i tillegg la vi til en grense som kunne konfigureres for systemet. Denne blir representert som et rødt rektangel på kartet, og angir en grense for området der flydata blir hentet fra databasen. Vi valgte å sette denne grensen såpass generell at alle flydata i systemet ville bli bli inkludert til å begynne med. Hvis oppdragsgiver så ønsket det kunne denne forandres i ettertid, for å begrense størrelsen på utvalget. Denne grensen fungerte også slik at brukeren ble opplyst om hvor det kunne forventes å finne data.

I tillegg til dette skulle brukeren få velge et tidspunkt (dato og klokkeslett) for når data skulle uthentes. For eksempel skulle alle fly mellom klokken 13.30 den 01.08.2013 og klokken 13.50 samme dag hentes ut og vises på kartet. Som et alternativ til dette la vi også til muligheten for brukeren å se på flydata for "siste time" og liknende, hovedsakelig for å teste systemet. Ved hjelp av dette ble det enkelt og intuitivt for brukeren å finne de dataene som var relevante.

#### *3.2.4.3 Informasjonsvinduer for fly*

Oppdragsgiver ba spesifikt om en knapp der det kunne velges om informasjonsvinduene skulle forbli på skjermen, eller om åpning av et nytt informasjonsvindu skulle lukke det forrige. På denne måten var det både mulig for brukeren å ha flere informasjonsvinduer åpne for å sammenlike informasjon, men også mulig å raskt gå over informasjon uten at skjermen ble for rotete, eller krevde manuell opprydding.

I utgangspunktet har man bare et informasjonsvindu oppe av gangen. Hvis man trykker på et nytt fly, blir forrige informasjonsvindu lukket men det nye åpnes. I "sticky"-modus er det mulig å ha flere informasjonsvinduer oppe samtidig, slik at man enkelt kan sammenlikne dataene for flere ulike fly. Dersom sticky-modus slås av igjen, skal alle informasjonsvinduer lukkes.

Vi måtte ta hensyn til at brukeren ved en feil kunne trykke på sticky-knappen etter å ha åpnet flere informasjonsvinduer. Dette kunne ført til at arbeid gikk tapt, og måtte gjøres på nytt. For å unngå dette inkluderte vi en dialog som advarte brukeren om at X informasjonsvinduer ville lukkes hvis brukeren valgte å skru av sticky-mode. Slik unngikk brukeren å tape arbeidstid ved et feilklikk.

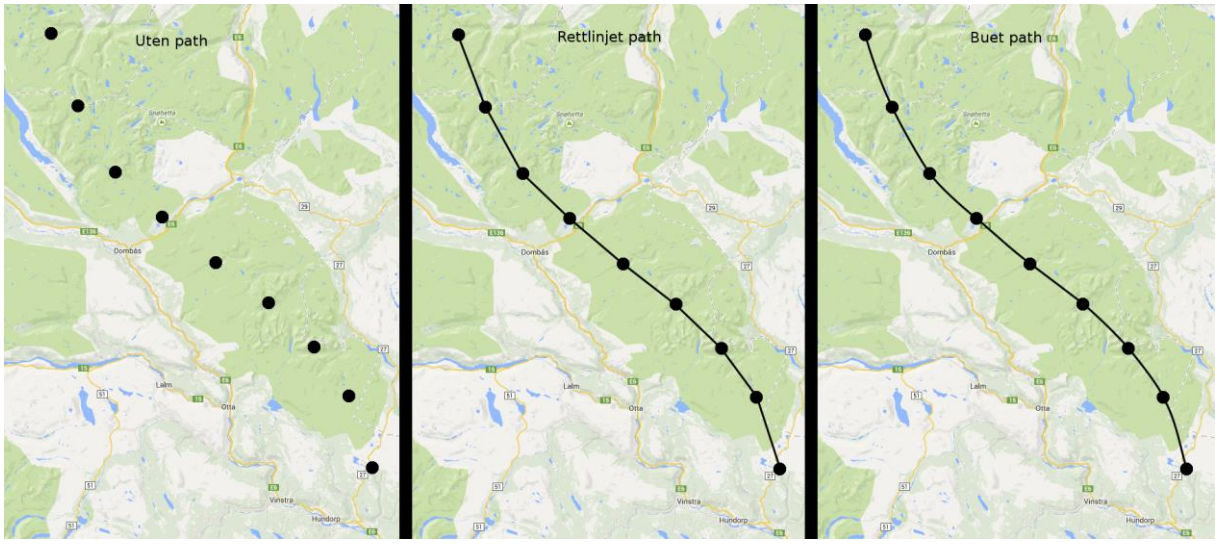
#### *3.2.4.4 Vise veien et fly tar (stier og observasjoner)*

Slik nevnt flere ganger tidligere var det snakk om relativt store datamengder. Et døgn kunne inneholde data om flere tusen punkter. Det var ikke utenkelig at en bruker hentet ut data for for eksempel en dag eller to, og dette ville ha ført til et relativt stort antall ikoner på kartet. Selvsagt

kunne brukeren "spole" frem og tilbake i tid (se 3.2.4.5) for å se banen til flyet, men dette var relativt tungvint.

Vi bestemte oss for å tegne en enkel, rett linje mellom hver loggførte posisjon i en overflyvning. Denne gir en mye klarere oversikt over hvor flyene kommer fra, og viser nesten nøyaktig hvor de passerer fra første til siste loggførte posisjon. Eventuelle avvik på grunn av små forandringer, svakt kurvede baner og liknende anses å være neglisjerbare. En sammenlikning ser du i Figur 3.6: Sammenliking, rettlinjet og buet path. Vi ser at forskjellen er minimal, selv i et konstruert eksempel med overdreven svingning.

**Figur 3.6: Sammenliking, rettlinjet og buet path**




For ordens skyld skal funksjonen kunne slås av dersom brukeren ønsker det.

Slik funksjonen var opprinnelig, ble hver enkelt observasjon også plottet på kartet. Dessverre er ikke Google Maps optimisert med tanke på større mengder "markers". Derfor ble det raskt problematisk å tegne alle disse på kartet, og det viste seg at en del av grunnen til dette var formatet på ikonet. Til å begynne med brukte vi det samme ikonet for hver observasjon som for selve flyet (som egentlig bare skulle representere hvor flyet befant seg i øyeblikket valgt ut på tidslinjen). Bildet er gjengitt i Figur 3.7.

**Figur 3.7: Vektorikon for fly**

Vektorgrafikk er et format der man oppgir en rekke koordinater som representerer hjørnene i bildet. Dette gjør at det er lett å transformere bildet i ettertid, for eksempel når det dreier seg om rotasjon, størrelse, farger, gjennomsiktighet og andre aspekter vi kan tenke oss å ha bruk for. Vektorgrafikk er også mye mindre i filstørrelse.

Problemet er imidlertid at denne fleksibiliteten har en pris – det blir betraktelig tyngre å prosessere disse enn et enkelt bilde. Dette var akseptabelt (og nødvendig) for selve flyikonet, men holdepunktene på stien til et fly kunne bruke betraktelig enklere ikoner, da vi ikke hadde bruk for all denne funksjonaliteten.

Til å begynne med prøvde vi å bruke enklere vektorgrafikk, for eksempel bare en enkel sirkel, men til og med dette var relativt krevende å prosessere når det ble mange observasjoner å vise samtidig. Den beste løsningen var å ta i bruk rastergrafikk. Vi ofret muligheten til å forandre ikonet dynamisk, men bildene ble til gjengjeld mye mindre krevende å vise, og ytelsen i applikasjonen ble merkbart bedre.

Ikonet vi valgte å bruke var en svart sirkel med et hvitt sentrum, dette er avbildet i Figur 3.8: Flysti. Dette ville mest sannsynlig alltid være lett synlig på kartet, og var derfor perfekt får vårt bruk. Etter å ha tatt i bruk dette ikonet hadde vi ikke lenger problemer med å tegne opp stier for fly.

For å gjøre det lettere for brukeren å skille ut enkelte fly på kartet valgte vi å gi unike farger til ulike overflyvninger. Flyet og dets tilhørende "path" ble gitt samme farge. Det var selvsagt fordelaktig hvis fargen tildelt et fly var så unik som mulig i forhold til andre fly på kartet. Vi valgte å løse dette ved å la en tilfeldig tallgenerator bestemme fargene for en overflyvning. Problemet med de fleste slike generatorer er at de ikke faktisk er tilfeldige, de følger visse trender. Dette ville vært for dårlig for vårt bruksområde. Det er mulig å oppnå større grad av "tilfeldighet" ved å gi tallgeneratoren et såkalt "seed", altså et tall som brukes som utgangspunkt for funksjonen. Så lenge seedet er unikt vil også tallene produsert av generatoret mest sannsynlig være meget ulike tallene produsert av en generator med et annet seed.

JavaScripts innebygde tilfeldig tallgenerator har ikke støtte for å spesifisere seed-verdi. Vi måtte altså ty til et eksternt bibliotek for å oppnå den funksjonaliteten vi trengte. Seedet vi ga til funksjonen var summen av flyets ID, rutenummer og registreringstidspunkt. Det var sterkt usannsynlig at to ulike overflyvninger ville produsere samme tall, dermed var dette et bra seed.

Deretter brukte vi så dette tallet for å bestemme de tre HSL-fargekomponentene til flyet, som vi til sist konverterte til RGB-farger. Denne fargen brukte vi så på både flyet og dets sti. Resultatet ser du i Figur 3.8: Flysti.

**Figur 3.8: Flysti**

Vi ser at flyikonet og stien har samme farge, samt at flyobservasjonspunktene er hvite sirkler med en svart ytterkant. Disse er godt synlige både i forhold til selve kartet og linjen for flyets sti.

Det skal også nevnes at tidsrommet spesifisert av brukeren kan være større enn intervallet for en gitt overflyvning. Vi måtte finne en løsning på dette problemet, slik at brukeren raskere kunne få detaljert (interpolert) informasjon om en spesifik overflyvning, uten å måtte lete rundt på tidslinjen. Hvordan vi løste dette er beskrevet i 3.2.4.7.

En liten detalj som må nevnes var størrelsen på ikonene. Vi bestemte oss for å bruke relativt små ikoner, slik at kartet skulle bli mer oversiktlig ved store samlinger av fly. Vi valgte imidlertid å utvide den klikkbare regionen, slik at man kunne trykke på ikonet selv om pekeren var plassert litt utenfor.

Hele funksjonaliteten for å vise noder for loggførte posisjoner, og selve stien som forbinder disse, ble gjort tilgjengelige for bruker å skru av og på, slik at bruker selv kunne tilpasse etter behov.

#### *3.2.4.5 Verktøy for manipulasjon av tid*

I 3.2.4.2 spesifiserte vi brukerens muligheter for å bestemme tidsrommet for datauthenting. Innenfor dette tidsrommet var det opprinnelig ønskelig å animeres disse i tid ved hjelp av "Play/Pause"-funksjonalitet. Altså hadde vi en tidslinje med markert tidspunkt for hva som ble vist akkurat "nå", et øyeblikksbilde. Muligheter for å justere tidspunktet frem og tilbake var lett tilgjengelig ved å trykke på ulike steder på tidslinjen eller ved å dra i markøren. Hvert "øyeblikk" ville så bestemme hvor flyikonet skulle plasseres på kartet, for å illustrere hvor flyet befant seg på dette tidspunktet.

Ved hjelp av interpolasjonen beskrevet i 3.2.4.6 var det mulig å hente ut data fra en "kontinuerlig" datamengde, i motsetning til de diskrete verdiene fra databasen. Uten dette hadde ikke tidslinjen vært like relevant, da ikonene og deres verdier hadde blitt meget "hakkete". I denne situasjonen hadde det vært like greit å bare vise alle tilgjengelige data til brukeren samtidig. Dette var selvsagt ikke mulig med kontinuerlig tidsutvalg, da det fantes potensielt uendelige mengder data (man kunne teoretisk sett hente ut data fra klokken 17:36:50.762 hvis dette var ønskelig, for eksempel).

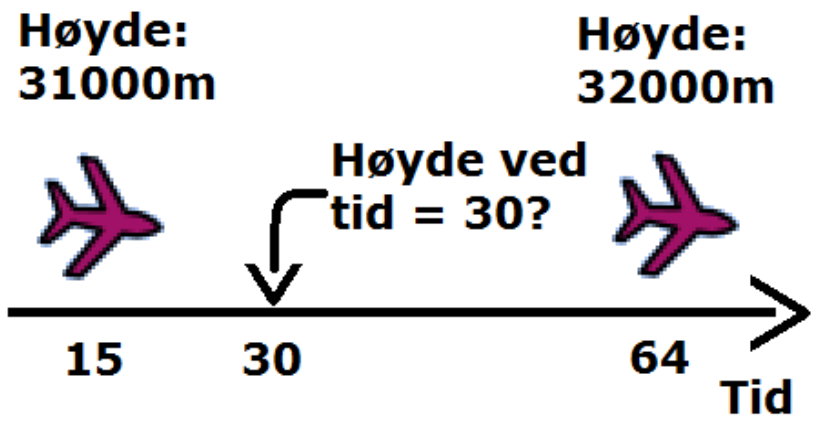
Med denne funksjonaliteten på plass var animasjon relativt enkelt å få til. Vi trengte bare å flytte det valgte "øyeblikket" på tidslinjen ved regelmessige intervaller. Det viste seg imidlertid at animasjon ikke var viktig for oppdragsgiver, derfor implementerte vi ikke dette. Selve tidslinjen kunne nå manipuleres manuelt, for å velge tidspunkt for interpolering, dette var godt nok for brukeropplevelsen. Animasjon hadde ikke tilført noe betydelig utover dette.

#### *3.2.4.6 LERP, lineær interpolasjon*

Interpolasjon går i hovedsak ut på at man vil finne ut en tredje verdi som ligger mellom to andre diskrete verdier. Dette er illustrert med et enkelt eksempel, se

Eksempel 3.1: Interpolasjonsoppgave.

Eksempel 3.1: Interpolasjonsoppgave




Problemstillingen er altså at det finnes to sett med data, og man ønsker å finne verdien på et punkt plassert mellom disse. Fremgangsmåten er som følger:

Hvis høyden går fra å være 31000m til 32000m vil det si at det skjer en forandring på 1000 meter. Denne forandringen foregår i et tidsrom på 64 - 15 = 49 tidsenheter (benevningen er irrelevant). Tiden der vi ønsker å finne høyden, 30, er 15 sekunder etter starttidspunktet.

Altså vil vi vite hva høyden er ut av totalt 1000 meter på et tidspunkt som er 15 ut av totalt 49 tidsenheter. Dette er triviell matematikk:

$$\frac{X}{1000} = \frac{15}{49} => X = \frac{15}{49} * 1000 = 306$$

Altså er flyet i en høyde av 31000 + 306 meter på det angitte tidspunktet, 31 306 meter.

Denne metoden kan så brukes for å interpolere alle de relevante dataene mellom to punkter (rader i databasen).

Merk at det her er snakk om lineær interpolasjon, det finnes andre mer avanserte metoder som også kan brukes for å få mer nøyaktig resultater, men dette gir tilstrekkelig nøyaktighet for vårt bruksområde.

Det skal nevnes at denne metoden potensielt kunne produsere feilaktige verdier, hovedsakelig hvis en av nodene som utregningene ble basert på inneholdt feil eller mangler. Alle slike verdier måtte altså tas med en klype salt. Likevel gjorde dette det betraktelig lettere for brukere av vårt system å finne ut om et potensielt lysfenomen var et fly eller ikke. Hvis området for et detektert lysfenomen var midt mellom to flyregistreringer, slapp nå brukeren å manuelt regne ut verdier for dette området. Nå ga vi brukeren verktøy for å automatisk få dette opp ved å bevege tidslinjen, slik at flyikonet ble plassert på det aktuelle området. Verdiene som så ble fremstilt var en tilnærming til hva de ville ha vært hvis flydeteksjonssystemet hadde logget en verdi for dette øyeblikket.

#### *3.2.4.7 Snarveier og praktiske funksjoner*

Etter at de grunnleggende verktøyene var på plass, viste testing raskt at programmet kunne bli litt tungvindt å bruke. Ved å legge til noen mindre praktiske funksjoner, ble sluttbrukers opplevelse

straks mer behagelig. Disse viste seg nødvendige hovedsakelig på grunn av de store tidsrommene som kunne hentes ut. Tidslinjen ble vanskelig å navigere med hvis denne omhandlet et stort tidsintervall.

For det første gjorde vi det mulig for sluttbruker å begrense det uthentede tidsrommet til varigheten til en enkelt overflyvning. For eksempel kunne bruker hente ut data for flere timer, for å få en overordnet oversikt. Deretter kunne det vise seg at det ikke var mer enn *én* relevant overflyvning i hele utvalget. Denne funksjonaliteten gjorde det så lett for bruker å velge denne overflyvningen, og gjøre et nytt utvalg som bare inneholdt tidsrommet for denne, slik at uninteressant data ble filtrert bort. Etter en slik filtrering ble også tidslinjen lettere å bruke, da den arbeidet på et mye mindre tidsrom.

En annen funksjon vi la til, som hovedsakelig var for å gjøre navigasjon på tidslinjen enklere, var å la brukeren hoppe til ønskede tidspunkt ved hjelp av kartet. Alle observasjons-noder på kartet ble gitt en "hopp til tidspunkt"-funksjon. Dette gjorde at tidslinjen ble satt til det tidspunktet for den valgte observasjonen. Ved hjelp av dette ble det lettere for brukeren å "finne" en overflyvning på tidslinjen, for å få frem flyikonet for denne. Deretter kunne ikonet beveges frem og tilbake på overflyvningen for å få de interpolerte verdiene.

Etter å ha holdt en demo for oppdragsgiver viste det seg også at systemet ikke var helt intuitivt, spesielt med tanke på heatmap og tidslinjen. Derfor la vi vil "hjelp"-ikoner ved disse, som forklarte funksjonaliteten til programmet.

## 3.3 Database og backend

Databasen med flydata var helt kritisk for vårt prosjekt, da alt vi foretok oss baserte oss på disse dataene. Det var viktig for oss at dataene var til å stole på, og at vi fullt ut forsto betydningen av alle kolonnene (datafeltene). Under har du en liste av alle kolonnene i databasen, eksempler på data de kan inneholde og hva disse dataene faktisk betyr. Hver rad i databasen omhandler ett "øyeblikksbilde" av *ett* fly (en flyobservasjon), og inneholder alle disse datakolonnene.

Kolonner:

- **Hexident:** 06A052
    - Dette er ID'en på den fysiske boksen som sender ut disse datameldingene, som så blir plukket opp av flydeteksjonssystemet.
- **Postime:** 1370085774
    - Dette er tidspunktet for når denne meldingen ble sendt ut, og derfor også tidspunktet for når dette flyet hadde disse dataene.
- **Flightid:** QTR991
    - Dette er ID'en på ruten som et fly tar. Denne er for det meste unik og identifiserende på samme måte som hexident, men ikke stabil nok til at denne kan brukes på noen gunstig måte.
- **Latpos:** 61.51859000
    - Dette er breddegraden flyet var på ved tidspunktet gitt av postime.
- **Longpos:** 11.16329000

    - Dette er lengdegraden.
- **Track:** 295
    - Dette er rotasjonen til flyet (ut av 360 grader).
- **Speed:** 483
    - Dette er hastigheten til flyet.
- **Altitude:** 32000
    - Dette er høyden over havet som flyet befinner seg på for øyeblikket.
- **Verticalrate:** 64
    - Dette er hvor raskt flyet stiger (0 betyr at flyet flyr rett frem uten å stige eller synke).
- **Incam:** 1
    - Denne kolonnen tilsier om flyet befant seg i kameraet i Hessdalen ved dette tidspunktet.
    - Dette regnes ut av systemet til den tidligere prosjektgruppen.
- **Regtime:** 2013-06-01 13:35:16
    - Dette angir tidspunktet for når dataene ble logget til databasen. Merk at alle punktene for en enkelt overflyvning lastes opp til databasen på samme tid, slik at alle får samme regtime.

En mer grundig beskrivelse av databasen finner du i Kapittel 4 Implementasjon.

Hvordan vi løste de ulike problemstillingene med tanke på databasen, og hvordan dataene skulle behandles i vårt system, er beskrevet i de neste delkapitlene.

### 3.3.1 Relevante kolonner

Hvilke kolonner vi tok i bruk ble hovedsakelig bestemt av arbeidsgiver. Generelt sett var det ønskelig at så mye data som mulig ble tatt med. De eneste kolonnene som var relativt uinteressante var "incam" og "regtime".

Slik nevnt i databasebeskrivelsen i 3.3 var "incam" datafeltet som anga om flyet var innenfor området der det ble fanget opp av kameraet plassert ved Hessdalen AMS. Dette ble regnet ut ved å se på om flyets koordinater var innenfor et sett med statiske verdier. Dette var litt problematisk da verdiene som ble brukt som grenser bare var approksimasjoner. De tok heller ikke hensyn til flyets vinkel fra bakkenivå, vær og avstand fra kameraet. Oppdragsgiver var enig i at dette ikke var spesielt brukbart, og vi bestemte oss så for å ikke ta med denne kolonnen i datafremvisningen. Dette kunne ha gitt brukere et feilaktig inntrykk av at de for eksempel skulle kunne se et fly i kameraet, når det i virkeligheten var plassert høyt over kameraet, for eksempel.

Når det gjaldt "regtime" så var dette et datafelt som tilsa når dataene ble registrert i databasen. Dette var ikke av interesse for oppdragsgiver eller sluttbruker. Det var bare et datafelt som ble brukt av systemet. Vi tok likevel disse dataene med i dataoverføringen, men det ble ikke tilgjengeliggjort for sluttbrukeren. Grunnen til at vi tok det med var at regtime viste seg å være kritisk for å kunne gruppere punkter på en overflyvning, se 3.3.6.

Alle andre kolonner ble tatt med, men det skal nevnes at flere ble gitt nye navn slik at det skulle være mer forståelige for sluttbruker. Dette er beskrevet i kapittel 4.

### 3.3.2 Feil og mangler i data

Dataene som plukkes opp av flydeteksjonssystemet kan inneholde feil. Det skjer også at systemet ikke klarer å identifisere en gitt verdi, og derfor setter disse til å være en standard-verdi. Dette er 0 for tall-verdier og en tom streng ("") for tekst-baserte verdier. Vi måtte ta hensyn til dette i vår løsning, da spesielt 0-verdier kunne være problematiske å tolke.

