Prosjekt Hessdalen – Værstasjoner

BO14-GR15

Kristoffer Jensen, Kristian Norum Karlsen, Mikael Johansen Grimstad og Morten Lindstad

Bacheloroppgave, Avdeling for informasjonsteknologi, Høgskolen i Østfold

21.05.2014

HØGSKOLEN I ØSTFOLD

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

Bacheloroppgave

Prosjektkategori: Bacheloroppgave	X	Fritt tilgjengelig
Omfgang i studiepoeng: 20	30/12 2029	Fritt tilgjengelig etter
Fagområde: Informasjonsteknologi	X	Tilgjengelig etter avtale med oppdragsgiver

Tittel: Prosjekt Hessdalen - Værstasjoner	Dato: 20. mai 2014
Forfatterere: Mikael Johansen Grimstad, Kristoffer Jensen, Kristian Norum Karlsen og Morten Lindstad	Veileder: Einar Von Krogh
Avdeling / Program: Avdeling for Informasjonsteknologi (alle programmer)	Prosjektnummer: BO14-G15
Oppdragsgiver: Erling Petter Strand	Kontaktperson hos oppdragsgiver: Erling Petter Strand

Ekstrakt:
Vår bacheloroppgave er å lage to værstasjoner som skal måle temperatur, luftfuktighet, lufttrykk, vindhastighet og vindretning. Disse dataene skal lagres på en server på Høgskolen i Østfold og visualiseres på hjemmesiden til Project Hessdalen. Rapporten inneholder en introduksjon til gruppa og oppgaven vår. Deretter vil vi beskrive hvordan vi har planlagt arbeidet, produksjon av produktet, testet og til slutt en refleksjon på hvordan vi syns det har gått. Hva som kunne vært bedre og hva vi er fornøyde med.

3 emneord:

Programvare. . .
Maskinvare . . .
Værstasjon . . .

ØSTFOLD UNIVERSITY COLLEGE

Faculty of Computer Science
Remmen
1757 Halden
Phone number: 69 21 50 00
www.hiof.no

BACHELOR'S PROJECT

Project category: Bachelor's Project	X	Free
ECTS: 20		Free after: 30.12.2029
Area: Faculty of Computer Science	X	Free after agreement with costumer

Document title: Project Hessdalen – Weather Stations	Date: 22. may 2014
Authors: Kristoffer Jensen, Kristian Norum Karlsen, Mikael Johansen Grimstad og Morten Lindstad	Tutore: Einar Von Krogh
Department/Program Faculty of Computer Science (all)	Project number: BO14-G15

Costomer: Erling Petter Strand	Person of contact: Erling Petter Strand

Abstract:

This bachelor's projects goal is creating two weather stations that measure temperature, humidity, air pressure, wind speed and wind direction. The data will be stored on a server at Østfold University College and visualized on the homepage of Project Hessdalen. The report provides an introduction to the group and our task. Further on we describe how we have planned our work, the production process, how we tested our product and ultimately a reflection of how we think things have gone. What could be better and what we are satisfied with.

3 keyword:			
	Hardware	Software	Weather station

Forord

I denne prosjektrapporten skal vi ta oss for hele oppbyggingen av en værstasjon som skal være plassert i Hessdalen. Hensikten er å måle værdata som temperatur, lufttrykk, luftfuktighet, vindhastighet og vindretning. I tillegg skal all værdata lagres på en server hos Høgskolen i Østfold og visualiseres på Project Hessdalens hjemmesider.

Prosjektet ble gitt av høgskolelektor Erling Petter Strand og han har gjennom prosjektperioden fungert som oppdragsgiver. Dette prosjektet er laget for Prosject Hessdalen, som også ledes av Strand.

Værstasjonen er i skrivende stund i drift, men det kan være forbedringspotensiale i framtiden.

Sammendrag

Dette er en bacheloroppgave laget av Erling Petter Strand for Prosjekt Hessdalen. Oppgaven går ut på å lage to værstasjoner som skal installeres i Hessdalen. Disse værstasjonene skal måle temperatur, luftfuktighet, lufttrykk, vindhastighet og vindretning. Værstasjonene skal bestå av en mikrokontroller, som skal sende målingene til en server på Høgskolen i Østfold. Målingene skal da vises på nettsiden til Prosjekt Hessdalen.