Da vi innførte geografiske avgrensninger i systemet (slik beskrevet i 3.2.4.2), ble dette problemet med feil i koordinatdata løst. Alle koordinater utenfor de spesifiserte grensene ble filtrert bort ved uthenting.

Når det gjaldt de andre kolonnene, bestemte vi oss for å ikke "reparere" disse dataene. Dette var hovedsakelig fordi vi ikke hadde noen gunstig måte å løse dette på. Hvis vi hadde for eksempel interpolert verdier fra andre flyobservasjoner på overflyvningen ville dette heller gitt et "falskt" bilde av flyet. Da var det bedre å bare la feilene i data være. Brukeren ville selv se når data var mangelfulle, slik som fly med hastighet "0".

Det skal også nevnes at visse kolonner kunne ha unøyaktigheter. For eksempel kunne det skje at Regtime (altså tidspunktet der hele overflyvningen logges til databasen) ble forskjellig for 2 ulike punkter i den samme overflyvningen. Dette skjedde hvis systemklokken akkurat hadde "tikket over" til et nytt sekund midt mellom innsettingen av to rader i databasen. Vi løste dette problemet enkelt ved å bestemme at to nærliggende regtime verdier for samme overflyvning ble satt til den samme verdien. Dette er beskrevet nærmere i 3.3.6.

#### *3.3.2.1 Uthenting av for store data*

En problemstilling som var uungåelig når brukeren kunne spesifisere tidsintervall for uthenting, var at brukeren kunne velge å hente ut enorme mengder data. Brukeren ville uansett ha potensiale til å sprenge grensene, selv om vi reduserte mengdene data (se 3.3.2.2). Det var hovedsakelig på webserveren at dette problemet presenterte seg. Eventuelle problemer med datamengde på klientmaskinen ville bare bli begrenset av maskinvaren, noe brukeren selv hadde ansvar for.

En meget fleksibel løsning vi først prøvde å implementere, var at systemet automatisk skulle forsøke å hente ut data, og detektere problemer med plassmangel. Hvis problemet oppsto ville så systemet prøve igjen, men denne gangen med bare halvparten av dataene i første omgang, deretter siste halvdel for seg. Dette kunne så bli gjort rekursivt helt til en akseptabel datamengde ble funnet. Desverre hadde dette en stor problemstilling som viste seg å være uløselig. Når webserveren oppdager at grensen for minneallokering er nådd, blir en uhåndterbar feilmelding sendt fra systemet. Programmet blir terminert, og det er umulig å berge situasjonen. Vi måtte defor se på alternative løsninger.

Etter å ha utforsket mulighetene fant vi en betraktelig bedre løsning, som løste alle problemene vi hadde med minneallokering. Metoden vi hadde brukt tidligere for å hente ut data, var å samle alle radene fra tabellen på webservere før de så ble sendt videre til klient i en stor "pakke". Dette var fordi vi ikke var klar over muligheten for å skrive alle rader fra databasen direkte til output-strømmen til klienten, etterhvert som de ble uthentet. Vi unngikk altså at data ble aggregert på serveren, ved at de ble forløpende hentet ut fra databasen, prosessert, og så videresendt til klienten. Detaljene rundt hvordan dette foregikk blir forklart nærmere i 3.3.3 og 3.3.4.

#### 3.3.2.2 Problemer med datamengde

I systemet kunne det oppstå situasjoner der vi hadde for mye eller for lite data til å få en gunstig fremvisning i vårt system. Hvis vi for eksempel bare hadde **ett** punkt for en hel overflyvning, ville disse dataene bli vanskelige å tolke, spesielt hvis de inneholdt feil. Vi bestemte oss likevel for å ta med alle slike enkelt-punkter. Grunnen til dette var at effektmålet for vårt system i bunn og grunn var å bedre forstå lysfenomenene som forekommer i Hessdalen. Med dette klart for oss sa det seg selv at å filtrere bort data som på overflaten så mangelfulle ut, kunne være et enormt feilsteg. Det var ikke helt utenkelig at det kunne være en sammenheng mellom lysfenomenet og forekomster av datamangel. Derfor arbeidet vi med den filosofien at vi tok med "så mye data som mulig", så sant det lot seg gjøre å presentere på kartet.

Vi valgte imidlertid å gi brukeren direkte tilbakemelding på helt tomme utvalg, da dette kunne tyde på at systemet hadde vært offline i denne perioden. Brukeren fikk også en liste med datoer der vi var sikre på at systemet ikke ville inneholde data. Disse var datoer vi kom frem til ved å manuelt analysere dataene i databasen. Disse tidsperiodene var som følger:

- 19.05.13 til 22.05.13
- 10.06.13 til 08.08.13
- 18.08.13 til 13.01.14
- 16.01.14 til 24.01.14
- 27.01.14 til 20.02.14

Unntaket til denne filosofien var situasjoner der vi hadde for mye data, at det var så mye data å prosessere at systembegrensninger hindret oss i å fremvise det. Dette kunne få konsekvenser for både hvor store intervaller som kunne velges ut, samt om det i det hele tatt var mulig å bruke dataene til dataanalyse (krever data fra store tidsrom for å gi brukbare resultater).

Problemet med for store datamengder oppsto ved en av de første systemtestene vi utførte. En tidsperiode på en time kunne gi mange hundre rader av flyobservasjoner. Problemet oppsto på webserveren, som måtte allokere plass til alle disse dataene før de ble sendt videre til klienten. Det fantes en grense for hvor mye data man kunne allokere, og denne ble nådd raskt. Det var mange mulige løsninger på dette problemet, og vi endte opp med å implementere flere av disse i vårt system.

For det første kunne vi ha flyttet webserveren over til en annen maskin for å ha bedre kontroll på slike grenser, eller spøre driftsansvarlig om å øke grensene på webserveren, men mer om dette i 3.4.

En annen mulighet var å hente ut dataene i "puljer". Altså, man hentet ut litt data av gangen, prosesserte det og sendte det videre til klienten, uten at data ble aggregert på serveren. Det viste seg at dette ville bli helt nødvendig, også for andre problemstillinger enn minneallokering. Hvordan vi implementerte dette er forklart i 3.3.3.3.

Selv med disse løsningen var problemet fortsatt et faktum: det var for mye data per overflyvning. Mengdene var så store at det ble nødvendig å redusere samplingsfrekvensen til systemet. Dette ble detaljert i 3.1.2.

Neste steg var så å få reflektert denne forandringen i dataene som allerede hadde blitt logget til databasen før denne systemmodifikasjonen hadde blitt utført. Her oppsto nødvendigheten av å

gruppere fly, som vi dekker i 3.3.6. Etter at muligheten for å gruppere fly var på plass, var det relativt simpelt å tynne ut dataene i databasen.

#### *3.3.2.3 Tynning av databasen*

Behovet for å tynne ut databasen var et faktum etter noen av de tidligste testene vi utførte. Databasen inneholdt unødvendig store mengder data, så store at det var problematisk for oss å ta dem i bruk. Slik beskrevet i 3.1.2 fikk vi hjelp av den tidligere prosjektgruppen til å forandre på samplingsfrekvensen til systemet, slik at hver overflyvning fikk et mer gunstig antall punkter.

Dette skjedde imidlertid etter at systemet allerede hadde logget data i ca. et halvt år, noe som betydde at disse dataene måtte bli tynnet for å reflektere forandringen i samplingsfrekvens. Fremgangsmåten for hvordan vi løste dette er beskrevet under.

Den generelle idéen for hvordan uttynningen foregår er relativt simpel (se Figur 3.3: Forbedret utvalg av punkter, i begynnelsen av kapittel 3). Hvis man for eksempel har 100 punkter som oppgjør banen til en overflyvning, men man vil ha dette tynnet ut slik at man heller får 25 punkter til sammen, er dette en relativt triviell jobb. $\frac{100}{25} = 4$, altså skal vi ta med hvert fjerde punkt. Likevel var det ikke så rett-frem å utføre en slik forandring.

Første store problemstilling var hvor denne forandringen skulle utføres. Skulle vi lage et SQL script som utførte dette? I så fall hadde vi et problem med hvordan man skulle klare å gruppere punkter til en overflyvning med SQL, og deretter klare å iterere over alle disse punktene. Vi bestemte oss for at dette kom til å bli unødvendig komplisert, og heller ikke spesielt gunstig da dette var en engangs-operasjon. Metoden vi valgte å gå for var heller å lage et php-script som hentet ut data fra databasen, bestemte seg for hva som måtte slettes, og så produserte en .sql-fil som inneholdt DELETE-statements (spesifiserer hva som skal slettes i databasen) for dette. En DELETE-statement per flyobservasjon/rad som skulle slettes. Deretter kunne så denne filen eksekveres, og databasen ble tynnet ut.

Da vi valgte å gå for et php-script ga dette oss mye større fleksibilitet med tanke på logikken som bestemte hva som skulle slettes. Derfor valgte vi å gå for en relativt sofistikert uttynningsalgoritme.

Prinsippet var det samme som før, man tok vare på (for eksempel) hver fjerde rad, eller med andre ord, sletter 3 rader, så hoppes en rad over, deretter sletter man 3 til, osv. Men noe vi gjerne ønsket å få til var å alltid ta vare på første og siste rad, da dette garanterte en bra beskrivelse av flyets bane. Problemstillingen oppstår når man både vil få til dette, samtidig som man garanterer at det aldri blir mer enn 25 rader igjen for en overflyning etter sletting. De valgte punktene skulle selvsagt også være jevnt distribuert (og ikke bare for eksempel de 25 første).

Uten å gå for nøye til verks (da dette blir forklart i større detalj i kapittel 4), så kan dette greit forklares med et eksempel:

Hvis man har 200 rader som oppgjør en overflyvning, og man bare vil ivareta 4 av disse punktene, kan dette gjøres slik: $\frac{200}{4} = 50$. Problemet er at dette ikke vil gi en jevn distribusjon. Hvis man imidlertid gjør det slik: $\frac{200}{4-1} = 66.67$, altså at punktene 1, 66, 132 og 198 ivaretas, ser vi at dette er jevnt fordelt utover området. Vi valgte deretter å bestemme at sist punkt alltid skulle med, vi bytter

ut 198 med 200. Vi ivaretar disse 4 punktene, og sletter alle andre. Dermed får vi et jevnt utvalg fra hele intervallet.

Til sist skal det nevnes at grensetallet 25 (eller 4 i eksempelet over) er en **øvre** grense for antall punkter som tas med. Hvis en overflyvning hadde mindre enn 25 punkter, ville dette ført til at alle punkter ble ivaretatt. Det viktigste var bare å begrense antallet slik at det ikke ble problematisk å jobbe med datamengden. Mindre enn 25 punkter ville ikke være et problem for systemet.

For den fulle beskrivelsen se kapittel 4.

### 3.3.3 Dataflyt

Den komplette beskrivelsen av datastrømmen vil beskrives mer i detalj i kapittel 4, men det er endel hovedpunkter anngående hvordan dette foregår som må på plass først.

#### *3.3.3.1 Dataoverføring, JSON vs. XML*

Data, store mengder data, måtte sendes mellom webserver og klientmaskin, dette var uungåelig. Problemet var å finne et passende format. Et format som både innfridde våre behov og var effektivt nok til å kunne behandle de datamengdene vi jobbet med. Det var uakseptabelt hvis responstid og brukervennlighet ble redusert på grunn av prosesserings- og overføringstid.

Det ble klart tidlig at vi hovedsakelig hadde to alternativer, JSON og XML. Det var teknisk sett også mulig for oss å utforme vårt eget format, men disse var såpass optimale at vi mest sannsynlig ikke ville ha klart å utforme et bedre alternativ selv.

Hovedforskjellen på disse er at JSON er mer kompakt, mens XML gir større fleksibilitet. Spørsmålet var om vi trengte denne ekstra fleksibiliteten i vårt system. Svaret var kort og godt *nei*. De dataene vi trengte å overføre var i all hovedsak flyobservasjonsobjekter, et dataobjekt som inneholdt et predefinert sett med datafelter. Disse forandret seg aldri fra objekt til objekt, og hadde også verdier av samme datatype. Det var nettopp dette JSON var laget for. XML hadde vært relevant hvis de ulike attributtene for et dataobjekt kunne variere mellom ulike objekter, men dette var ingen faktor i vår situasjon. Altså gikk vi for JSON da det ikke bare var det optimale valget med tanke på "plassbruk", men også fordi det var betraktelig enklere å behandle JSON formatet i JavaScript enn det var å arbeide med XML.

#### *3.3.3.2 Alternativer til PHP*

Det var mange mulig alternativer vi kunne bruke istedenfor PHP, som var det mest åpenbare valget når det gjaldt skriptspråk for webservere. Vår problemstilling var at vi ikke hadde spesielt mye erfaring med de andre alternativene. Det ville også ha tatt tid både å sette oss inn i de andre alternativene for å se om de potensielt hadde noen fordeler fremfor PHP, samt tid for å forstå språket på et såpass høyt nivå at vi kunne skrive effektiv kode i det.

Vi bestemte oss derfor for at de potensielle vinstene med alternativer til PHP var såpass usikre, at den totale tidsbruken antakelig ikke ville være verdt det. Dessuten hadde alle gruppemedlemmer erfaring med PHP fra tidligere, derfor var vi relativt trygge på mulighetene vi hadde med språket. Da ingen umiddelbare feil eller mangler meldte seg, gikk vi for PHP uten å undersøke konkurrentene i større detalj.

#### *3.3.3.3 Uthenting av data med kontinuerlig printing av JSON*

Slik beskrevet i 3.3.3.1 var JSON var det mest gunstige valget for vårt bruksområde. Likevel kunne det oppstå problemer med datamengder, slik beskrevet i 3.3.2.2. Det ble aggregert data på web-serveren mens det ble hentet fra databasen, deretter ble alt konvertert til JSON-format, som så ble sendt til klienten. Dette lot seg ikke gjøre med store mengder data, da webserveren ikke tillot å allokere nok minne.

Løsningen vi kom frem til, relativt sent i prosjektet, var å kontinuerlig konvertere de uthentede dataene til JSON mens uthentingen fra databasen foregikk. Data ble hentet fortløpende fra databasen og gruppert i overflyninger (se 3.3.6). Dette ble bearbeidet, umiddelbart konvertert til JSON og sendt til klienten. Før neste overflyvning ble behandlet var så disse dataene friet opp, slik at det aldri ble allokert store mengder data. Vi unngikk så en av de største og mest kritiske problemstillingene i forbindelse med denne type prosjekt, der store data må behandles og overføres. Til sist må det nevnes at dette også hadde vært mulig med XML, hvis vi hadde valgt å gå for det istedenfor JSON.

### 3.3.4 Prosessering av data, server vs. klient

Slik beskrevet i 3.3.3 går det en datastrøm fra databasen til klienten, via en webserver. Data må prosesseres på veien, og hvor denne prosesseringen er mest gunstig å utføre er det vi skal ta for oss i dette kapittelet.

I 3.4 beskriver vi hvor de ulike systemene blir driftet, noe som får store følger for hvor dataprosessering bør foregå. Resultatet vi kom frem til var at databasen og webserveren skulle forbli på skolens server, frigg. Lys- og flydeteksjonssystemet fortsetter å kjøre på maskinen på oppdragsgivers kontor. Dette var ikke optimalt, men akseptabelt etter at disse ble oppgradert (se 3.1.1).

Med dette på plass var spørsmålet hva som skulle prosesseres hvor. Dataflyten beskrevet i 3.3.3 tilsier at alt begynner ved database-serveren på frigg. Dette er dekket i 3.3.4.1.

Etter dette skal data grupperes, dette kunne vært gjort både på webserveren og på klient-maskinen, men vi velger å gjøre dette på serveren. Dette er beskrevet i 3.3.4.2.

Når databaseserveren og webserveren har gjort sitt står vi så igjen med klient-maskinen, som så står for resten av prosesseringen. Dette bringer opp et annen viktig moment når vi snakker om prosessering, *stress*. Hessdalen fenomenet er av internasjonal interesse, så det er ikke utenkelig at det potensielt kan bli flere som ønsker å bruke vårt system samtidig. Altså blir det stor pågang på alle deler av prosessen. Hva skjer hvis 10 brukere prøver å hente ut data fra databasen samtidig, går dette bra? Hva med webserveren? Hva med 100 forespørsler tett oppunder hverandre?

Slik vår løsning er, vil prosesseringen på serveren nå være relativt "lett". Data hentes ut, gjennomgår en relativt enkel prosessering på webserveren, før de så sendes videre til klienten. Slik beskrevet i 3.3.3.3 blir ikke lenger data aggregert på webserveren mens det behandles, derfor er ikke dette lenger en kritisk problemstilling. Både database- og webserveren blir driftet på skolens servere, og med den relativt minimale prosesseringen som kreves av dem, burde de kunne takle denne pågangen.

Det skal også nevnes at dette blir en engangs-forespørsel til serveren, det er ingen kontinuerlig forbindelse tilstedet. Dette gjør at når serveren er ferdig med å behandle forespørselen til en bruker så vil den ha helt frie hender når neste forespørsel dukker opp.

Til sist kommer vi til klienten, som står for all gjenstående prosessering. Dette betyr at antall brukere av systemet nå er likegyldig, da prosessering, samme hvor krevende, uansett bare vil gå utover den enkelte klientmaskin.

Da er det eneste som står igjen å forsikre seg om at klienten takler den arbeidsbyrden den nå har blitt tildelt. Dette er detaljert i 3.3.4.3.

#### *3.3.4.1 Prosessering i databasen*

Databasen gir oss de data vi ønsker, men har også mulighet til å manipulere og filtrere disse ved uthenting. Dette hentes så inn til vår webserver som kjører på samme server-nettverk (frigg). Deretter skal disse dataene prosesseres, konverteres til JSON og sendes til klient-maskinen.

Det første vi må ta med i bektraktningen er hvilke maskiner som er raskest og mest effektive med tanke på prosessering. Det er et faktum at databaseservere er designet fra bunnen av til å være effektive på å prosessere data. I tillegg kjører denne database-serveren på skolens servere, som har relativt mye ressurser tilgjengelige, og er således godt egnet for prosessering av data. Vi prøver altså å overlate så mye prosessering som mulig til database-serveren.

Desverre har databasen den problemstillingen at SQL er relativt begrenset når det gjelder avanserte transformer og prosesseringer. Det er fullt mulig å få til det meste med SQL, men det er relativt tungvindt. Et av de største behovene vi hadde med tanke på bearbeiding av data, var å gruppere punktene i en overflyvning til et objekt. Desverre var dette vanskelig/umulig å få til med den relasjonsdatabasen vi jobbet mot. Det hadde vært mulig å bygget om på strukturen i databasen for å gjøre slikt lettere, men da hadde vi samtidig måttet endre på flydeteksjonssystemet til den tidligere prosjektgruppen. Dette gjorde vi ikke, slik detaljert i 3.1.

Noe databaser *er* usedvanlig godt egnet til er filtrering og sortering av data. Derfor ble begrensninger av hvilke data vi hentet ut (for eksempel bare fly innenfor en gitt breddegrad) noe vi overlot til databasen. I tillegg gjorde sortering av verdier det mye enklere å gruppere dataene på webserveren. Dette er trivielt arbeid for databaser, derfor overlot vi også sortering til databasen.

Desverre var det begrenset hvor mye vi kunne dra nytte av databasens evner utover dette.

#### *3.3.4.2 Prosessering på webserveren*

Et viktig poeng å dra frem når det gjelder prosessering på webserveren er gruppering av data. Når data grupperes blir det mulig å skille ut redundant informasjon, altså får man mindre data å behandle i senere deler av systemet. Dette er hovedsakelig grunnen til at vi valgte å utføre gruppering på webserveren, slik at vi slapp å overføre redundant data til klienten. Dette er selvsagt universalt positivt. For å forklare hva vi mener med utskilling av redundant data, tar vi for oss et par rader fra databasen, se Tabell 3.1: Databasekolonner 2 (kopi av tidligere tabell).

Tabell 3.1: Databasekolonner 2

| Hexident | Postime | Flightid | Latpos | Longpos | Track | Altitude | Osv. |
|---|---|---|---|---|---|---|---|
| 06A052 | 1370085739 | QTR991 | 61.48540 | 11.30859 | 295 | 32000 | |
| 06A052 | 1370085774 | QTR991 | 61.51859 | 11.16329 | 295 | 32000 | |
| 06A052 | 1370085796 | QTR991 | 61.53936 | 11.07198 | 295 | 31975 | |

Slik vi ser er dette *ett* fly i nærheten av Hessdalen, som har blitt plukket opp av sensoren. Hver rad er et "punkt" på kartet der flyet hadde dataene vist i de relaterte kolonnene. Vi ser at Hexident og Flightid er lik for hele overflyvningen, noe som er logisk å forvente. Et fly vil ha samme id og rutenummer for en hel overflyvning. De andre feltene vil imidlertid forandre seg (merk at Track tilfeldigvis er lik for alle 3 rader i dette eksempelet, dette er en tilfeldighet).

Altså kan vi slå sammen alle disse radene til en overflyvning, med de gitte Hexident og Flightid verdiene, slik at disse bare oppgis *en* gang. Disse verdiene er deretter gjeldene for alle tilhørende punkter på overflyningen. Vi har altså spart oss for å ta med redundant informasjon, noe som betyr at dataene sendt til klienten vil være betraktelig mindre. På denne måten sparer vi prosessorkraft og båndbredde både på webserveren, linjen mellom server og klient og klientmaskinen.

#### *3.3.4.3 Prosessering på klientmaskinen*

Nå som klientmaskinen har mottat sine data, er det essensielt at arbeidsbyrden som har blitt overlatt til den er noe den kan takle.

Dataene som klienten mottar må plottes på et kart, og selve denne jobben, samt manipulering av kartet når alle data er tilstedet, er klientens hovedoppgave. Slik detaljert i 3.3.2.2 ble det raskt et problem med for store mengder data. Dette klarte vi å løse ved å redusere samplingsfrekvens og tynne ut databasen. Etter flere tester var det klart at systemet ble uresponsivt ved store datautvalg, men at det fungerte tilfredsstillende innenfor intervallene oppgitt som øvre grense av oppdragsgiver (maksimum et par dager).

Klientmaskinen må hovedsakelig lage ikon-objekter for Google Maps, og binde disse opp mot kartet sammen med tilhørende informasjonsvinduer. Informasjonsvinduer inneholder dataene for et ikon på kartet (en flyobservasjon). Dette kan ikke kommes utenom, og er akseptabelt selv om det er en krevende oppgave.

En annen problemstilling var hvor nøye flyets rute skulle beskrives. Vi måtte selvsagt illustrere flyet der det befant seg i et gitt "øyeblikk" (se 3.2.4.5). Spørsmålet var om vi også skulle illustrere den fulle banen til flyet, altså alle punkter på overflyvningen. I tillegg burde disse forbindes med en linje slik at sammenhengen var tydelig. Dette ville være relativt krevende, da mange ikoner og linjer måtte vises på kartet samtidig.Dette lot seg løse på en gunstig måte, og er beskrevet i 3.2.4.4.

Den siste store arbeidsoppgaven på klientmaskinen var visning av en såkalt "heatmap", slik beskrevet i 3.2.4.1. Dette gir brukeren en samlet oversikt over hvor flytrafikken var mest intens i det valgte tidsrommet, og krever at brukermaskinen prosesserer en relativt stort sett med data. Alle koordinatsett til flyobservasjoner innenfor et gitt intervall må sendes til en funksjon som lager en heatmap basert på disse. Det viste seg imidlertid at dette hadde en uventet positiv effekt. Vårt heatmap baserte seg på det samme tidsutvalget som allerede var gjort for kartet. Derfor var det

hovedsakelig snakk om de samme mengdene data, bare fremstilt anderledes. Det var mye lettere for maskinen å prosessere og tegne opp et heatmap enn det var å tegne opp individuelle noder og fly. På grunn av dette ble heatmap en praktisk måte å plotte store mengder data på, uten at dette tok for lang tid. Det en heatmap tegnet på et par sekunder, tok nærmere det tidobbelte når alle noder skulle tegnes individuelt. Altså var generasjonen av selve heatmap-en neglisjerbar, og selve fremstillingen på kartet var *mer* effektiv enn det var med ikoner og linjer.

### 3.3.5 Analyse av data

Dataanalysen i vårt prosjekt har to distinkte vinklinger, disse blir dekket i henholdsvis 3.3.5.1 og 3.3.5.2.

#### *3.3.5.1 Dataanalyse for effektmålet, forståelse av lysfenomenet*

Den første type analyse er den som har til hensikt å få bedre forståelse for lysfenomenet i Hessdalen, altså en del av effektmålet for vår oppgave. Hvis vi kunne introdusere direkte analysemuligheter for dette, ville arbeidet til oppdragsgiver potensielt forenkles kraftig. Problemet med et fenomen av denne karakteren er at det finnes potensielt uendelig med kilder. Det kan være forårsaket av vær, fly kan spille en viktig rolle, kanskje topologien til området er relevant? På grunn av de mange muligheten vil fremgangsmåtene for å komme til bunns i dette også være nærmest ubegrenset. Det ble meget vanskelig for oss å utarbeide former for analyse som ville være gunstige for arbeidsgiver, da dette ville ha krevd mye tid og ressurser fra vår side. Vi måtte prioritere å bli ferdig med oppgaven gitt i prosjektbeskrivelsen i førsterekke.

Et eksempel kunne vært at vi hadde en hypotese om at lavtflyvende fly var relatert til lysfenomenet. For å verifisere dette måtte vi ha implementert spesifik funksjonalitet for å fremheve lavtflyvende fly. Dette bare for teste en hypotese.

Alle former for analyse av denne typen ville også vært basert på spekulasjon: "hva hvis X er relatert til lysfenomenet". Analysemetoder måtte utarbeides for hvert unikt tilfelle. Da vi ikke hadde noen klar oversikt over hva som var interessant å analysere (da dette ikke var en del av vår oppgave), så vi det best å gå bort fra denne type analyse. Vi overlot dette heller til fremtidige brukere av systemet, som kunne utføre dette manuelt. Vi fokuserte på å legge til rette slik at nødvendige verktøy skulle være tilgjengelige for å best mulig kunne utføre denne oppgaven. Ulike teorier kunne dermed "manuelt" verifiseres ved hjelp av vårt system. For å gå tilbake til eksemplet over, kunne et problem av denne typen nå løses ved å lese av høyde-verdiene for diverse fly. Eneste forskjell var at vi ikke nå la til dedikert funksjonalitet for dette.

#### *3.3.5.2 Dataanalyse for resultatmålet, bedre oversikt over flytrafikken*

Den andre type dataanalyse relevant for vår oppgave, var den som gjorde det lettere å bearbeide data om flytrafikken generelt. Et eksempel på dette var å analysere regelmessigheten av flyruter, slik at det ble lettere å få oversikt over disse. Dette kunne så brukes som et verktøy for effektmålet i ettertid. Et annet eksempel kunne være å analysere uregelmessigheten av fly, altså fly som ikke var del av en standard rute. Denne tankegangen hadde imidlertid en fundamental brist i logikken. Lysfenomenet i Hessdalen viser seg på tilsynelatende helt tilfeldige steder. Det var altså irrelevant hvilken type fly som passerte området, det var irrelevant om det var et regelmessig fly eller ikke. Det eneste som var av interesse var om et fly, hvilket som helst fly, var i område på samme tidspunkt som lysfenomenet eller ikke.

Dette bringer oss til den typen analyse som viste seg å være mest relevant for vår oppgave, sannsynligheten for fly. En stor svakhet ved flydeteksjonssystemet var at dette ikke hadde mulighet for å hente inn data om fly som ikke inneholdt den relevante sensoren. Derfor ville det vært meget gunstig hvis vi kunne presentere for brukeren en sannsynlighet for fly i et valgt område.

Flere fremgangsmåter fantes for denne type analyse (se 2.2.4.1). Den beste og mest praktiske å implementere var en såkalt heatmap. Altså et slags filter som legges over kartet, som representerer intensiteten av fly ved hjelp av farger. Områder som hadde mest flytrafikk innenfor et gitt intervall ble gitt en "varmere" farge enn områder der flytrafikken var minimal. På denne måten fikk brukeren en approksimasjon for sannsynlighet for fly. Områder med mye flytrafikk ville selvsagt ha større sannsynlighet for fly, enn områder med liten flytrafikk. Selve implementasjonen av heatmap ble dekket i 3.2.4.1.

Problemet var hvordan denne løsningen skulle implementeres. Heatmaps har den svakheten at det gir en meget generell oversikt. Overblikket brukeren får er ikke nødvendigvis spesifikt nok til å kunne hjelpe med å bekrefte eller avkrefte om et fly befant seg i et område på et gitt tidspunkt. Vi vurderte muligheten for å la brukeren spesifisere parametere rundt hvilke data som skulle brukes for å lage et heatmap. Løsningen vi til slutt endte opp med var at tidsutvalget brukeren gjorde til tidslinjen var det samme utvalget som heatmap-en baserte seg på. Slik fikk brukeren full kontroll på tidsintervallet, og kunne derfor fritt bestemme hvilke data som skulle brukes.

Et usikkerhetsmoment sto igjen, og det var hvordan punktene som heatmap-en baserte seg på skulle behandles. Databasen inneholder diskrete verdier for en overflyvning, ikke en kontinuerlig "linje". Altså kunne for eksempel 25 punkter beskrive en strekning på flere kilometer. Dette kunne i værste fall gi en dårlig representasjon av overflyvningen, og derfor også en dårlig heatmap. Vi vurderte muligheten for å løse dette ved hjelp av interpolasjon (se 3.2.4.6). På denne måten kunne vi generere så mange punkter som vi ønsket.

Det viste seg imidlertid at en heatmap basert på de diskrete verdiene fra databasen var mer en godt nok for vårt bruk, da dette ga et tilstrekkelig bilde av en overflyvning.

### 3.3.6 Gruppering av fly i en overflyvning

Behovet for å gruppere flyobservajoner i en overflyvning ble umiddelbart klart for oss da vi fant ut at databasen måtte tynnes, slik beskrevet i 3.3.2.3. Gruppering var også viktig med tanke på hvordan dataene ble behandlet på kartet, samt andre mer avanserte funksjoner (slik som å vise stier, se 3.2.4.4). Hvilke muligheter vi hadde for gruppering er beskrevet i delkapitlene under.

For å spesifisere nøyaktig hvilken type gruppering det er snakk om, dreier det seg altså om gruppering av alle punkter som et fly "etterlater" seg mens det flyr over Hessdalen. Mens flyet passerer (i en enkelt overflyvning) vil flysensoren lese av data fra flyet ved regelmessige intervaller, disse lagres som separate rader i databasen. Alle disse tilhører en logisk gruppe, en "overflyvning". Et fly som kom tilbake senere ville bli behandlet som en *separat* overflyvning.

Målet for gruppering er således å bruke dataene fra flyobservasjonene for å finne ut hvilke som hører sammen. Deretter kan disse behandles som *ett* objekt i resten av systemet.

Vi vurderte flere metoder som kunne ha gjort gruppering meget lettvint. Blant annet å introdusere en ekstra kolonne i databasen som identifiserte en overflyvning. Alle disse metodene hadde

imidlertid en stor svakhet, det krevde at vi utførte fundamentale forandringer i flydeteksjonssystemet, som så måtte bli reflektert for alle data allerede logget til databasen. Derfor måtte problemet løses ved å se på kolonner allerede tilstedet i databasen.

#### 3.3.6.1 Gruppering ved hjelp av hexident og postime

Gruppering ved hjelp av hexident og postime vil si at man ser på en hexident (en unik identifikator for et spesifikt fly) , samt tidsforskjeller (gitt av postime) mellom ulike observasjoner av dette flyet. Hvis man tok for seg alle observasjoner av en hexident, og deretter så på tidsforskjellen mellom disse, kunne alle punkter som var innenfor en gitt grense grupperes. Man dannet altså logiske grupper for hexident'er som forekom rundt samme tidspunkt, altså overflyvninger.

Denne løsningen virket lovende, men etter å ha holdt et møte med Dick fra den tidligere prosjektgruppen, ble vi klar over en viktig detalj. Det var en liten mellomlanding i nærheten av Hessdalen der fly kunne stå på bakken i bare få minutter, før de så fløy videre (nå med et nytt rutenummer). Dette tidsrommet kunne være mindre enn tidsforskjellen mellom to punkter på en overflyvning. Altså ville en gruppering på denne måten gå glipp av en signifikant hendelse, spesielt med tanke på at flyet som tok av fra denne mellomlandingen nå kanskjen hadde en helt ny Flightid (et slags rutenummer). Det kunne også bli problematisk å prøve å skille på flightid, da denne (slik nevnt i 3.3) ikke var til å stole på. Flightid kunne til tider være "blank".

Derfor gikk vi bort fra denne tilnærmingen, og så på muligheten for å bruke Flightid, slik beskrevet i neste delkapittel.

#### 3.3.6.2 Gruppering ved hjelp av FlightID

Denne tilnærmingen hadde en stor problemstilling som gjorde at det nesten umiddelbart ble umulig å bruke denne metoden. Flightid-feltet i databasen var et datafelt som i motsetning til hexident kunne være "tom". Altså kunne rader i databasen være uten flightid. Vi hadde heller ingen garanti for at flightid forble konstant for en overflyvning (spesielt med tanke på feil som kunne oppstå i flydeteksjonssystemet). Selv om flightid potensielt sett kunne være ideell for gruppering, var det vanskelig å finne en løsning hvis denne manglet data. Disse usikkerhetene førte til at vi raskt gikk bort fra å bruke flightid til noe så kritisk som å gruppere data.

#### 3.3.6.3 Gruppering ved hjelp av hexident og regtime

Gruppering ved hjelp av hexident og regtime baserer seg på en antakelse, at regtime er tidspunktet der systemet lagrer en overflyvning til databasen. Altså at denne er lik for alle flyobservasjoner i overflyvningen. Når dette er tilfellet kan så gruppering skje hovedsakelig på regtime, med hexident for å skille på overflyvninger som ble registrert nøyaktig samtidig. Hexident er altså en unik identifikator på et bestemt fly.

Det eneste problemet med denne fremgangsmåten er at regtime potensielt kan inneholde små variasjoner innenfor en overflyvning. Regtime er et datafelt som blir opprettet når dataene logges til databasen, og alle observasjoner som utgjør en overflyvning blir lagret til databasen samtidig. Problemet oppstår hvis den interne systemklokken i databasen "tikker over" til et nytt sekund mens lagringen utføres. I slike situasjoner vil det altså oppstå et skille i regtime, selv for observasjoner som egentlig tilhører samme overflyvning. En liknende problemstilling oppstår hvis flyet er usynlig for systemet i et kort tidsintervall (se "LAST_POS_TIMEOUT" i 4.1.3). Hvis dette intervallet er langt nok vil

systemet anta at flyet har forlatt sensorens radius, og overflyvningen blir konkludert og lagret i databasen. Deretter dukker flyet opp igjen, men nå blir dette en ny overflyvning.

Dette var en problemstilling vi effektivt kunne løse ved å "manuelt" sette sammen grupper der regtime-verdiene var innenfor en gitt grense. Altså hvis flere observasjoner med samme hexident forekom med regtime verdier som var kort tid fra hverandre, ville disse bli satt sammen til samme gruppe.

Regtime feltet genereres av funksjonen "`NOW()`" i databasen, altså tidspunktet i databasen når dataloggingen finner sted. Etter å ha undersøkt denne funksjonen i detalj viste det seg at det var sterkt usannsynlig at en overflyvning skulle få ulik regtime på punkter i en sammenhengende overflyvning. En del av dokumentasjonen til "`NOW()`" ser du under:

"`NOW()` returns a constant time that indicates the time at which the statement began to execute. (Within a stored function or trigger, `NOW()` returns the time at which the function or triggering statement began to execute.) This differs from the behavior for `SYSDATE()`, which returns the exact time at which it executes." (Oracle, 2014)

Slik vi ser av denne dokumentasjonen så returnerer `NOW()` tiden da denne funksjonen **begynte** å eksekvere. Vi antar at dette betyr tidspunktet når databasen mottar forespørselen om å lagre en overflyvning. Flydeteksjonssystemet overfører alle radene i en overflyvning samtidig, ergo burde disse bli tildelt samme regtime, selv om tiden skulle forandres under lagring.

Det er fortsatt mulig at flydeteksjonssystemet overfører en og en rad, slik at de teoretisk sett kan få ulike tidspunkter, men dette er også meget usannsynlig. Databasehåndteringen i deres programkode sørget mest sannsynlig for at alle forespørslene overføres som en transaksjon til databasen. Desverre var det problematisk å finne ut nøyaktig hvordan dette foregikk uten å ha satt oss grundig inn i flyeteksjonssystemet og dets biblioteker.

Vi undersøkte ikke dette nærmere, da vi likevel ikke kunne være sikre på at alle leddene i prosessen var helt nøyaktige. Derfor valgte vi å utforme vår løsning slik beskrevet tidligere, at overflyvninger med nærliggende regtime verdier ble slått sammen. På denne måten var vi sikret mot potensielle feil som kunne oppstå, uavhengig av system.

Bruken av hexident og regtime for gruppering var altså vellykket. Disse datafeltene var også meget pålitelige. Hexident var del av tabellens primærnøkkel, som automatisk garanterte at denne var unik og veldefinert. Regtime ble produsert av en funksjon i databasen, derfor var det også umulig at denne kunne føre til problemer.

#### *3.3.6.4 Oppslag i eksterne systemer*

Det var også en fjerde mulighet for gruppering, som vi i værste fall kunne ty til. Hvis dataene i databasen ikke hadde vært tilstrekkelige til gruppering, kunne oppslag i eksterne systemer hjelpe oss å finne ut hvilke flyobservasjoner som hørte sammen.

Vi utforsket ikke denne mulighet i større grad, da vi oppnådde en akseptabel løsning med hexident og regtime. Denne muligheten er likevel verdt å nevne, da det antakelig hadde blitt helt essensielt hvis regtime-feltet ikke eksisterte i databasen.

En slik løsning ville mest sannsynlig ha innebært at nøkkel-verdier fra databasen skulle slås opp i eksterne systemer, som kunne gitt os data for å hjelpe med gruppering. Dette ville ført til at datauthenting hadde tatt betraktelig mer tid, da forespørsler til eksterne systemer måtte skje for hver eneste overflyvning. Det var altså meget gunstig for brukeropplevelsen at vi kunne løse problemet på en annen måte.

#### *3.3.6.5 Hvor grupperingen finner sted*

Etter at grupperingsmetoden var på plass, var spørsmålet hvor dette skulle finne sted. Gruppering av data var en relativt tidskrevende prosess. Derfor var det ønskelig at dette foregikk på en del av systemet som hadde ressurser nok til å gjøre dette raskt og effektivt.

Den optimale løsningen ville vært å utføre selve gruppering i databasen. Dette var desverre umulig da databasen ikke omhandler objekter, men enkeltrader. I teorien ville det vært mulig å gjort en uthenting som gjorde gruppering **enklere**, men full gruppering ville ikke la seg gjøre. Data måtte uansett bli videre prosessert på webserveren før det ble gjort om til JSON format og sendt til klienten. Det var dette vi valgte å gjøre.

Gruppering på webserveren var relativt enkelt, da vi hadde direkte kontroll over databaseuthentingene, samt mulighet for mer avansert programmering (noe databasen selv ikke støttet direkte). Webserveren var også plassert på en relativt kraftig maskin, slik spesifisert i 3.4.3, dermed burde ikke arbeidsbyrden by på problemer. Grupperingen var heler ikke en så krevende prosess at det ville føre til problemer ved stor pågang. Derfor valgte vi å utføre datagrupperingen på webserveren.

Det skal også nevnes at det var mulig å gruppere data på klientmaskinen, men dette hadde hovedsakelig to store problemstillinger. For det første måtte data enten hentes direkte til klienten, eller så måtte det sendes fra webserveren ugruppert. En direkte forbindelse ville betydd at klientmaskinen hadde login-informasjon til databasen, noe som var uakseptabelt. Problemet med å overføre ugrupperte data var at dette førte til unødvendig bruk av båndbredde. Gruppering på webserveren førte til at redundante data som hexident (lik for alle punkter på overflyvningen) bare ble oppgitt **en** gang, og deretter var gjeldende for hele overflyvningen. Beste løsning var altså å utføre gruppering på webserveren.

## 3.4 Plassering og drifting av systemer

Vår løsning omfattet tre ulike systemer, alle disse utførte relativt krevende arbeid. De kommuniserte også seg imellom. Det var mange muligheter når det gjeldt hvilke maskiner disse skulle kjøre på, og vi måtte bestemme oss for hvilken plassering som ville bli mest gunstig for hvert enkelt system, og for helheten.

### 3.4.1 Database

Da prosjektet begynte kjørte databasen på frigg.hiof.no, en server som driftes av høgskolen. Da databasen kontinuerlig ble fylt opp med informasjon, og data aldri ble slettet, var vi avhengige av at lagringsmediumet kunne utvides hvis dette skulle bli et problem. Det var altså viktig å tenke på at serveren som hostet databasen ble overvåket og vedlikeholdt regelmessig. Dette oppnådde vi enklest ved å la databasen forbli på skolens server. Skolen var ansvarlig for å holde serveren ved like,

som første til at eventuelle nødvendigheter, slik som utvidelse av lagringsmedium, var noe som automatisk ville bli tatt hånd om.

Eventuelle systemfeil som førte til at databasen ble utilgjengelig ville også bli raskt løst når vi lot databasen bli driftet på skolens server. Hadde vi flyttet databasen til den dedikerte maskinen på arbeidsgivers kontor, kunne det potensielt ta måneder før dette ble løst. Hele vårt system ville så blitt utilgjengelig i denne perioden. Det skal også nevnes at denne maskinen hadde ansvar for lysdeteksjonssystemet, et system som utførte krevende bildeanalyseoppgaver. I tillegg var flydeteksjonssystemet også plassert på denne maskinen. Altså var beste valg å la databasen forbli på frigg.

### 3.4.2 Flydeteksjonssystem

Fly- og lysdetksjonssystemene kjørte på en dedikert maskin på oppdragsgivers kontor. Denne ble sjeldent vedlikeholdt. Disse var to helt separate systemer, men begge var allerede tett integrert med maskinen de kjørte på. Lysdeteksjonssystemet utførte relativt tunge bildeanalyseoperasjoner flere ganger i sekundet. Dette systemet hadde også problemer med en minnelekkasje, som gjorde at hele maskinen som drev *begge* systemer kræsjet regelmessig. Dette var ikke en optimal løsning. Da vi snakket med den tidligere prosjektgruppen kom det frem at de opprinnelig ønsket at systemet skulle driftes på Frigg (skolens server), men dette var ikke mulig å få til. Ved et senere tidspunkt gikk den tidligere prosjektgruppen inn i systemet for å prøve å løse disse problemene, se 3.1.2 for en forklaring på hva dette innebar.

Etter at systemet hadde blitt forbedret konkluderte vi med at det *var* uoptimalt å hoste dette systemet på maskinen på oppdragsgivers kontor, men dessverre var det ingen annen mulighet. Dessuten var løsningen mer akseptabel, nå som systemet hadde blitt forbedret. Eventuelle feil ville også bare gå utover flydeteksjonssystemet, ikke vårt eget system (med unntak av datamangel i perioder der systemet var nede). Selv om dette systemet sluttet å fungere, ville fortsatt vårt system fungere som normalt, fordi det bare baserte seg på dataene fra databasen, samt webserveren.

### 3.4.3 Webserver

Plassering av webserver var den vanskeligste problemstillingen med tanke på hva som ville være mest gunstig. Dette var hovedsakelig fordi vi hadde et problem med å hente ut de store mengdene data som systemet vårt arbeidet mot. Ved prosjektstart ble vi tildelt plass på skolens webserver, frigg. Altså samme server som databasen kjørte på. På grunn av dette hadde vi lite kontroll over selve webserveren.

Vi vurdert derfor å flytte dette over til samme maskin som flydeteksjonssystemet, for å få full kontroll på webserveren. Likevel, slik beskrevet i forrige delkapittel, så var antakelig ikke dette en god ide. Dette var fordi problemene med lysdeteksjonssystemet, og generelt dårlig vedlikehold av maskinen, gjorde dette for ustabilt til at dette kunne være en varig løsning.

Hessdalenfenomenet er også av potensiell internasjonal interesse, noe som kan føre til perioder med store mengder trafikk. Hvis vårt system var ustabilt, kanskje regelmessig, ville dette være uegnet til bruk på et webområde som dette.

Derfor var det altså klart at vi måtte gå for en mer stabil webserverløsning. Spørsmålet var om vi lot websiden forbli på frigg, eller om vi så etter andre leverandører. Vi bestemte oss for å bruke frigg, da

dette var en stabil server vedlikeholdt av skolen. Hvis det skulle vise seg å bli problemer i fremtiden kunne websiden lett flyttes. Merk også at vår side potensielt kunne bli innlemmet i hovedwebområdet for Hessdalen. Derfor utarbeidet vi vår løsning slik at den lett kunne flyttes i ettertid.

Problemet vi deretter måtte ta hensyn til var at vi nå hadde begrenset kontroll på webserveren. Et problem med minneallokering oppsto raskt. Webserveren tillot ikke allokering av de store mengdene data som vårt system arbeidet med. Vi fikk økt denne grensen ved å ha snakket med driftsansvarlig for frigg, som gjorde at vi nå kunne hente ut betraktelig mer data. Desverre var det ikke sikkert at denne løsningen var permanent, derfor endte vi blant annet opp med å tynne ut dataene i databasen. Dette ble dekket i detalj i 3.3.2.2.

Problemet med dataallokering løste seg imidlertid da vi oppdaget muligheten for kontinuerlig printing av JSON, slik beskrevet i 3.3.3.3. Dette gjorde at data aldri ville bli aggregert på webserveren utover en enkelt overflyvning, derfor var vi nå trygge på å aldri nå grensen for minneallokering.

En siste problemstilling vi sto ovenfor var hvordan webserveren ville tåle stor pågang, med tanke på at blant annet gruppering av data og liknende ble prosessert av serveren. Ville dette føre til problemer hvis for mange klienter aksesserte serveren samtidig? Frigg er en server som har ansvaret for å hoste mange systemer ved høgskolen, derfor var det sannsynlig at denne vile tåle eventuell stor pågang. Det var nettopp derfor frigg var et ideelt valg for webserveren, da den var designet for å håndtere slik pågang. Vi valgte altså å la webserveren være på frigg, men med den forutsetningen at vi måtte ruste vårt program for å takle potensielle feil som kunne oppstå, slik at ikke vårt system sluttet å fungere hvis en eventuelle grense ble nådd. Til gjengjeld for at vi tok i bruk en webserver vi ikke selv kunne administrere, var dette en stabil server som ville bli vedlikeholdt selv etter at vårt prosjekt konkluderte.

I ettertid, da produktet var ferdig utviklet, ble systemet integrert med hovedsiden for Hessdalenprosjektet, Hessdalen.org. Vi regnet med at denne hadde minst like god kapasitet som frigg, og dermed ble dette bare en detalj som ikke ga utslag på ytselsen.

## 3.5 Arbeidsmetode

Da programmering var en vesentlig del av vårt prosjekt, var det viktig å legge til rette for at dette kunne skje mest mulig effektivt. Derfor var det å utarbeide konkrete arbeidsmetoder helt kritisk for å få et vellykket prosjekt.

### 3.5.1 Versjonshåndtering

To alternativer som ble presentert for oss var Git og SVN, der SVN var i overkant simplistisk og Git i overkant komplekst med tanke på vår arbeidsmetode (se 3.5.2).

Løsningen vi valgte å gå for var Mercurial (med klienten TortoiseHg) (TortoiseHg, 2014). Denne hadde et gruppemedlem hatt erfaring med tidligere, derfor var vi sikre på at programvaren dekket de behovene vi hadde for vårt prosjekt. Mercurial var også en løsning der hver bruker hadde sitt eget lokale repository (se 2.5.1), noe vi spesifikt ønsket av løsningen. Det var heller ikke like problematisk som Git med tanke på brukervennlighet. Derfor var Mercurial er en god balanse mellom Git og SVN, og perfekt for våre behov.

### 3.5.2 Delegering av arbeidsoppgaver

Programmering har den problemstillingen at det kan være vanskelig å delegere ut arbeidsoppgaver, da det ofte krever en helhetlig oversikt over systemet for å kunne utarbeide en løsning. Det er også viktig å ta med i betraktningen at potensielle feil i et ledd av programkoden kan ha katastrofale konsekvenser for andre deler av systemet. Samt at deler av programmet kan anta visse ting om andre deler, noe som byr på problemer hvis disse antakelsene viser seg å ikke bli innfridd.

Altså er det konseptuelt problematisk å dele opp programmeringsarbeid, så sant det ikke er snakk om frittstående deler som kan utvikles separat. Vår løsning var en enkelt webside, der det var problematisk å dele opp arbeidet på en gunstig måte. Dette var i hovedsak fordi behandling av data fra databasen, og hvordan dette ble vist frem, var den største (og nesten enerådende) delen av arbeidet. Hvert ledd i kjeden var tett knyttet sammen, slik at det nærmest var umulig å utarbeide disse delene individuelt. Hvert ledd av utviklingsprosessen måtte ha vært klart definert og testet før neste del kunne fungere, og forandringer i konsepter (som var uunngåelig i lengden) ville få konsekvenser for hele prosjektet - alle andre ledd. Hvis disse delen ble utarbeidet separat, ville det bli enormt komplisert og forandre og integrere disse.

Vi valgte å løse denne problemstillingen ved å hovedsakelig overlate programmeringsarbeid til ett gruppemedlem. Løsninger ble utformet i fellesskap, men selve implementasjonen ble utført av en person. Ved hjelp av versjonshåndteringen beskrevet i delkapittel 3.5.1, var det lett for alle gruppemedlemmer å holde følge med utviklingsprosessen, selv om man selv ikke aktivt arbeidet på koden. På denne måten var alle tett involvert i utviklingsprosessen, men vi slapp problemstillinger med delegering av programmeringsjobber til flere gruppemedlemmer.

Da vi hovedsakelig hadde en person som jobbet med implementasjonen av programmene, hadde vi mulighet til å effektivt arbeide med rapport og liknende samtidig som programmene ble utviklet. Vi fikk altså en situasjon der vi konstant hadde fremgang i prosjektet, mens vi på samme tid fikk loggført det nøye (se delkapittel 3.5.3). Det meste av arbeidet foregikk som gruppearbeid, slik at alle var tilgjengelig for spørsmål, forslag osv. i de ulike fasene av prosjektet. Når det gjaldt konseptuelle problemstillinger som måtte løses, slik som hvordan vi skulle gruppere data (se 3.3.6), ble disse som oftest utarbeidet på et møte med alle tilstedet. På denne måten fikk vi oversikt over alle muligheter med bidrag fra alle gruppemedlemmer, og alle var til en hver tid klar over statusen til prosjektet.

### 3.5.3 Loggføring av arbeid

Prosjektet strakk seg over flere måneder med arbeid, og alt arbeidet skulle resultere i en hovedrapport sammen med sluttproduktet vårt. Denne hovedrapporten skulle inneholde en detaljert beskrivelse av alt arbeidet som hadde skjedd i prosjektet. Derfor var det viktig å loggføre arbeidet grundig, slik at dette lett kunne innarbeides i hovedrapporten senere. En av de viktigste formene for loggføring i vårt prosjekt var issue tracking, altså en oversikt over arbeidsoppgaver i programløsningen vår. Se 3.5.3.1.

#### *3.5.3.1 Issue tracking*

Vi bestemte oss for å bruke Mercurial for versjonshåndtering, da det inneholdt den funksjonaliteten vi hadde behov for. Det var også enklere hvis administrasjon av issues og versjonshåndtering ble behandlet av det samme systemet. Selve funksjonaliteten til Mercurial var også riktigere for vårt prosjekt enn de andre løsningene vi så på. Dette hadde både med brukervennlighet, tekniske muligheter og støtte for eksportering å gjøre. Mercurial lot oss raskt og enkelt binde commits (en

forandring i koden) opp mot et pågående issue i Bitbucket. Mulighetene for å opprette og behandle issues var også bedre i Mercurial.

#### 3.5.3.2 Møtereferater

I løpet av prosjektet har det selvsagt vært mange møter. Disse var hovedsakelig delt i to ulike typer, møter med oppdragsgiver/prosjektveileder og interne møter for gruppen.

Disse var begge på samme format, men hadde to meget ulike hensikter. Møtene med oppdragsgiver og prosjektveileder var hovedsakelig for å administrere gangen i prosjektet, og for å få vite oppdragsgivers ønsker angående problemstillinger. Altså for å legge en plan for videre arbeid i prosjektet.

De andre møtene som var internt for vår gruppe ble mer som en arbeidslogg for hvordan vi hadde angrepet problemstillinger, og hvilke løsninger vi hadde kommet frem til. På denne måten hadde vi til en hver tid oversikt over hendelser som forekom, selv med møter som bare var diskusjoner internt i gruppen.

#### 3.5.3.3 Fortløpende hovedrapportskriving

Senere i prosjektet, etter at vi hadde kommet godt i gang med selve implementasjonen av sluttproduktet, begynte vi å arbeide aktivt på sluttrapporten. På denne måten hadde vi et ferdig rammeverk, der hendelser i prosjektet kunne settes direkte inn etterhvert som de oppsto. Dette gjorde det lettere å innarbeide informasjon inn i rapporten. Rapporten ga også alle gruppemedlemmer en detaljert oversikt over prosjektets status så langt, samt en oversikt over arbeidet som gjennsto.

Alternativet hadde vært å utarbeide hele rapporten mot slutten av prosjektet, noe som hadde ført til at hele rapportstrukturen med innhold måtte lages samtidig. Og da dette selvsagt hadde forekommet flere måneder etter prosjektstart, ville det blitt vanskelig å få dekkende beskrivelser av hendelser fra tidligere i prosjektet. Ved å heller arbeide med hovedrapporten under store deler av prosjektet, ga det også en mer jevn presentasjon av forløpet, og ikke bare den siste tiden. Dette gjorde det også mulig å presentere de ulike mulighetene vi hadde for å løse diverse problemstillinger, og ikke bare den løsningen vi valgte å gå for. Resultatet ble kapittel 2 og kapittel 3 slik de er utformet i denne rapporten.

#### 3.5.3.4 Timetall

I løpet av prosjektet førte vi også en kompakt oversikt over alle arbeidstimer i løpet av prosjektet. Dette inneholdt en kort beskrivelse av gjøremålene på en gitt dag. Vi valgte å gjøre dette slik at vi internt i gruppen hadde en lett tilgjengelig oversikt over prosjektets gang. Det ble lett å få en oversikt over prosjektet i retrospekt og danne seg en overordnet oversikt over alt som hadde hent.

### 3.5.4 Programmeringsfilosofi

Da vår oppgave i all hovedsak var basert på ønsker og tilbakemeldinger fra oppdragsgiver var det ikke utenkelig at forandringer kunne oppstå i forhold til den opprinnelige planen. Derfor ville programmering etter fossefall-prinsippet ikke være gjennomførbart. Vi valgte derfor å basere oss på en iterativ prosess, med regelmessige forandringer av systemet, hvis situasjonen krevde dette. Systemet ville bli testet og modifisert regelmessig, slik at det til enhver tid var mulig å innarbeide nye krav som måtte oppstå. Dette var også spesielt viktig med tanke på detaljene i

flydetekesjonssystemet. Hvis det ble nødvendig å modifisere systemet på grunn av en uforutsett problemstilling, var dette nå fullt mulig.

I tillegg til dette utarbeidet vi noen generelle retningslinjer for programmering. Under arbeidet tok vi sikte på å gjøre koden kortfattet. Vi hadde som mål å være relativt pragmatiske uten å lempe for mye på idealer og "regler" for programmering, å kategorisere koden på en logisk måte, og å forsøke å gjøre den så lesbar som mulig.

Vi satt opp noen interne retningslinjer for koding. Disse vektla blant annet:

- Navngivning av funksjoner eller variabler som brukes flere ganger:
  Disse navnene skulle være så klare og intuitive som mulig, uten å bli for lange.
- Retningslinjer for kommentarer:
  Vi antok at lesere var lesekyndige i de aktuelle språkene og forklarte kode der det ble ansett som nødvendig, men kommenterte ikke "blindt" kode der navngivning og struktur gjorde virkemåten innlysende.
- Hvordan kode skulle formateres og indenteres:
  Vi indenterer koden konsekvent med 4 mellomrom per blokk, og brukte ikke TAB-tegnet (`\t`). Klammeparenteser ble plassert i henhold til konvensjoner i de forskjellige språkene, der disse var tilgjengelige. Whitespace ble ikke brukt kun i blokksammenheng, men også for å gjøre mindre, logiske skiller i kodelinjer tydeligere.
- Hvordan versjonskontroll skulle brukes:
  Forandringer burde committes når de var "atomiske", frittstående endringer.

# 4. Implementasjon

I dette kapittelet gir vi en grundig oversikt over de ulike delene av vårt system, og hvordan disse henger sammen. Her er det mange ulike typer kode som kjører på de ulike systemene vår løsning omfatter, vi kommer *ikke* til å beskrive alle disse separate delene i detalj. Det er imidlertid flere hovedelementer som er kritiske for vårt system, samt flere mindre funksjoner og liknende som utfører mer avanserte arbeidsoppgaver. Disse delene vil vi gjerne forklare grundigere, da de ikke nødvendigvis er forståelige ved første øyekast, eller at de generelt er viktige å belyse for forståelse av systemet som helhet. I slutten av kapitlet finnes en total oversikt over alle deler av systemet, og hvordan disse er integrert med hverandre.

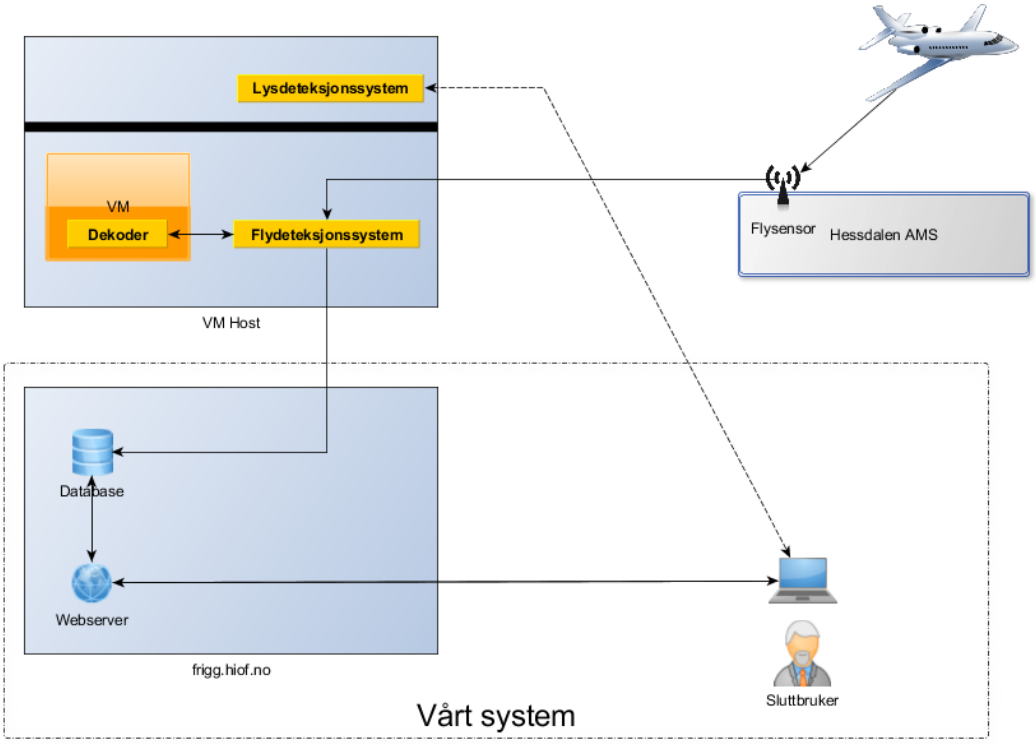
## 4.1 Flydeteksjonssystemet

Flydeteksjonssystemet som ble satt opp av den tidligere prosjektgruppen var et relativt omfattende system vi valgte å ikke sette oss nøye inn i. Likevel gir vi her en kort oversikt over hvordan dette systemet var utformet, da dette gir kontekst for vårt system.

### 4.1.1 Systemoversikt

En oversikt over flydeteksjonssystemet, samt hvordan dette henger sammen med vårt system, ser du i Figur 4.1: Systemoversikt. Merk at produktet ble integrert med hovedsiden for Hessdalenprosjektet etter at prosjektet var ferdig. Dette betyr at webserveren ikke lenger er plassert på frigg.hiof.no. Dette hadde likevel liten innvirkning på ytelsen og funksjonaliteten til vårt produkt, og er således ikke tatt hensyn til i systembeskrivelsen.

Figur 4.1: Systemoversikt




Formålet med systemet er at sluttbrukeren kan sammenlikne lysobservasjoner i lysdeteksjonssystemet med flyobservasjoner fra flydeteksjonssystemet, via vårt grensesnitt.

Slik vi ser av figuren, er både fly- og lysdeteksjonssystemet plassert på samme maskin, her kalt "VM Host". Dette er den dedikerte maskinen plassert på oppdragsgivers kontor, som kjører operativsystemet Linux. Denne maskinen inneholder en virtuell maskin med Windows, derav navnet.

Lysdeteksjonssystemet fungerer uavhengig av flydeteksjonssystemet, og er bare direkte relevant i vår oppgave da en systemsvikt potensielt kan påvirke resten av maskinen - flydeteksjonssystemet.

Flydeteksjonssystemet, slik vi ser av figuren, henter inn data om overflyvninger fra flysensoren plassert på Hessdalen Automatic Measurement Station (AMS). Deretter blir disse dataene dekodet ved hjelp av et program som kjører på den virtuelle maskinen på VM-host, for så å bli logget til databasen. Det skal nevnes at data ikke vil bli umiddelbart lagret til databasen. Dataene for en pågående overflyvning vil aggregeres til flyet ikke lenger kan oppdages av sensoren, deretter lagres alle data for overflyvningen på en gang.

Deretter kommer vårt system inn i bildet, da vi utelukkende arbeider mot databasen. Dette blir beskrevet senere i kapittel 4, og omfatter både prosessering på databasen, webserveren og på maskinen til sluttbrukeren.

### 4.1.2 Svakheter

Slik beskrevet i tidligere kapitler var lys- og flydeteksjonssystemene utformet på en problematisk måte. For det første var lysdeteksjonssystemet et relativt krevende system, som arbeidet konstant med tyngre bildeanalyseoppgaver. Eventuelle problemer med dette systemet ville gå direkte utover flydeteksjonssystemet. Dette var imidlertid noe vi bare måtte akseptere.

En annen problemstilling, som også ble beskrevet av den tidligere prosjektgruppen, var dekoderen. Maskinen som flydeteksjonssystemet kjørte på var en Linux maskin, men det var nødvendig med et proprietært dekoderprogramvare for å tolke meldingene sendt ut fra flyene. Dette programmet var et Windows program, og dermed ble det nødvendig med en virtuell maskin. Denne løsningen var ikke optimal. Desverre var dette noe de ikke kunne kom utenom, da de ikke hadde tilstrekkelig med dokumentasjon for lage en egen løsning for å dekode flymeldingene. Dette var likevel en viktig faktor, da potensielle problemer kunne oppstå i både dekoderen og den virtuelle maskinen, feil utenfor vår kontroll. Hvis dette skulle feile, ville ikke flydeteksjonssystemet lenger kunne bearbeide data fra flysensoren, disse ville så gå tapt for alltid. Denne maskinen ble heller ikke vedlikeholdt regelmessig. Derfor ville eventuelle feil og svikter i systemet kunne gå uoppdaget over lengre tid.

Disse var alle svakheter vi måtte ta hensyn til i vår løsning, for å kunne redusere eventuelle konsekvenser dette ville få for vårt system. En av de fremste måtene vi oppnådde dette på var å utelukkende forholde oss til databasen, som var "trygt" driftet på skolens egen server. Dette, samt vårt valg om å abstrahere oss fra detaljene og virkemåtene til disse systemene, førte til at vi kunne fokusere på vår løsning i første rekke.

Til sist skal det nevnes at bare fly som inneholder senderen som flydeteksjonssystemet arbeider mot, vil bli oppdaget av dette systemet. Alle andre fly vil være usynlige for sensoren, og vil således ikke dukke opp i vårt system.

### 4.1.3 Samplingsfrekvens

Vi skal ikke gå i detalj i flydeteksjonssystemet, men en viktig konfigurerbar parameter må nevnes.

Da vi begynte å hente ut data fra databasen viste det seg raskt at det var betraktelig flere punkter per overflyvning enn hva vi hadde bruk for i vårt system. Det var også problematisk å behandle de enorme datamengdene. Derfor ble det nødvendig å gjøre en forandring i flydeteksjonssystemet for å redusere antall punkter som ble lagret for en overflyvning. Resultatet av dette ble en justerbar parameter i en konfigurasjonsfil i flydeteksjonssystemet. På maskinen som hoster systemet er denne filen plassert her: `/home/hessdalen/flight/flight.cfg`

Innholdet i denne filen finner du i Figur 4.2: Konfigurasjonsfil, flydeteksjonssystem (noe informasjon er obfuskert).

**Figur 4.2: Konfigurasjonsfil, flydeteksjonssystem**

```
# Configuration file for flight detection.
#
DB_HOST_IP = frigg.hiof.no                  # MySQL server address
DB_HOST_PORT = 3306                      # MySQL server port
DB_NAME = ---                           # Database name
DB_USER = ---                           # Database user name
DB_PWD = ---                            # Database user password
#                                          #
```

```
LOG_FILENAME = ./errorFile.log           # ./ for Linux, .\ for Windows
#                                        #
LAST_POS_TIMEOUT = 60                    # Timeout in seconds, Save to db when
#                                        # flight out of sight for more than x seconds.
#                                        #
BASESTATION_IP = 127.0.0.1          # Base station PC's address
BASESTATION_PORT = 30003                 # Base station port
#
MAX_POS_REGS_PER_TRACK = 25
#
VERBOSE = 0                             # Write extra information to errorLog
```

Slik vi ser er det nær slutten av filen en innstilling kalt "MAX_POS_REGS_PER_TRACK". Denne er satt til 25, den minste mulige verdien støttet av systemet. Denne verdien forsikrer at en overflyvning aldri vil inneholde mer enn 25 punkter, slik at mengden data blir begrenset.

Dette påvirker imidlertid bare nye data, da det som allerede er logget til databasen vil beholde den "oppløsningen" det hadde fra før. Derfor ble det nødvendig å tynne ut databasen selv etter at denne forandringen hadde blitt implementert i systemet. Dette er foklart i 4.3.1.

Til sist må også "LAST_POS_TIMEOUT" nevnes. Slik beskrevet i filen er dette tidsintervallet systemet venter før systemet antar at flyet har forsvunnet fra sensoren, og overflyvningen blir konkludert og sendt til databasen.

## 4.2 Database

Databasen var en sentral del av vår oppgave, da hele vår løsning måtte tilpasses formatet dataene var definert på. I dette kapittelet gir vi defor en grundig oversikt over hvordan databasen er definert, samt hvordan datafeltene blir opprettet.

Databasen er delt inn i rader og kolonner. En rad representerer et punkt på en overflyvning (en flyobservasjon), med de assosierte dataene. Et fly vil altså ha en høyde, en posisjon, en rotasjon, etc. ved en gitt logging til databasen. Disse datafeltene utgjør så kolonnene i databasen, som er den samme for alle overflyvninger.

Et utsnitt fra databasen finner du i Tabell 4.1: Eksempeldata, ulike fly, her er hver rad et øyeblikksbilde fra et enkelt fly fanget opp av sensoren i Hessdalen. Flere slike rader/flyobservasjoner oppgjør så en overflyvning av et bestemt fly.

Tabell 4.1: Eksempeldata, ulike fly

| hexident | postime | flightid | latpos | longpos | track | speed | altitude | verticalrate | incam | Regtime |
|---|---|---|---|---|---|---|---|---|---|---|
| 4CC2AD | 1398765590 | ICE306 | 60.86649 | 10.36756 | 103 | 486 | 37000 | -64 | 0 | 2014-04-29 12:02:10 |
| 4CC2AD | 1398765606 | ICE306 | 60.85695 | 10.43888 | 103 | 487 | 37000 | -64 | 0 | 2014-04-29 12:02:10 |
| 4CC2AD | 1398765617 | ICE306 | 60.85085 | 10.48806 | 103 | 487 | 37000 | 64 | 0 | 2014-04-29 12:02:10 |
| 47A619 | 1398765228 | NAX375 | 62.54672 | 11.91427 | 176 | 450 | 35975 | 64 | 0 | 2014-04-29 11:59:10 |
| 47A619 | 1398765239 | NAX375 | 62.52328 | 11.91762 | 176 | 452 | 35975 | 128 | 0 | 2014-04-29 11:59:10 |
| 47A619 | 1398765250 | NAX375 | 62.50108 | 11.92078 | 176 | 451 | 36000 | 64 | 0 | 2014-04-29 11:59:10 |

Her ser vi altså et lite utsnitt av to ulike overflyvninger oppdaget av flydeteksjonssystemet. De første 3 radene omhandler et fly med id (hexident) 4CC2AD. Dette er altså den fysiske adressen på senderen som gir ut disse meldingene, og som er plassert inne i flyet. Postime er tidspunktet (i unix time) da flyet sendte ut denne meldingen, med den relaterte informasjonen. Vi ser her at tre av disse meldingene forekom med korte mellomrom, det var bare snakk om noen få sekunder. Deretter ser vi en ny overflyvning, denne gangen et fly med id 47A619.

Innholdet i de ulike kolonnene er beskrevet i detalj under, med eksempeldata tatt fra den nederste raden i figuren over.

**Hexident:** 47A619

- Dette er den fysiske adressen til senderen plassert inne i flyet. Dette er en helt unik identifikator som ikke vil bli brukt av noen andre fly (med untak av sendere som fysisk flyttes til andre fly), derfor blir dette indirekte en identifikator på selve flyet.
- Denne vil alltid bestå av 6 heksadesimale tall, og er påkrevd av flydeteksjonssystemet. Hvis dette datafeltet ikke kan tolkes fra meldingen sendt ut fra flyet, vil meldingen bli forkastet.
- Hexident er således også en del av primærnøkkelen i databasen, altså et datafelt som er med på å gi en unik identifikator på en rad i databasen. Dette impliserer også at dette datafeltet MÅ spesifiseres (det kan ikke være *null*). Altså er dette et meget pålitelig datafelt.

**Postime:** 1398765250

- Postime er tidspunktet da flyet sendte ut denne spesifike meldingen med de relaterte dataene.
- Dette er et heltall som oppgir tidspunktet i unix time, altså antall sekunder fra klokken 00:00:00 den 1. januar 1970 UTC (/GMT).

- Postime er den andre halvdelen av primærnøkkelen, sammen med hexident. Dette betyr altså at tidspunktet for en melding sendt ut av et fly og hexidenten for flyet, sammen er unikt og identifiserende for en rad i databasen.
- Postime er del av primærnøkkelen, og er derfor et pålitelig datafelt som uten problem kan brukes i vårt system.

**Flightid:** NAX375

- Dette er en identifikator på en overflyvning, og fungerer som et slags rutenummer. Lengden på flightid varierer fra 3 til 8 tegn, der de første 3 tegnene ofte representerer et flyselskap slik som "NAX" og "SAS".
- Flightid blir tildelt av flyselskapet som har ansvar for flyet, og det er til en viss grad identifiserende for en overflyvning.
- Da vi begynte å undersøke mulighetene for gruppering av fly viste det seg imidlertid at dette datafeltet ikke var pålitelig nok. Det største problemet med dette datafeltet var at det ikke var veldefinert på samme måte som hexident. Flightid kunne nemlig være en tom tekststreng (""). Altså helt uspesifisert. Dette gjorde at denne kolonnen ikke kunne tas i bruk når det gjaldt behandling av data, og ville dermed bare bli brukbar når det gjaldt direkte fremvisning til sluttbrukeren.

**Latpos og Longpos:** 62.50108, 11.92078

- Dette var flyets latitude (breddegrad) og longitude (lengdegrad) ved det angitte tidspunktet (postime).
- Disse datafeltene var helt kritisk for utformingen av vår oppgave, da det ville blitt umulig å plassere ikoner på kartet hvis det manglet koordinatdata. Derfor måtte alle data som ikke hadde brukbare koordinater forkastes av vårt system.
- Merk at databasen støttet relativt høy nøyaktighet for disse datafeltene, men at disse dataene aldri ble spesifisert med høyere enn fem desimalers nøyaktighet.
- Databasen tillater ikke at disse verdiene blir satt til *null*, men det var forekomster av flyttalsverdier meget nære 0 (j.fr. Track). Verdier som dette filtrerte vi ut i vårt system.

**Track:** 176

- Track, med sitt noe missvisende navn, er flyets orientering (/rotasjon) ut fra 360 grader. 0 grader tilsvarer nord, mens 90 grader er øst. 360 grader blir aldri brukt av systemet, da dette isteden vil være 0 grader.
- Dette, i likhet med de neste tre datafeltene, var ikke alltid tilgjengelig, slik at verdien ble satt til 0 hvis de manglet. Merk at dette overlapper med fly som faktisk hadde en vinkel på 0 grader.

**Speed:** 451

- Dette oppgir flyets fart.
- I likhet med track blir dette satt til 0 hvis disse dataene ikke var tilgjengelige. Merk at dette ikke er en problemstilling i vår oppgave, da et fly aldri vil ha en fart lik 0, da ville flyet ha stått stille. Dette er ikke mulig utenom tider der flyet står på bakken (utenfor rekkevidden til flydeteksjonssensoren).

**Altitude:** 36000

- Dette er flyets høyde, gitt i *meter over havet*.
- Denne verdien ble satt til 0 hvis dataene var utilgjengelige. På samme måte som Speed, var ikke dette en problemstilling. En høyde lik 0 meter kunne bare forekomme hvis flyet sto på bakken ved hav-nivå, eller hvis det hadde nødlandet i havet.

**Verticalrate:** 64

- Verticalrate oppgir flyets stigning, der en positiv verdi betyr at flyet øket i høyde.
- Denne verdien ble satt til 0 hvis dataene ikke var tilgjengelige. Dette var relativt problematisk å løse, da en verticalrate på 0 betyr at flyet flyr rett frem, noe fly gjør store deler av tiden. Altså var det nærmest umulig å avgjøre om denne verdien var 0 grunnet en feil, eller om dette var fordi flyet faktisk fløy rett frem.

**Incam:** 0

- Dette er en binær verdi som angir om flyet var innenfor synsfeltet til kameraet i Hessdalen eller ikke.
- Dette blir bestemt utifra statiske verdier som oppgir et todimensjonalt område i Hessdalen. Området er definert ved hjelp av følgende verdier:
    - Longpos mellom 9.8 og 10.5
    - Latpos mellom 60.0 og 61.0
    - Høyde blir ikke tatt hensyn til

**Regtime:** 2014-04-29 11:59:10

- Regtime er tidspunktet da data om en overflyvning ble lagret til databasen. Dette blir generert av funksjonen NOW() i mysql. Dette ble beskrevet i 3.3.6.3.
- Databasen tar det nåværende tidspunktet i systemet og setter dette inn i regtime kolonnen. Merk at dette er tidspunktet når henvendelsen til databasen *forekom*, og ikke når dataene *faktisk* ble innsatt. Dette betyr at selv om databasen kan bruke tid på å få bearbeidet dataene, vil alle forespørsler som ble sendt samtidig bli tildelt samme regtime-verdi.Dette betyr at når flydeteksjonssystemet setter inn data for en overflyvning (dette skjer som en sammenhengende operasjon etter at hele overflyvningen er ferdig) vil samme regtime verdi bli gitt til alle radene.
- En svakhet ved dette systemet er hvordan det konkluderer en overflyvning. Systemet venter til et predefinert tidsintervall (se "LAST_POS_TIMEOUT" i Figur 4.2: Konfigurasjonsfil, flydeteksjonssystem) har passert etter siste logging, og konkluderer overflyvningen når tiden løper ut. Problemet oppstår når flyet bare har vært usynlig for sensoren i en kort periode. Dette vil føre til at to ulike regtime verdier vil bli produsert for et fly som egentlig er en og samme overflyvning. Dette er beskrevet i 3.3.6.3.

Til sist skal det nevnes at de fleste av disse kolonnene (med unntak av der det motsatte var spesifisert) blir tillat å være *null* (ikke definert) av databasen. Dette kunne ha vært problematisk for vårt system da vårt system mest sannsynlig ville ha prøvd å behandle disse som faktiske tall- eller tekstverdier. Dette viste seg imidlertid å ikke være et problem, da flydeteksjonssystemet aldri ga verdien *null* til et datafelt (det ble heller satt til 0 eller en tom tekststreng).

All prosessering av data som foregår i databasen blir beskrevet i 4.3, da det er webserveren som initierer dette.

## 4.3 Backend - Kode på webserveren

Vårt system opererer både på sluttbrukerens maskin, webserveren, og i databasen (via SQL). I dette kapittelet tar vi for oss hvordan koden på webserveren fungerer, samt hvordan denne arbeider mot databasen. Før selve systemkoden forklares skal vi beskrive maint.php, som var et program vi laget for uttynning av databasen, et tema som har vært nevnt flere ganger tidligere. For en full oversikt over hvordan de ulike delene av systemet arbeider samme, og den totale dataflyten, se 4.5.

### 4.3.1 Databasetynning

I 4.1.3 beskrev vi hvordan flydeteksjonssystemet ble modifisert for å unngå at unødvendige mengder data ble lagret. Konklusjonen var at systemet inneholdt en konfigurasjonsparameter som ga en øvre grense for hvor mange punkter som skulle lagres per overflyvning. Denne grensen ble satt til 25, laveste grense som var mulig i flydeteksjonssystemet. Dette måtte så reflekteres for dataene lagret i databasen, dette skal vi ta for oss her.

maint.php var et skript vi utviklet for å tynne databasen. Dette skriptet ble kjørt bare en enkelt gang, og resultatet var at alle data i databasen ble normalisert, slik at de aldri ville inneholde mer enn 25 rader per overflyvning. For å få til dette var gruppering av fly nødvendig, dette er beskrevet i 4.3.2.

maint.php fungerte slik at alle data ble hentet fra databasen, gruppert i overflyvninger, og deretter bearbeidet.

Hvis en overflyvning inneholdt færre punkter enn den valgte grensen (25) ville hele overflyvningen bli bevart. Hvis det var flere punkter, ville imidlertid algoritmen bestemme hvilke punkter som skulle beholdes, og alle andre ble slettet.

For å forklare hvordan skriptet bestemte hvilke punkter som skulle beholdes, tar vi for oss et eksempel:

Hvis man har 200 rader som man ønsker å beholde 4 av (for enkelhets skyld), tar man $\frac{200}{4} = 50$. Man tar vare på hver femtiende rad, dette blir radene: 1, 50, 100 og 150. Vi ser at radene 150 til 200 blir helt ignorert. Dette er ikke en tilfredsstillende jevn distribusjon.

Trikset er simpelt, man deler på antall ønskede rader minus 1, slik: $\frac{200}{4-1} = \frac{200}{3} = 66.66$. Altså får man nå med rad 1, 66, 132 og 198 (merk at desimalene tas vekk, i motsetning til å runde av tallet). Vi tar så bort det siste elementet og heller bestemmer at dette skal være rad 200, altså det helt siste elementet, alltid. Vi får deretter et utvalg som er slik vi ser i Figur 4.3: Jevnt utvalg (4 valgte punkter ut fra totalt 200).

Figur 4.3: Jevnt utvalg

Vi får en jevn distribusjon, i motsetning til slik vi ser i Figur 4.4: Ujevnt utvalg.

Figur 4.4: Ujevnt utvalg

Vi ser på den nederste versjonen at det er 4 punkter, med endepunktene, men det er ikke jevnt distribuert. Vår løsning tok hånd om dette slik at det alltid ble generert et konstant antall elementer, som alltid var jevnt fordelt.

Etter at de ønskede punktene var valgt ut, kunne alle andre punkter slettes. Måten vi utførte dette på var *ikke* en direkte forespørsel mot databasen, da dette potensielt ville få katastrofale følger under programtesting. Det vi heller bestemte oss for å gjøre var at eventuelle rader som skulle slettes, altså rader som IKKE ble valgt, produserte en såkalt DELETE-statement. Denne kunne for eksempel se slik ut:

```
DELETE FROM flightdata WHERE hexident="06A052"AND postime=1370085750;
```

Vi hentet altså ut hexident og postime for radene som skulle slettes, da disse var unike og identifiserende verdier for en rad (se 4.2). Deretter slettet vi radene en av gangen.

Merk at disse utskriftene ble plassert i en såkalt .sql-fil. Altså en fil som inneholdt en liste med slike database-kommandoer, som vi så kunne kjøre direkte i databasen da filen var ferdig generert.

Etter å ha kjørt denne filen ble antall rader omtrent halvert fra ca. 600 000 til ca. 300 000. Det vil si at ca. 300 000 DELETE-statements ble generert. Det viste seg problematisk å faktisk få kjørt dette, da det ville bli en enormt tidskrevende prosess. Vi prøvde først å åpne denne i MySQL Workbench, og kjøre den derfra, men dette viste seg å gå ekstremt sakte. Hver enkelt-setning ble overført og eksekvert en-etter-en. Dette ville ha tatt flere dager. Derfor bestemte vi oss heller for å kjøre dette ved help av MySQLs command line verktøy. På denne måten fikk vi kjørt .sql-filen på få timer.

### 4.3.2 Gruppering av fly

En annen viktig del av vårt system var hvordan vi skulle få gruppert fly i en overflyvning. Metoden vi kom frem til var å bruke en sammensetning av databasekolonnene Hexident og Regtime. Hexident var en unik identifikator for et gitt fly, Regtime var tidspunktet da en overflyvning ble lagret i databasen. Derfor ville disse kolonnene til samme identifisere en unik overflyvning. Det skal nevnes at Regtime nesten var tilstrekkelig i seg selv, men da det ikke var utenkelig at to ulike fly kunne fly ut fra rekkevidden til sensoren på nøyaktig samme tid, måtte vi også bruke Hexident for å skille disse tilfellene.

Grupperingen foregikk slik at uthentingene fra databasen ble sortert på hexident i førsterekke, og deretter på regtime. Dette ble altså behandlet i selve databasen.Vi fikk en forløpende oversikt over hvert enkelt unike fly, som videre var sortert i forhold til tidspunktene dette hadde vært i nærheten av sensoren. Deretter var det trivielt å gruppere flyene.

Radene ble løpt igjennom en etter en. På grunn av sorteringen ville alle like hexidents komme rett etter hverandre. Hver nye hexident programmet kom over ville opprette en ny gruppe (for en overflyvning), og hver nye regtime innenfor den samme hexidenten ville på samme måte også opprette en ny gruppe. Regtime-verdier som var innenfor en gitt grense, med tanke på unøyaktigheter i systemet, ville bli plassert i samme gruppe.

Hver gang en ny gruppe ble opprettet, ble den gamle gruppen konkludert og konvertert til JSON, som så ble sendt til klienten. Formatet på dataene er beskrevet i 4.3.3.

Unøyaktighet i regtime verdier, slik nevnt over, måtte vi ta hensyn til under gruppering. Regtime kunne potensielt kunne inneholde små variasjoner for samme overflyvning. Dette ville skje i situasjoner der lagringen av rader til databasen skjedde på et tidspunkt der systemklokken tikket over til et nytt sekund i løpet av innsettingen, eller flyet var usynlig for sensoren i et lite tidsrom. Dette var usannsynlig, men det var et hensyn vi valgte å ta for å være på den sikre siden. Vi løste dette enkelt ved å si at en regtime som var innenfor en predefinert grense (konfigurerbar) ble slått sammen, slik at selv punkter med litt varierende regtime ble oppført under samme overflyvning. Dette gjaldt selvsagt bare punkter med samme hexident.

### 4.3.3 JSON

Vårt system overførte data mellom webserveren og klienten i form av JSON, et utsnitt av dette ser du i Eksempel 4.1: JSON.

**Eksempel 4.1: JSON**


```
{"flightdata":[{
    "planeID": "478481",
    "flightID": "NAX758",
    "regtime": 1398781587,
    "sightingCount": 2,
    "sightings": [
        {
            "speed": 336,
            "altitude": 21250,
            "lat": 62.6667,
            "lng": 11.5918,
            "time": 1398781512,
            "rotation": 2,
            "ascent": -2048
        },
        {
            "speed": 336,
            "altitude": 21250,
            "lat": 62.6667,
            "lng": 11.5918,
            "time": 1398781512,
            "rotation": 2,
            "ascent": -2048
        }
    ]
}], "flightCount": 1, "sightingCount": 2, "rowCount": 2}
```

Slik vi ser av denne utskriften omhandler det en overflyvning med to punkter for den gitte overflyvningen (dette er et kraftig forenklet eksempel). I tillegg er generell informasjon om utvalget presentert som separate datafelter, i slutten av utskriften. Noe som også skal nevnes er at denne utskriften blant annet inneholder et felt kalt "rotation". Dette er altså "track" fra databasen, men gitt et nytt og mer forståelig navn. Oversikten over dette er gitt i 4.3.4.

En detalj som er verdt å nevne er at måten vår kode fungerer på er at JSON-koden blir skrevet ut kontinuerlig, hver gang en overflyvning er ferdig prosessert. Dette er da i motsetning til å gjøre ferdig all data, for så å sende alt samtidig. Ved å sende det i små deler, forløpende, løste vi mange problemer med de store mengdene data som lå i databasen, da de ikke ville bli aggregert noe annet sted enn på klientmaskinen.

Det skal også nevnes at vår kode gjorde det mulig å hente ut en oversikt over utvalget ved å utføre en såkalt "probing". På denne måten fikk man oversikt over hvor mye data dette utvalget ville medføre (uten å utføre selve datauthentingen), slik at man kunne gjøre om på utvalget hvis dette viste seg å bli for mye.

### 4.3.4 Omdøping av kolonnenavn

De ulike kolonnene i databasen inneholder navn som potensielt kan være forvirrende og uforståelige for sluttbruker. Disse ble derfor omdøpt til navn som var mer gunstige. En oversikt over disse finner du her (databasenavn til venstre, vårt navn til høyre):

- hexident => planeID: En unik identifikator på flyet basert på senderen som står i flyet.
- postime => time: Tidspunktet for når senderen sendte ut denne datameldingen.
- flightid => flightID: Nummeret for den aktuelle "flighten", gitt stor bokstav på ID for å matche PlaneID.
- latpos => lat: Breddegraden flyet befant seg ved.
- longpos => lng: Lengdegraden flyet befant seg ved.
- track => rotation: Rotasjonen/orienteringen til flyet, altså hvilken retning det fløy i.
- speed: Ingen forandring.
- altitude: Ingen forandring.
- verticalrate => ascent: Flyets stigning.
- regtime => UNIX_TIMESTAMP(regtime): beholdt samme navn, men forandret formatet til å være et unix timestamp, altså samme som postime.

## 4.4 Frontend - Kode på klientmaskinen

Data blir hentet fra databasen og behandlet på webserveren, deretter blir det sendt videre til klientmaskinen. Her utføres også gjennstående prosesseringer, samt fremvisning av resultatet for sluttbruker. I 4.4.1 gir vi en oversikt over koden som kjører på klientmaskinen, mens 4.4.2 gir en oversikt over hvordan GUI-et ser ut, og hvordan det brukes.

### 4.4.1 Code behind

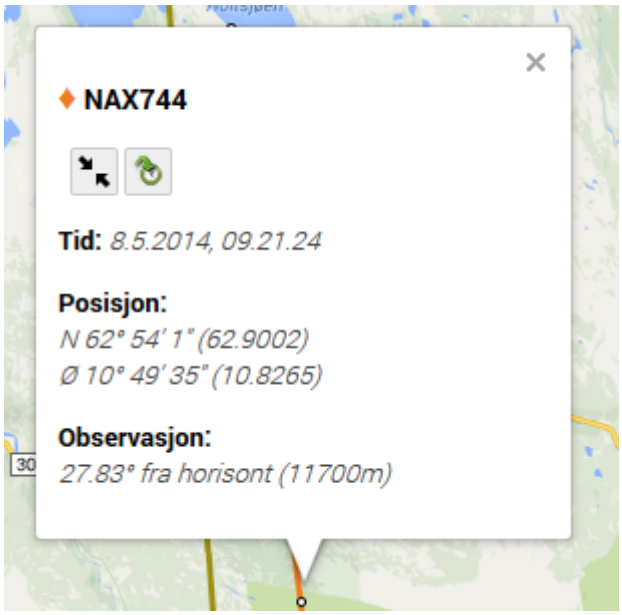
Koden som kjører på klientmaskinen har helt andre forutsetninger enn det som kjører på webserveren. Mens vi på webserveren må ta hensyn til eventuelt stress ved stor pågang, må vi på klientmaskinen ta hensyn til at datamaskinen er kraftig nok til å gi brukeren en responsiv og stabil

opplevelse. På den annen side er det ikke like kritisk å ta hensyn til store arbeidsoppgaver som brukeren initierer, da dette bare vil gå utover brukeren selv, i motsetning til at det går utover webserveren (og alle andre brukere).

#### 4.4.1.1 *Informasjonsvinduer og flyikoner*

Dataene som hentes ut må presenteres for sluttbrukeren på en gunstig måte. Vi har valgt å gjøre dette ved å plassere ikoner for alle overflyvninger på kartet, sammen med alle datapunktene for overflyvningen hvis bruker ønsker dette. Begge disse kan klikkes på for å presentere de assosierte dataene for brukeren i et informasjonsvindu / popup boble. Et eksempel på dette ser du i Figur 4.5: Informasjonsvindu.

Figur 4.5: Informasjonsvindu




Merk at dette er et simplifisert informasjonsvindu som ikke inneholder alle data fra databasen. Merk også de to ikonene øverst. Disse er ekstrafunksjoner som blir beskrevet nærmere i 4.4.2.

Dette informasjonsvinduet blir åpnet ved å trykke på enten et fly-ikon, eller ved å trykke på en "vei-node" som representerer et datapunkt fra databasen, alle disse danner så til samme den totale overflyningen.

Informasjonen som vises frem i dette vinduet blir utformet av en løsning vi selv laget, som gjør det er relativt lett å forandre innholdet i informasjonsvinduet. Dette gjøres ved hjelp av tekst som inneholder små kodesnutter. Disse tolkes av vårt program når teksten behandles, med mulighet for å blant annet utføre konverteringer og liknende der dette er relevant. Et eksempel på en slik tekststreng (den som genererte informasjonsvinduet i Figur 4.5: Informasjonsvindu) er vist i Eksempel 4.2: Syntaks for informasjonsvindu. Dette plasseres i egne filer, en for hver type informasjonsvindu.

**Eksempel 4.2: Syntaks for informasjonsvindu**

```
<div class="infoWnd">
    <p class="header">
        <span class="flightSymbol" style="color: $F.color">&diams;</span> {printOrDefault
$F.flightID; (Ingen ID)}
    </p>

    <p class="controls">
        <img src="img/narrow.png" class="actionImage" title="Begrens tidsrom til denne
avgangen" onclick="selectRange({_flightRange $F.sightings})">
        <img src="img/gototime.png" class="actionImage" title="Gå til tidspunkt"
onclick="selectInstant({date $S.time; unix})">
    </p>

    <p>
        <span class="name">Tid:</span> <span class="value" title="$S.time">{date
$S.time}</span>
    </p>

    <p>
        <span class="name">Posisjon:</span> <br/>
        <span class="value">
            {convertPos $S.lat; N; S} ({round $S.lat; 4})
            <br/>
            {convertPos $S.lng; Ø; V} ({round $S.lng; 4})
        </span>
    </p>

    <p>
        <span class="name">Observasjon:</span> <br/>
        <span class="value" title="%S:angle%">{round $S.angle; 2}° fra horisont ({round
$S.altitude; 0}m)</span>
    </p>
</div>
```

Uten å gå i for stor detalj ser vi at dette er vanlig HTML, som spesifiserer innholdet i informasjonsvinduet. Unntaket er linjer som denne:

```
<span class="flightSymbol" style="color: $F.color">&diams;</span> {printOrDefault
$F.flightID; (Ingen ID)}
```

Denne linjen spesifiserer fargen på diamanten øverst til venstre, som er fargen til overflyvningen. Her brukes nøkkelordene Flight (overflyvning) og Sighting (flyobservasjon) i form av **`$F`** og **`$S`**. En flight består av flere sightings. Fargen til diamanten blir altså bestemt av Flight-objektet sin Color, **`$F.color`**.

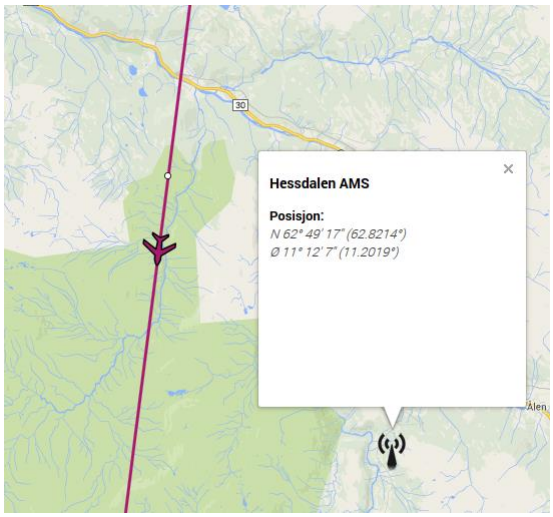
Deretter ser vi at funksjonen **`printOrDefault`** skriver ut flyets FlightID, **`$F.flightID`**. Hvis denne ikke er tilgjengelig brukes *default* verdien, her bestemt til å være "**`(Ingen ID)`**". **`printOrDefault`** er altså en funksjon i vårt program som blir kalt når navnet blir oppdaget i denne strengen. Flere slike funksjoner sees i eksempelet over, slik som **`round`**, **`convertPos`** og **`date`**.

Flere datatransformer er definert, og formatteringen på det som skulle vises i informasjonsvinduene er lett å utføre ved å sette opp de ønskede dataene (med henvisninger til Flight eller Sightings, henholdsvis), samt en referanse til funksjoner/transformer der dette er relevant.

#### 4.4.1.2 AMS

Selve basestasjonen, Hessdalen AMS, ble inkludert på kartet for å lettere kunne orientere seg. Dette er illustrert i Figur 4.6: AMS ikon.

Figur 4.6: AMS ikon




Av figuren ser vi AMS ikonet med tilhørende data. Ved å plassere ut dette ble det blant annet lettere å tolke dataene fra fly i området, slik som høyde og vinkel mot basestasjon. På denne måten ble det også lettere å se om terreng ville blokkere fri sikt mot flyet, osv.

Dette ikonet fungert også som sentrum for kartet, og spesifiserte standard-utsnittet av kartet som brukeren så da websiden ble åpnet, igjen for å gjøre det lettere for brukeren å orientere seg.

#### 4.4.1.3 Datauthenting

Selve uthentingen fra databasen blir prossesert på webserveren og konvertert til JSON. Klientmaskinen tar så imot disse dataene og plotter disse på kartet. Dette er en relativt grei operasjon som ikke krever større utdyping, men likevel er det en viktig detalj ved dette som må belyses. Mens data hentes ut skal disse plottes på kartet. Dette er en relativt krevende prosess, da det er snakk om store mengder data. Kartet kunne bli uresponsivt, og i værste fall kræsjet hele nettleseren.

Måten vi løste denne problemstillingen på var relativt simpel. Mellom hver overflyvning som skulle plottes på kartet tok systemet en pause på ett millisekund. Dette ga systemet tid til å rendere (/tegne) selve kartet, bearbeide data, og utføre eventuelle andre arbeidsoppgaver (som ikke ville blitt behandlet hvis vi ikke tvang vårt program til å ta en kort pause, da alt av JavaScript blir kjørt synkront). Dette gjorde at selve prosessen tok lenger tid, fordi systemet nå ventet i et lite øyeblikk før det gikk videre med neste overflyvning.

Denne tidsforsinkelsen økte lineært med antall overflyvninger. Ved for eksempel en måned (ca. 6000 overflyvninger) ville systemet altså **minst** ta 6 sekunder lenger. Likevel ville dette i praksis gjøre at systemet ble raskere, da systemet ikke lenger hang seg. I tillegg ville ikoner bli innført på kartet litt-etter-litt, i motsetning til at **alt** ble fremstilt samtidig etter flere sekunder med uresponsivitet.

Det skal også nevnes at før selve uthentingen fant sted, ble en liten forespørsel sendt til databasen for å se hvor mye data som utvalget ville medføre. Hvis dette utvalget var helt tomt (og større enn 24 timer) ble brukeren presentert med en dialog som informerte om at systemet kunne ha vært offline i den aktuelle perioden. Flytrafikken over Hessdalen så sjeldent mer enn en time uten trafikk, derfor var det greit å anta at systemet var offline hvis det gikk lang tid uten flytrafikk. En liste med datoer der det var kjent at systemet var offline ble også presentert til brukeren i denne situasjonen. Denne listen er som følger:

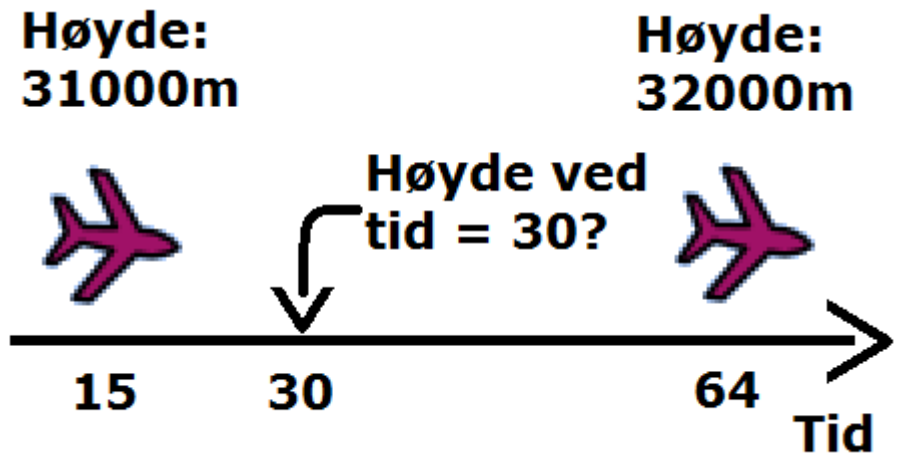
- 19.05.13 til 22.05.13
- 10.06.13 til 08.08.13
- 18.08.13 til 13.01.14
- 16.01.14 til 24.01.14
- 27.01.14 til 20.02.14

Brukeren fikk på samme måte også informasjon om usedvanlig store utvalg, da dette kunne medføre kraftig last på klientmaskinen. Antall overflyvninger som ville bli uthentet ble presentert for brukeren.

#### *4.4.1.4 Interpolasjon og animasjon*

Da vi bestemte oss for at systemet skulle tillate et dynamisk tidsutvalg (et "øyeblikk"), ble det nødvendig med interpolasjon. Dette lot oss finne verdier for flyobservasjoner plassert mellom to av punktene fra databasen. Dette er relativt simpel matematikk hvis det er snakk om lineær interpolasjon. Vi valgte å ikke gå for en mer avansert form, da lineær interpolasjon var mer en godt nok for vårt bruksområde. Et eksempel på dette ble gitt i 3.2.4.6, vi inkluderer dette eksempelet igjen, for enkelhets skyld (se Figur 4.7: Interpolasjonsoppgave 2):

Figur 4.7: Interpolasjonsoppgave 2




Problemstillingen er altså at vi har to sett med data, og ønsker å finne verdien på et punkt mellom disse. Fremgangsmåten er som følger:

Hvis høyden går fra å være 31000m til 32000m vil det si at det skjer en forandring på 1000 meter. Denne forandringen foregår i et tidsrom på 64 - 15 = 49 tidsenheter (benevningen er irrelevant). Tiden der vi ønsker å finne høyden, 30, er 15 sekunder etter starttidspunktet.

Altså vil vi vite hva høyden er ut av totalt 1000 meter på et tidspunkt som er 15 ut av totalt 49 tidsenheter. Dette er triviell matematikk:

$$\frac{X}{1000} = \frac{15}{49} => X = \frac{15}{49} * 1000 = 306$$

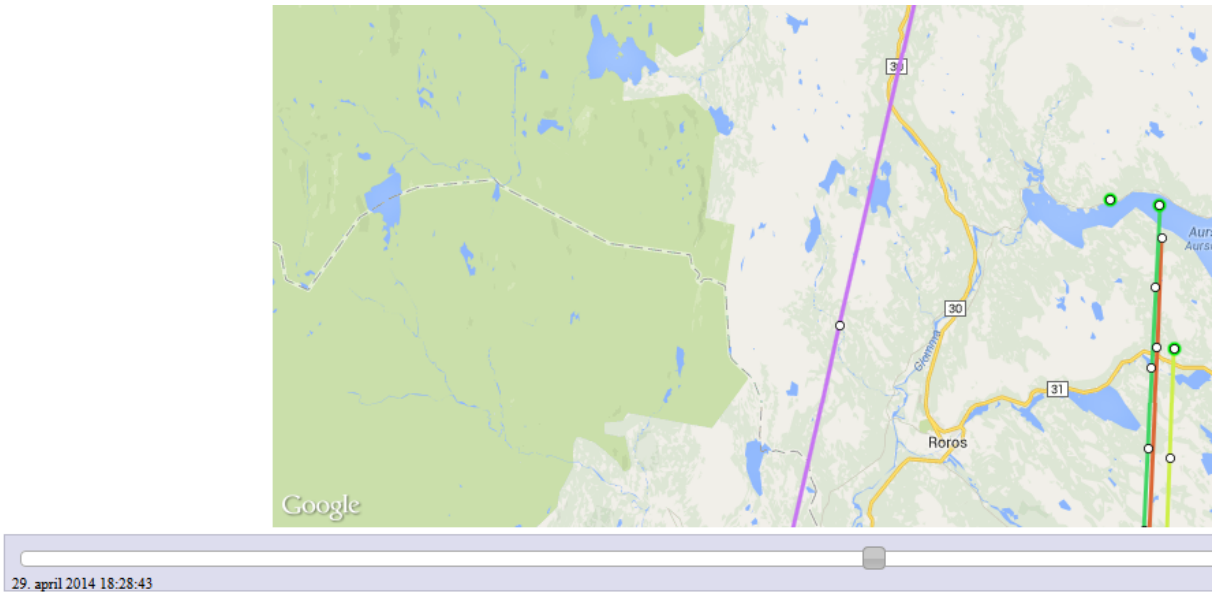
Altså er flyet i en høyde av 31000 + 306 meter på det angitte tidspunktet, 31306 meter.

Denne metoden kan så brukes for å regne ut alle de ulike datafeltene på en flyobservasjon plassert mellom to andre.

Koden som utfører denne interpolasjonen er relativt grei, og krever ingen dyptgående forklaring. En problemstilling må imidlertid nevnes, og det er at datafelter, slik beskrevet i 4.2, kan ha blitt satt til 0 grunnet mangel på informasjon. Dette vil gjøre at interpolasjonen vil gi potensielt misvisende verdier. Det kan for eksempel se ut som om et fly er på vei til å lande hvis høyde har blitt satt til 0, og flyet sakte beveger seg mot denne verdien (uten at dette blir reflektert i verdien som angir flyets stigning). Dette var det desverre lite å gjøre med, uten å "pusse" på dataverdiene. Dette kunne ha gitt brukeren et feil bilde av overflyvningen. Derfor var det bedre å la feildataene være som de var. Brukeren hadde også tilgang til alle punktene på overflyvningen, derfor var det enkelt for brukeren å selv verifisere dataene interpolasjonen var basert på.

Interpolasjon ble kontrollert av en tidslinje som brukere selv kunne velge ut et "øyeblikk" fra. Dette utførte så de nødvendige interpolasjonene og flyttet flyikonene til de nye koordinatene (med interpolerte dataverdier). Et eksempel på denne tidslinjen ser du på Figur 4.8: Tidsmanipulator.

**Figur 4.8: Tidsmanipulator**



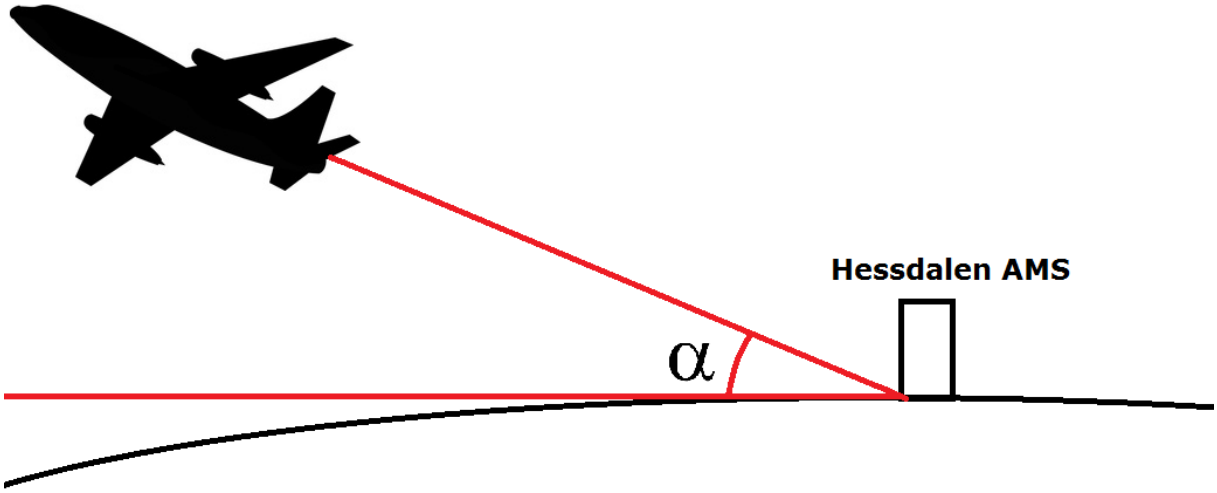

Slik vi ser på figuren er et øyeblikk (29. april 2014 18:28:43) valgt på tidslinjen. Ingen fly er synlige på denne delen av kartet i dette "øyeblikket". Alle overflyvninger vil bli interpolert på bakgrunn av dette tidspunktet, og flyikonene vil bli plassert på de korrekte koordinatene på kartet, med nye, utregnede verdier. Altså ville påvirkning av tidslinjen direkte flytte ikonene på kartet. Det vil si at animasjon enkelt kunne oppnås ved å kontinuerlig flytte tidslinjen med jevne mellomrom. Det viste seg

imidlertid at animasjon ikke var av stor interesse for oppdragsgiver. Derfor bestemte vi oss for å gå bort fra dette, for å heller prioritere andre aspekter av prosjektet.

#### *4.4.1.5 Vinkel til basestasjon*

Det ble spesifikt etterspurt av arbeidsgiver at systemet viste vinkelen til basestasjonen, hvis dette viste seg å være mulig. Vinkelen det er snakk om er altså vinkelen som man må se "opp" for å se flyet når man står ved Hessdalen AMS, slik illustrert i Figur 4.9: AMS vinkel.

Figur 4.9: AMS vinkel




Vi ser at dette er vinkelen ut fra basestasjonen og opp mot flyet. Slik illustrert (sterkt overdrevet i dette bildet) vil kurvaturen til jordkloden påvirke denne utregningen. Dette er på grunn av at flyets høyde selvsagt er oppgitt rett ned mot bakken (og ikke til den røde grunnlinjen vist i figuren). Det har seg heldigvis slik at det er snakk om et relativt lite område, som gjør det mulig å behandle strekningen langs bakken som en rett linje. På denne måten kan vinkelen regnes ut med vanlig trigonometri som tar i bruk høydeforskjellen mellom AMS-en og flyet, samt avstanden mellom dem (i en tilnærmet rett linje langs jordoverflaten).

Høydeforskjellen er lett å regne ut, da flyets høyde blir sendt ut av senderen i flyet, og høyden til basestasjonen (i meter over havet) lett kan finnes ved hjelp av atlas og liknende (AMS-en er også plassert omtrent i bakkehøyde).

Problemet er å finne avstanden mellom flyet og basestasjonen i meter, da de eneste verdiene vi vet er koordinatene til begge. Hessdalen AMS er plassert vet 62°49’17” nord, 11°12’7” øst. Flyets koordinater blir sendt ut av flyets sender. Disse koordinatene er oppgitt i "grader", og sier oss ingen ting om avstanden mellom punktene i meter. En annen problemstilling er igjen jordens kurvatur.

Måten denne avstanden så regnes ut på er ved hjelp av en formel som tar dette med i betraktning og produserer en avstand i meter som vi kan bruke direkte i vår løsning. Dette er den såkalte "haversine"-formelen som produserer "storsirkel" avstanden mellom to geografiske punkter (Movable Type Ltd, 2014). Denne formelen finner du i Formel 4.1: Haversine.

Formel 4.1: Haversine

$$a = \sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos(\varphi 1) * \cos(\varphi 2) * \sin^2\left(\frac{\Delta\lambda}{2}\right)$$
$$c = 2 * \operatorname{atan2}\left(\sqrt{a}, \sqrt{1-a}\right)$$
$$d = R * c$$

Der $\varphi$ er breddegrad, λ er lengdegrad og R er jordens radius (6.371 km). Merk at vinklene som blir brukt i disse funksjonene må være oppgitt i radianer, samt at denne funksjonen regner ut avstanden i kilometer, der vi bruker meter i vårt program.

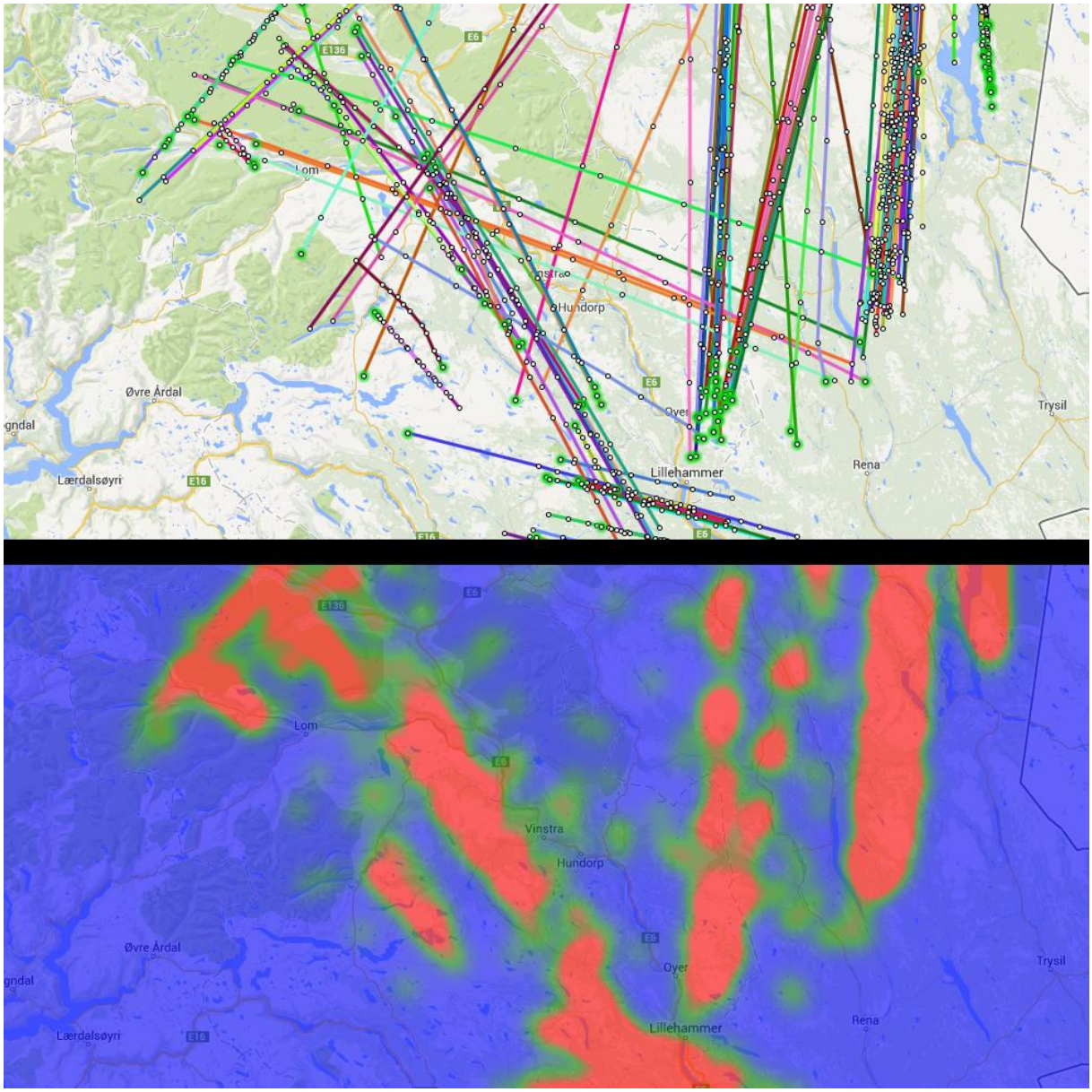
Etter å ha regnet ut denne vinkelen settes denne så inn som et eget datafelt for en overflyvning.

Denne utregningen kunne også ha vært utført på webserveren, men vi valgte å ikke overlate mer prosessering til denne enn den allerede hadde, slik at klientmaskinen måtte ta på seg dette arbeidet.

#### *4.4.1.6 Heatmap*

På grunn av svakheter i flydeteksjonssystemet vist det seg nødvendig å gi brukeren en mulighet for å se sannsynligheten for fly, selv om systemet ikke hadde gjort noen registreringer i det aktuelle området. Dette implementerte vi som en heatmap, altså et slags filter som ble lagt over kartet, som viste intensiteten av fly ved help av farger. Et eksempel på en heatmap, og data dette ble basert på, ser du i Figur 4.10: Heatmap med kilde.

Figur 4.10: Heatmap med kilde




Slik vi ser av denne figuren angir heatmap-en en overordnet oversikt over hvor det var størst forekomster av fly. Ved hjelp av dette er det mulig å anslå sannsynligheten for at et fly passerte et område, selv hvis ingen data var logget på det aktuelle tidspunktet. Dette gjør også at det blir lettere å fremstille store mengder data. Grunnen til dette er at genereringen og tegningen av en heatmap er betraktelig mindre krevende for datamaskinen, enn det er å tegne opp et ikon for alle observasjoner, samt linjer som knytter disse sammen.

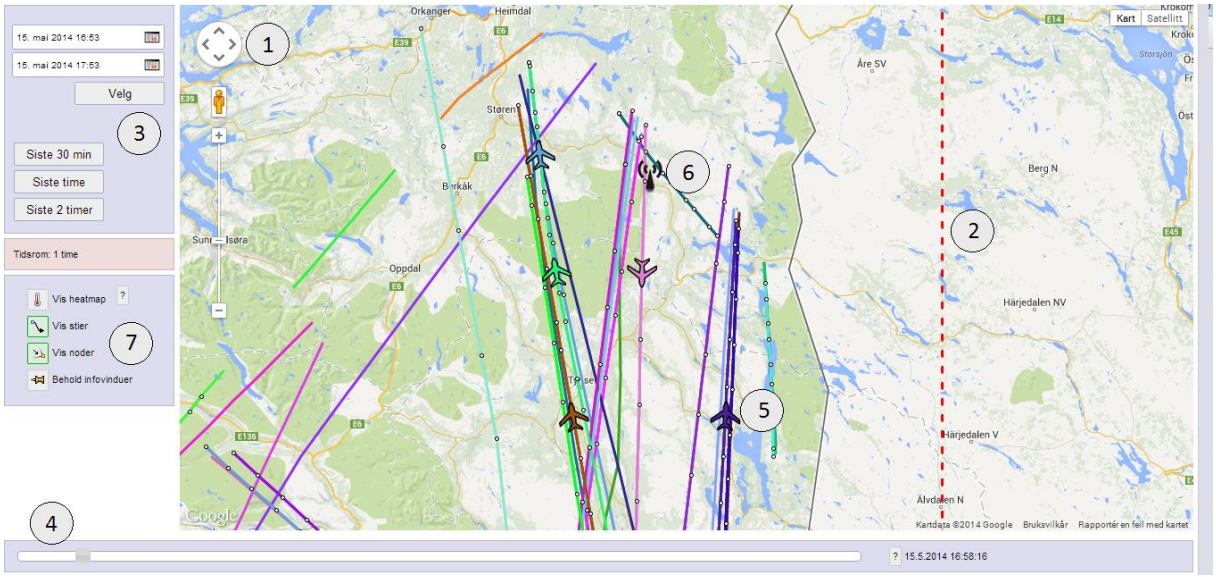
Da brukeren selv bestemmer tidsintervallet som heatmap-en skal baseres på, vil dette også si at brukeren selv har kontroll på hvor spesifikt det blir. Større tidsrom gir selvsagt mer generell oversikt, men dette vil også si at det blir vanskeligere å ta en avgjørelse på bakgrunnen av resultatene. Det beste var altså å gi brukeren full kontroll på dette.

### 4.4.2 Webside GUI og funksjoner

Sluttproduktet er en webside med et kart som viser flytrafikken i et gitt tidsrom. Det skal være mulig for brukeren å sette et start- og sluttidspunkt, og ikoner skal plasseres ut på kartet for å representere et fly på et gitt tidspunkt. Disse ikonene skal kunne klikkes på for å gi mer utfyllende informasjon (alt dette hentes fra databasen).

Overflyvninger blir presentert som en linje med punkter, der hvert punkt er en rad fra databasen. Et flyikon blir brukt for å representere et valgt "øyeblikk" i tid. Systemet er vist i Figur 4.11: Websideelementer.

**Figur 4.11: Websideelementer**



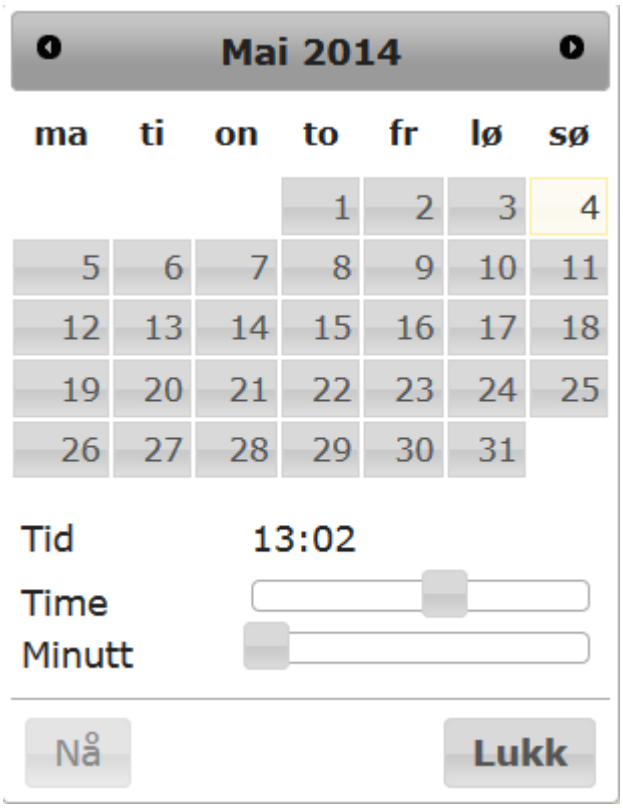

1. Kartet kan navigeres ved hjelp av verktøyene øverst til venstre på kartet, eller ved å klikke og dra på kartet.
2. På kartet er det plassert et rødt rektangel som representerer rekkevidden til målestasjonen. Den rødstriplede linjen er en del av dette rektangelet. Denne begrenser datautvalget, slik at man aldri vil finne data for områder utenfor. Grensen er satt til de største og minste verdiene av lengde- og breddegrader registrert i systemet ved tidspunktet for vårt prosjekt. Grensene kan forandres ved hjelp av en konfigurasjonsfil, hvis det ønskes å begrense utvalget ytterligere.
3. Det ønskede tidsrommet for datauthenting kan bestemmes ved hjelp av dette verktøyet. Her kan dato og klokkeslett bestemmes for start- og slutt tidspunkt for datautvalget. Data hentes automatisk for den siste halvtimen når websiden åpnes for første gang (eller fra forrige utvalg hvis dette ble spesifisert ved et tidligere besøk). Det er også mulig å velge mellom ulike pre-definerte tidsintervaller. Siste halvtime, siste time eller siste 2 timer.

   Når det gjelder de spesifike tidsutvelgerne benytter disse en kalender slik vist i Figur 4.12: Tidsvelger. Pilene øverst brukes for å bestemme måned (og år). Dagen i dag er alltid markert,

og aktuell dag for datauthentingen velges ved å trykke på dagen i kalenderen. Tiden bestemmes ved hjelp av to "slidere". Når ønsket tidspunkt er valgt kan dialogen bare lukkes ved "lukk" knappen, da selve tekstfeltet alltid holdes oppdatert med verdiene valgt i vinduet. Tekstfeltet som datostrengen plasseres i kan også editeres direkte.

Brukeren får tilbakemelding hvis et stort antall data blir uthentet, hvis brukeren ønsker å avbryte operasjonen. Hvis det valgte utvalget er tomt og større enn 24 timer, kan dette tyde på at systemet var nede. Brukeren blir gitt et informasjonsvindu som blant annet inneholder data om tidspunkter der det er kjent at systemet var nede.

Figur 4.12: Tidsvelger




4. Den lange "slideren" nederst til venstre er tidslinjen. Denne bestemmer "øyeblikket" for dataene som vises på kartet. Dette brukes for å bestemme hvor de interpolerte flyikonene plasseres. En forklaring på tidslinjen blir gitt i programmet ved å trykke på spørsmålstegnet.
5. Her kan man se både øyeblikksbildet for et fly, alle punktene fra databasen ("punkt-prøver" for overflyvningen) og linjene som knytter disse samme. Flyikonet representerer som sagt den nøyaktige plasseringen for flyet på øyeblikket valgt på tidslinjen. De andre punktene er verdiene hentet fra databasen, og viser sammen med den tilknyttende linjen hvilken bane flyet følger.

   Hvis man trykket på enten et flyikon eller et punkt på linjen, vil et informasjonsvindu dukke opp. Dette viser de tilhørende dataverdiene for dette punktet. Man kan se informasjon som posisjon, høyde, fart osv. for hver observasjon. Verdiene brukt for flyikonet er interpolert

basert på de nærmeste punktene. Navnet på flyet, som står øverst i informasjonsvinduet, er rutenummeret tildelt av flyselskapet. Dette er IKKE adressen til den fysiske senderen plassert inne i flyet (j.fr. 4.2, hexident). I tillegg inneholder dette vinduet to funksjoner for å gjøre navigering på tidslinjen enklere. Det er to ikoner, der det ene begrenser det valgte tidsrommet til å bare gjelde for perioden for den valgte overflyvningen. Det andre ikonet hopper i tid (på tidslinjen) til tidspunktet for det valgte punktet (flyobservasjonen).

6. Dette sender-liknende ikonet representerer målestasjonen i Hessdalen. Dette står på en fast posisjon på kartet. Dette inneholder også et enkelt informasjonsvindu med data om målestasjonen.
7. Knappene plassert i denne boksen gir brukeren mulighet til å påvirke hva som vises på kartet. Fra toppen og nedover er disse: Heatmap, Stier, Noder og Sticky.

   **Heatmap** bytter ut ikonene på kartet med en "heatmap", altså en generell oversikt over intensiteten av flytrafikk. På denne måten får man en mer generell oversikt over store mengder data. Dette kan for eksempel brukes for å gi en generell oversikt over sannsynligheten for fly, basert på hvor de fleste flyene har passert tidligere. Merk at heatmap-en er basert på de diskrete "punkt-prøve"-verdiene fra databasen, i motsetning til en kontinuerlig linje med punkter. Dette vil føre til områder med grønne enkelt-punkter uten tilknytning. Dette betyr bare at det er for lite data til å produsere et bruktbart heatmap i området, derfor kan disse bare oversees. En forklaring på heatmap-en vises ved at man trykker på spørsmålstegnet i programmet.

   **Stier** bestemmer om stien som flyet fulgte skal vises på kartet. Dette gjøres ved å tegne en strek mellom nodene fra databasen.

   **Noder** er de små rundingene på kartet. Disse representerer radene med data fra databasen, og kan trykkes på for å få detaljert informasjon om flyene. Flyikonene på kartet baserer sine verdier på interpolasjon mellom slike noder.

   **Sticky** gjør at informasjonsvinduer for flere ikoner kan åpnes samtidig. På denne måten blir det mulig å se data for flere fly samtidig, slik at disse lett kan sammenliknes opp mot hverandre.

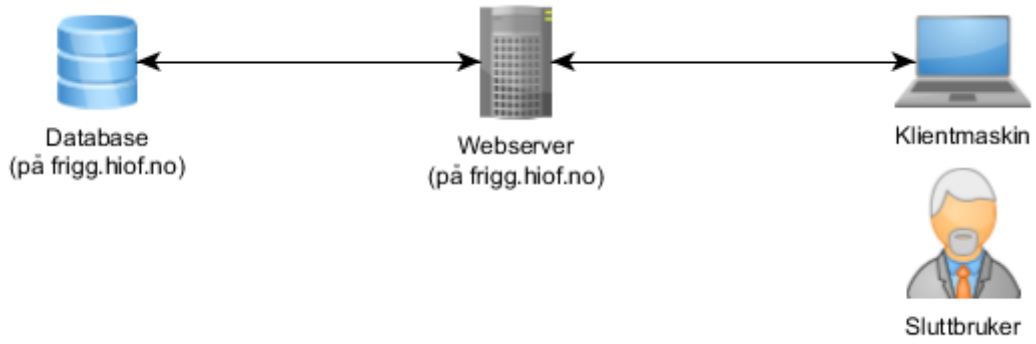
## 4.5 Systemoversikt

Etter å ha forklart de ulike systemene i de foregående kapitlene vil den totale systemoversikten beskrives her, hovedsakelig hvor de ulike filene som utgjør systemet eksekverer, og hvordan de forholder seg til hverandre. For sammenhengen mellom hovedkomponentene i vårt system, og hvordan dette henger sammen med fly- og lysdeteksjonssystemene, se 4.1.1.

### 4.5.1 Hovedtrekk

Komponentene som vårt system omfatter er vist i Figur 4.13: Vårt system.

Figur 4.13: Vårt system




Komponentene som vårt system omfatter er databasen og webserveren på frigg, samt klientmaskinen hos sluttbruker. Data blir overført mellom disse tre systemene, og undergår ulike typer prosessering langs veien. Merk at webserveren i ettertid ble integrert med websiden for Hessdalen, og flyttet til hessdalen.org. Dette hadde ingen innvirkning på ytelse eller liknende, men forekom etter at løsningen og rapporten var ferdig utarbeidet. Derfor tar systemforklaringen utgangspunkt i at webserveren er plassert på frigg.hiof.no.

Alt begynner med at brukeren går inn på vår side og velger et tidsintervall for uthentingen. All flytrafikk innenfor dette tidsintervallet blir hentet ut. Forespørselen skjer via en AJAX forespørsel tilbake til webserveren (uten dette vil ikke brukeren få kontakt tilbake til webserveren etter at siden har blitt lastet).

Forespørselen blir behandlet av webserveren, som kontakter databasen (som foreløpig er plassert på samme fysiske servernettverk). Dette skjer ved hjelp av en SQL forespørsel, som blant annet begrenser hvilke tidspunkter som er interessante under uthentingen. Bare data med "postime"-verdier (tidspunktet da flyet sendte ut meldingen) innenfor grensene satt av brukeren vil bli hentet ut. Webserveren inneholder brukerinformasjon om databasen, slik at den får tilgang til dataene. Disse blir ikke gjort tilgjengelige for sluttbruker, og blir heller ikke påkrevd.

I tillegg til begrensning på tidspunkt, blir også koordinater utenfor det aktuelle området filtrert bort. Dette skjer ved hjelp av statiske parameter, som kan konfigureres. Databasen sorterer deretter dataene, først på hexident og deretter på regtime (samt postime helt til sist) før det sendes tilbake til webserveren.

Webserveren mottar disse dataene, som så blir gruppert. Grupperingen baserer seg på hexident og regtime, og produserer "overflyvning"-objekter, som inneholder datafelter for alle flyobservasjonene overflyvningen omfatter. Svaret på AJAX forespørselen blir deretter sendt tilbake til klienten, altså disse dataene. Dataene er pakket inn i JSON format, som kontinuerlig sendes til klienten under uthentingen fra databasen. Hver gang et overflyvningsobjekt er klart, blir dette sendt til klienten umiddelbart. Webserveren har derfor frie hender for å behandle neste overflyvning.

Klientmaskinen mottar disse dataene og begynner å plotte disse på kartet forløpende, med små pauser innimellom hver overflyvning, slik at systemet klarer å håndtere arbeidsmengden. Når klienten er ferdig med å plotte alle verdiene, er dataoverføringen komplett, og brukeren har nå mottat data for det valgte tidsrommet. Deretter kan brukeren ved hjelp av diverse verktøy begynne å analysere dette.

Det må til sist nevnes at heatmap-en som brukeren kan velge å vise, er basert på det faktiske datautvalget. Det blir ikke utført et separat utvalg for dette.

### 4.5.2 Filstruktur

Vårt system består av flere filer som kjører på to ulike systemer. PHP-filer kjører på webserveren, og de andre blir kjørt på klientmaskinen. Da produktet skulle legges over på webserveren til Hessdalenprosjektet, ble det klart at filene skulle ha både en engelsk og en norsk versjon. Derfor finnes det et duplikat av disse filene med engelsk tekst.

Filstrukturen var fortsatt den samme, derfor er ikke dette inkludert i forklaringen under. En oversikt over filene finner du her:

**Hovedmappe:**

- config.db.inc.php
- config.inc.php
- config.js
- config_eng.js
  Diverse konfigurasjonsfiler. .php-filene angir innstillinger for backend (behandling av data, osv.), mens .js-filene angir forskjellige innstillinger som brukes i grensesnittet og klientapplikasjonen.

- data.php
  Aksesspunktet for databasen, denne henter ut informasjonen og behandler denne for å gruppere fly og fjerne overflødig data.

- index.html
- index_eng.html
  Definerer HTML-struktur og binder applikasjonen sammen.

- /css
  - main.css
    Angir stilinformasjon.

- /data
  - infownd_ams.html
  - infownd_ams_eng.html
  - infownd_plane.html
  - infownd_plane_eng.html
    Angir “mal” for informasjonsvinduene som vises ved klikk på kartikoner.

- /img
  - ams.png
  - circle.png

    - circle_green.png
    - circle_red.png
    - date.png
    - gototime.png
    - menu_heatmap.png
    - menu_node.png
    - menu_paths.png
    - menu_pin.png
    - narrow.png
      Bildefiler som brukes i applikasjonen.

- /js
  Logikk på klientsiden. Javascriptkoden ligger i hovedsak her.

    - data.js
    - data_eng.js
      Logikk for databehandling og –overføring

    - icons.js
      Behandler bildefiler og ikoner

    - infownd.js
      Behandler informasjonsvinduer

    - map.js
      Oppsett og vedlikehold av kartet og relaterte objekter

    - settings.js
      Grensesnitt for innstillinger

    - template.js
    - template_eng.js
      Laster inn og fyller ut maler for informasjonsvinduer. Malene ligger i data/-mappen.

    - template-funcs.js
      Inneholder funksjoner som kan kjøres før informasjon settes inn i informasjonsvinduer. Dette inkluderer for eksempel formatering av tidspunkter og koordinater.

    - ui.js
    - ui_eng.js
      Inneholder logikken bak grensesnittet, event handlers og liknende

    - util.js
      Inneholder diverse funksjoner som har allmen nytteverdi og ikke passer inn noe annet sted.

- /lib
  Javascript-biblioteker fra andre kilder
  - date.js
    Nyttefunksjoner for enklere behandling av datoer og tidspunkter
  - seedrandom.js
    PRNG-implementasjon
  - /lib/timepicker
    GUI-objekter for enkelt utvalg av datoer og tidspunkter
    - timepicker.css
    - timepicker.js

index.html er dokumentet som kjører på klientmaskinen, og binder websiden sammen. Her blir alle elementene på websiden spesifisert, og alle relevante skript blir inkludert. Stilarket som bestemmer utseende for websiden ligger i main.css.

data.php er backend-skriptet som kjører på serveren, med ansvar for å hente ut data fra databasen og overføre dette til klienten. I denne filen kjøres SQL-spørringen mot databasen, som henter ut de relevante dataene, behandler dem, og konverterer dem til JSON. Merk at data.php også utfører en tynning på dataene før de sendes videre til klientmaskinen. Grensen er satt til maksimalt 10 observasjoner per overflyvning, men dette er konfigurerbart (opp til 25 observasjoner).

Interaksjonen mellom index.html og data.php foregår i data.js. Dette skriptet sørger for å sende forespørselen til data.php, og tar imot dataene (JSON) og konverterer dem til objekter som systemet tar i bruk. Denne filen inneholder også funksjonalitet for blant annet interpolering.

map.js inneholder logikken for selve tegningen av Google Maps -kartet. Her opprettes selve kartet og alle visuelle elementer som tegnes på det. ui.js inneholder tilsvarende logikk for resten av websiden, slik som dato-velgeren.

Konfigurasjon av systemet befinner seg i tre ulike filer: config.db.inc.php, config.inc.php og config.js.

config.js inneholder konfigurasjonen for klientmaskinen. Her bestemmes blant annet standard posisjonering av kartet, avgrensning av aktuelt område på kartet og innstillinger for heatmap og dato-velgeren. I tillegg defineres enkelte andre variabler som kan være interessante å “tweake”.

config.inc.php inneholder konfigurasjonen for backend-skriptet. Her settes eventuelle statiske grenser for dataene som hentes fra databasen, slik som absolutt geografisk avgrensning. Merk at dette tilsvarer den geografiske avgrensningen angitt i config.js (som kun er grafisk). Hvis disse verdiene forandres, må dette forandres i begge filer, slik at utvalget stemmer overens med den fysiske grensen tegnet på kartet. I tillegg til dette inneholder konfigurasjonsfilen "MAX_REGTIME_DIFF" og "RESOLUTION_DEFAULT", som angir hvordan fly skal grupperes og hvor detaljert informasjon som sendes til klienten skal være.

config.db.inc.php inneholder databaseinformasjonen, som bestemmer hvilken database som kobles til for å hente dataene samt brukernavn og passord som skal benyttes ved tilkobling.

I tillegg til disse inneholder programmet flere javaskriptfiler som håndterer mindre komponenter i systemet.

infownd.js står for genereringen av informasjonsvinduer, som så knyttes til flyikon-objekter.

template.js håndterer genereringen av selve informasjonsvinduteksten. Dette behandler filer for hver type informasjonsvindu plassert i data-mappen. Disse filene inneholder spesielle kodesnutter på vårt eget enkle makroformat, som gjør det enkelt å prosessere data og liknende, før det ferdige informasjonsvinduet blir presentert for brukeren. Funksjonene som kan brukes i disse malene defineres i template-funcs.js.

settings.js er et interface som gjør det lettere å behandle diverse innstillinger i programmet.

icons.js tilgjengeliggjør de ulike ikonene i programmet, både vektorgrafikk spesifisert med path, og faktiske ikon-filer plassert i img-mappen.

Til sist har vi util.js som inneholder diverse enkeltstående funksjoner, som å regne ut vinkelen mellom basestasjonen og et fly, eller konvertering fra HSL til RGB-farger.

lib-mappen inneholder ulike eksterne biblioteker vi har tatt i bruk i vårt produkt.

I tillegg til disse filene brukte vi en fil, maint.php, for tynningen av databasen. Denne hentet ut data fra databasen og produserte en SQL-fil for data som skulle slettes. Selve logikken som ble brukt er meget liknende tynning-logikken i data.php.

# 5. Testing og evaluering

Produktet gjennomgikk flere runder med testing internt, i flere ulike miljøer, noe som fulgte naturlig av at vi arbeidet iterativt (se 3.5.4). Etter hver store forandring prøvde alle gruppemedlemmer å finne feil og mangler i produktet. Det skal også nevnes at gruppemedlemmer til tider var distansiert fra utviklingen av produktet, grunnet arbeid på denne rapporten. Dette fikk en positiv effekt på testingen, da det ble lettere å få en "naturlig" respons fra hvordan det fungerte.

Det ble også holdt en demo for oppdragsgiver og veileder nær slutten av prosjektet. På denne kom et viktig poeng frem, at det ikke var helt intuitivt hvordan de ulike elementene fungerte. Spesielt heatmap. Dette førte til en direkte implementasjon av et informasjonsvindu for sluttbruker. Dette gir brukeren en generell oversikt over produktet.

Det ferdige produktet var stabilt og fungerte tilfredstillende. Problemer med datamengder og liknende ble løst med diverse grep, som gjorde at opplevelsen for sluttbruker var responsivt og relativt raskt. Et problem som imidlertid sto igjen, og som vi ikke fikk løst, var at produktet ikke fungerte i Internet Eplorer. Dette ble klart relativt sent i prosjektet, og var derfor noe vi ikke fikk tid til å løse. Produktet fungerte imidlertid som forventet i både Google Chrome og Mozilla Firefox. Brukere av Internet Explorer får en beskjed om at produktet ikke vil fungere for dem.

# 6. Diskusjon

I dette kapittelet diskuterer vi resultatene vi oppnådde, med fokus på målet, kravspesifikasjonene til produktet og arbeidsmetoden spesifisert i 1.4.

## 6.1 Oppnådde mål

Vårt produkt har gjort det lettere å avklare om et detektert lysfenomen var et forbipasserende fly eller ikke. Dette blir oppnådd ved å hente ut data fra databasen, og presentere dette på en webside. Noen av dataene var mangelfulle, og det var hele perioder der flydeteksjonssystemet hadde vært ute av drift. Det var dessverre noe vi ikke kunne løse.

Samplingsfrekvensen til systemet har også meget høy da vi begynte på vårt prosjekt, som førte til unødvendig store mengder data i databasen. Vi kunne ha latt være å forandre på dette, men da ville hele vårt system blitt mer uresponsivt. Vi fikk kuttet ned betraktelig på datamengden, og la til funksjonalitet for å tynne det ytterligere før det ble sendt til klientmaskinen. Hvis vi hadde hatt bedre tid, kunne vi prøvd å forbedre databasen, fått ryddet opp i den, men dette valgte vi å ikke gjøre. Fokuset vårt lå på å utvikle vårt produkt, og bare forholde oss til databasen og flydeteksjonssystemet slik dette var satt opp. Dette med unntak av uttynningen, som viste seg å være helt nødvendig.

Vi har gjort det lettvint å hente ut data fra nøyaktige tidsintervaller. Systemet henter automatisk inn data fra de siste 30 minuttene når websiden blir lastet. Knapper for å hente ut data fra predefinerte intervaller er på plass: siste 30 minutter, siste time og siste 2 timer. Nøyaktige tidspunkt, ned til minuttet, kan kontrolleres ved help av en kalender og "slidere".

Det var desverre tidspunkter der systemet hadde vært ute av drift, slik at det ikke fantes data. Slik nevnt over var det ikke noe å gjøre med dette. Derfor valgte vi å heller å informere brukeren om problemet. Hvis brukeren hentet ut data fra et tomt tidsrom på over 24 timer ble brukeren presentert en dialog med informasjon om datoer der flydeteksjonssystemet hadde vært ute av drift.

Data for nøyaktige tidspunkter kan hentes ut ved hjelp av en "slider" nederst på websiden. Denne gir interpolerte dataverdier for tidspunkter utenfor de som databasen dekker. Vi var fornøyd med tidsutvelgeren vår, og selv om vi hadde hatt bedre tid, er det ikke noe vi hadde gjort annerledes i denne delen av oppgaven.

Dataene presentert av vårt produkt er alle de aktuelle datafeltene fra databasen, men det var også data som var av liten interesse for arbeidsgiver. Vi lot være å hente ut og presentere disse dataene. De dataene vi valgte å hente ut blir vist på kartet. Noe blir representert ved plassering og orientering av flyikonene, andre data kan aksesseres ved å trykke på et flyikon, som åpner en tekstboks. Vi kunne ha valgt å legge til avanserte analysemuligheter, grafer og slikt, men bestemte oss for å ikke gjøre dette. Grunnen til dette var at nettsiden hadde blitt mer rotete og mindre brukervennlig. Det hadde også tatt lang tid å implementere. Dette, i forhold til hvor lite interessant dette var for oppdragsgiver, førte til at vi ikke implementerte slik funksjonalitet. Unntaket er en heatmap, som tilfører en slags oversikt over sannsynlighet for fly.

Vi er fornøyde med kartikonene. De ser bra ut med ulike mengder data. Det kan bli litt rotete hvis man velger et for stort tidsrom, men dette er uunngåelig da vi ønsker å gi brukeren så mye frihet som mulig.

Ikonene på kartet inkluderer flyikoner, noder, stier og et ikon for Hessdalen AMS. Flyikonene er produsert ved hjelp av vektorgrafikk, og utarbeidet av oss. Dette representerer det øyeblikket valgt på tidslinjen, og har verdier interpolert ved hjelp av nærliggende noder. Hessdalen AMS representerer målestasjonen som befinner seg i Hessdalen. Den er formet som en svart sender på kartet. Nodene er små hvite sirkler med svart kantlinje. Dette er data om flyobservasjoner fra databasen. Brukeren kan trykke på alle disse ikonene og få opp en tekstboks med informasjon, som forklart i tidligere i dette kapitlet. Stiene tegnes mellom nodene, og viser banen flyet har tatt. Dette gjør det lettere å vite hvilke ikoner som hører sammen, dette forsterkes av at flyikonene og stiene har samme farge for ett fly.

Brukeren kan også velge å feste informasjonsboksene på skjermen, slik at de ikke lukkes ved å trykke andre steder. Dette gjør det lettere å sammenlikne data mellom to ulike ikoner. Sticky-modus aktiveres ved å trykke på den korresponderende knappen på verktøylinjen. Vi har valgt å gjøre det på denne måten fordi det ser designmessig bra ut, og ga en ryddig måte å representere store mengder data. Hadde vi hatt bedre tid, kunne vi ha prøvd å forbedre ulike designaspekter, men vi er fornøyd med resultatet vi kom frem til.

Slik nevnt over inkluderer også programmet muligheter for å gi en oversikt over intensiteten av flytrafikk i en periode. Dette gjøres ved at et farget "filter" legges over kartet, som viser intensiteten på en skala fra blått til rødt. Denne baseres på de diskrete datapunktene ("stikk-prøver") fra databasen, noe som fører til områder med grønne punkter uten tilknytting. Dette betyr at det er for lite data i området til å gi en god oversikt. Grunnen til at dette er at vi tok i bruk et eksternt bibliotek for vår heatmap, og denne aksepterte ikke linjer, bare enkeltpunkter. Problemet kunne vært løst ved å interpolert flere punkter før heatmap-en ble tegnet, men vi valgte å ikke gjøre dette. Det hadde ført til unødvendige mengder ekstraprosessering, og hadde ikke tilført mye. Heatmap-en ser akseptabel ut med større mengder data, det er bare ved minimale mengder at slike resultater oppstår, og disse kan bare bli sett bort fra.

## 6.2 Levert produkt

Hovedproduktet er en webside som presenterer dataene fra databasen, ved hjelp av et interaktivt kart. Man kan også velge ut data i et bestemt tidsrom, ved hjelp av et sett med tidsvelgere. Dataene vises ved hjelp av flyikoner, noder og tekstbokser. Man kan også "feste" tekstboksene/informasjonsvinduene ved help av en "sticky"-knapp, som gjør at de ikke forsvinner når andre informasjonsvinduer åpnes. Dette gjør det lettere å sammenlikne data. For å vise intensiteten av flytrafikk implementerte vi et heatmap. Jo mer flytrafikk det er i et område, jo rødere blir fargen (på en skala fra blått til rødt). Vi valgte å gjøre det på denne måten, både fordi det så designmessig bra ut, og ga brukeren en god oversikt over store mengder data. Både vi og oppdragsgiver var fornøyd med produktet. Hvis vi kunne forbedret noe på sluttproduktet, hadde det vært å legge til flere verktøy. Animasjon er et godt eksempel på dette, altså at tidslinjen kunne "spilles av" med Play/Pause-funksjonalitet. Vi skulle ønske gjerne ha fått forbedret heatmap-en, slik

at enkeltpunktene fra databasen ikke ga "rare", og potensielt uforståelige, heatmaps. Dette kunne vi for eksempel ha gjort ved å basere heatmap-en på linjer istedenfor punkter.

Etter at prosjektet var ferdig, når vi skulle publisere det på nettsiden til Hessdalen-prosjektet (http://www.hessdalen.org/), støtte vi på et nytt problem. Oppdragsgiver ville ha produktet i både engelsk og norsk versjon. Resultatet var en rask "hack" der alle filene med norsk tekst i seg fikk tilsvarende oversatte versjone. Disse ligger parallelt med de originale filene. Dette er naturligvis langt fra optimalt, men det var det eneste vi kunne gjøre på så kort varsel. En idéell løsning på dette problemet ville vært dedikerte "språkfiler" som inneholdt alle tekststrenger som ble vist til brukeren, og der brukerens språkvalg ble brukt som oppslagsnøkkel. På denne måten kunne disse forandres dynamisk uten massiv duplisering av kode.

Vi er selv veldig fornøyd med rapporten, siden vi har diskuterte den grundig, og prøvde flere ulike angrepsvinkler. Hvis vi hadde satt den sammen som en liste over kronologiske hendelser, hadde dette vært meget enkelt å skrive. Dette hadde derimot fått rapporten til å høres for mye ut som en "fortelling", og hadde ikke hatt et like ryddig oppsett. Vi valgte å heller bruke en struktur som startet med en innledning, med oversikt over oppgaven, samt strukturen for resten av rapporten. Vi fulgte dette opp med et analysekapittel, hvor vi tok for oss en grundig analyse av oppdragsgivers ønsker, og så på de ulike teknologiene og mulighetene vi hadde til rådighet for å løse de relaterte problemstillingene. I neste kapittel tok vi for oss de løsningene vi valgte, og forklarte hvorfor og hvordan vi implementerte disse. Deretter hadde vi kapittelet som dekket implementasjonen av de ulike løsningene. Dette kapittelet inneholdt også en grundig oversikt over de ulike delene av systemet, og hvordan disse hang sammen. De tre siste kapittelene, testing og evaluering av systemet, diskusjon om hva vi syntes om prosjektet, og til sist en konklusjon av diskusjonen, reflekterte tilbake over de andre kapitlene. Her fikk vi rundet av rapporten på en god måte, der vi kunne uttrykke vår mening om oppgaven som helhet. Vi valgte å skrive hovedrapporten i Microsoft Word, da vi var mer kjent med dette enn OpenOffice og LaTeX. Word inneholdt også all funksjonalitet vi trengte. Dette var et godt valg, da det var lettere å gjøre raske forandringer frem-og-tilbake mellom ulike gruppemedlemmer. Word har imidlertid noen problemer og bugs, derfor kunne det i retrospekt ha vært greit å bruke LaTeX. Da hadde det vært lettere å garantere at rapporten hadde den ønskede strukturen, og en garantert homogen layout. Dette hadde imidlertid tatt lenger tid, da vi også måtte ha lært oss å bruke LaTeX mens vi utarbeidet rapporten, derfor var Word det beste valget for oss.

## 6.3 Evaluering av arbeidsmetode

Vi brukte mye tid på å diskutere ulike mulige løsninger, og deres styrker og svakheter. Dette tok relativt lang tid, men vi gjorde det slik for å oppnå beste mulige kvalitet på sluttproduktet. Ved å gjøre det slik, var det mindre sannsynlig at vi måtte gå tilbake ved et senere tidspunkt for å gjøre om på systemet hvis vi hadde tatt dårlige valg tidlig.

Implementasjonen av systemet har vært relativt uproblematisk. Noen av løsningene har tatt lenger tid enn forventet, og vi hadde flere problemer grunnet i systemet utarbeidet av den tidligere prosjektgruppen, men utenom dette har arbeidet gått greit. Vi har møtt på noen problemer der vi brukte eksterne biblioteker, og implementering og tilpassning av disse har til tider vært problematisk. Vi valgte å fortsatt bruke disse bibliotekene, da løsningene de ga levde opp til våre og oppdragsgivers ønsker. Det sparte oss også for mye tid, i forhold til hvis vi hadde utarbeidet dem manuelt.

Det skal også nevnes at en av de største komponentene av vårt sluttprodukt er et stort program bestående av mange komponenter som alle skal kommunisere med hverandre. Det er problematisk å delegere ut arbeidsoppgaver når man snakker om programmering av denne typen, derfor hadde vi hovedsakelig *en* utnevnt person som tok for seg den overordnede programmeringsjobben. Løsningene fremkom så i fellesskap, men det er hovedsakelig en person som tok seg av implementasjonen. Vi brukte lengre tid ved å gjøre det på denne måten, men vi fikk en bedre oversikt over programmet, og det ble ikke misforståelser som kunne skape større problemer. Vi kunne ha arbeidet på de samme filene samtidig, men dette ville ha krevet meget god kommunikasjon og planlegging, og potensielt mye tid på å rette opp eventuelle misforståelser. Vi foretrakk vår løsningen, fordi vi kunne splitte opp mindre arbeidsoppgaver som for eksempel utarbeiding av algoritmer for å løse spesifike problemer, som godt kan behandles som en separat arbeidsoppgave uten at dette hindrer fremgangen i prosjektet. Og da selve programmeringen i hovedsak ble håndtert av en person, kunne de andre jobbe videre på rapporten og andre arbeidsoppgaver parallelt med dette. Dette fungerte godt, og førte til et godt produkt med en helhetlig stil og kvalitet.

# 7. Konklusjon

Målet med oppgaven var å utvikle et system som gjorde det enkelt å få oversikt over flytrafikken i Hessdalen i et gitt tidsintervall. Det ferdige systemet gjør dette raskt og enkelt for sluttbruker å få til. Tidspunktene som begrenser intervallet kan velges ned til minuttet, og dataene blir presentert på en oversiktlig og intuitiv måte.

Det var også et godt valg å implementere en tidslinje for interpolasjon. Dataene i databasen var diskrete verdier som bare ga "punkt-prøver" av den fulle overflyvningen. Ved hjelp av interpolasjonen ble det mulig å også få en approksimasjon for flyets tilstand når det befant seg mellom slike punkter.

Diverse verktøy som heatmap og begrensning av tidsrom rundt valgte overflyvninger, gjorde det også betraktelig enklere å behandle de store mengdene data, som var en av de største problemstillingene i oppgaven.

Systemet ble levert til oppdragsgiver, som var fornøyd med resultatet. Rapporten ble også tilstrekkelig omfattende, og beskrev hele prosjektgangen grundig.

Vår metode fokuserte på å presentere alle mulige løsninger på en gitt problemstilling. Dette resulterte i skillet mellom kapittel 2 og kapittel 3 i denne rapporten. Vi følte at dette fungerte godt, da vi både fikk mulighet til å presentere problemstillinger fra et overordnet perspektiv, samt mulighet til å grundig forklare valgene vi tok, og hvorfor.

I tillegg gjorde vi et valg tidlig i prosjektet, om at vi hovedsakelig overlot implementasjonen av produktet til en designert person. Dette følte vi at fungerte godt, da dette ga et omfattende produkt, som fulgte en uniform stil. Da vi også jobbet i gruppe store deler av tiden, var det alltid mulig for alle gruppemedlemmer å bidra i problemløsningsprosessen.

Vårt valg om å abstrahere oss fra systemet til den tidligere prosjektgruppen ga oss også nok tid til å utarbeide et solid prosjekt. Dette betydde også at vi hadde relativt liten forståelse for databasen og flydeteksjonssystemet tidlig i prosjektet, noe som gjorde at litt tid gikk tapt grunnet misforståelser. En av de viktigste problemstillingene var gruppering av fly ved hjelp av regtime-kolonnen, som vi anså som ubrukbar da prosjektet startet. Men vi følte likevel at dette var et godt valg, da vi fikk tilstrekkelig med tid til å utarbeide vårt produkt slik vi ønsket det.

Hvis prosjektet skulle videreføres hadde implementasjon av mer spesifike typer datanalyse vært interessant. Vi valgte å ikke bruke tid på dette, da vi ikke hadde noen forutsetning for å utvikle gode algoritmer. Dataanalyse var heller ikke en del av vår oppgavebeskrivelse. Dette kunne ha vært en oppgave i seg selv, å utføre grundig analyse av flytrafikken for å finne en sammenheng med lysfenomenet. Slik det står må dette gjøres manuelt. Vi har bare lagt til rette for at dataene enkelt kan aksesseres og sammenliknes, og det blir opp til sluttbrukeren å tolke dataene i systemet.

# 8. Bibliography


CoffeeScript. (2014). *CoffeeScript*. Retrieved april 26, 2014, from CoffeeScript: http://coffeescript.org

Flightradar24 AB. (2014). *Flightradar24*. Hentet Mars 5, 2014 fra Flightradar24 Live Air Traffic: http://www.flightradar24.com

GitHub. (2014). *GitHub*. Retrieved 03 12, 2014, from GitHub: https://github.com/

Movable Type Ltd. (2014). *Movable Type Scripts*. Retrieved april 29, 2014, from Calculate distance, bearing and more between Latitude/Longitude points: http://www.movable-type.co.uk/scripts/latlong.html

Oracle. (2014). *Date and Time Functions, NOW()*. Retrieved April 24, 2014, from MySQL :: Documentation: http://dev.mysql.com/doc/refman/5.5/en/date-and-time-functions.html#function_now

TortoiseHg. (2014). *TortoiseHg*. Retrieved 03 12, 2014, from Bitbucket: http://tortoisehg.bitbucket.org/about.html