Denne prosjektrapporten tar for seg hvordan vi har satt sammen værstasjonen. Vi går gjennom hvordan vi har tilpasset sensorene til mikrokontrolleren. Her med tanke på hva som skal til for å kommunisere med disse og hva av maskinvare som må til. Hvordan målingene blir visualisert kommer også frem.

Værstasjonene ble ferdigstilt og installert i Hessdalen 5.-7.mai 2014. I skrivende stund har de vært i drift i to uker. Både oppdragsgiver og gruppen er veldig fornøyd med resultatet.

Takk Til

Vi vil gi en takk til:

Oppdragsgiver Erling Petter Strand med aktiv oppfølgning og innspill underveis i prosjektet.

Overingeniør Geir K. Strøm som har hjulpet oss med maskinvarerelaterte problemer.

Førstekonsulent Hans Olav Bøe som har håndtert utstyrsbestillingen.

Innhold

Figurer

Tabeller

Ordliste

AJAX er en teknologi for dynamisk lasting av innhold. 24, 25, 51, 52

API (Application programming interface) inneholder metoder som kan benyttes av en applikasjon eller et bibliotek. 26

back-end henter inn innhold og/eller utfører oppgaver for front-end. 25, 48–52, 54, 56

C er et høynivå programmeringsspråk utviklet av Dennis Ritchie. 26

ethernet er en familie med nettverksteknologier for tilkobling innen lokale datanett. 15

front-end sørger for kommunikasjon mellom bruker og back-end. 48

HTTP-variabler eller **superglobale variabler** gjør det mulig å sende variabler over HTTP-protokollen. 50

Java er et objektorientert programmeringsspråk. 26, 46, 47

JavaScript er et dynamisk programmeringsspråk/scriptspråk mest brukt i webprogrammering. 24, 25, 48, 49, 51–55

JDBC er kort for ≪Java Database Connectivity≫. Driver som muligjør databasekommunikasjon i Java. 47

JSON er en enkel tekstbasert standard for datautveksling. 25, 26, 46, 47, 50, 51

kompilator er et program som gjør om kildekode til kjørbar maskinkode. 26

mikrokontroller er en liten datamaskin på et enkelt brett. Inneholder vanligvis en prosessor, minne og programmerbare inn-/utganger. 2, 3, 15–19, 21, 23, 26–29, 41, 45, 48

MySQL er et SQL-basert databaseadministrasjonssystem. 23, 47, 56

nettverk-socket er et endepunkt under en toveis kommunikasjon, der prosesser kommuniserer over et IP-basert nettverk.. 26, 47

normaliseringsregler er regler for hvordan et standardisert databaseoppsett skal se ut. 24, 45

PHP er et serverside programmeringsspråk utviklet for webprogrammering. 24, 25, 48, 50, 56

stilark gjør det mulig å endre utseende på en nettside. 55

Kapittel 1

Introduksjon

1.1 Prosjektgruppen

Mikael Johansen Grimstad(født 23.08.1991) gikk tidligere studiespesialisering med realfag ved Frederik II VGS(2007-2010). I året som fulgte utførte han verneplikt som gardist ved Hans Majestet Kongens Garde. Videre valgte å studere Informatikk - Design og utvikling av IT-systemer ved Høgskolen i Østfold fordi han alltid har hatt en stor interesse for elektronikk, data og programmering.

Kristian Norum Karlsen(født 27.09.1991) gikk tidligere studiespesialisering med realfag ved Frederik II VGS(2007-2010). I året som fulgte utførte han verneplikt som sambandsmann i Sambandsbataljonen. Videre valgte han bachelorstudiet dataingeniør ved Høgskolen i Østfold som han fikk et innblikk i under et elevbesøk i 2. klasse på videregående. Dataingeniørstudiet falt som et naturlig valg da dette tar for seg mange av hans interesser.

Kristoffer Jensen(født 14.11.1991) gikk tidligere studiespesialiserende elektrofag ved Glemmen VGS(2007-2010). I året som fulgte utførte han verneplikt som luftvernartillerist ved Ørland hovedflystasjon. Videre valgte han bachelorstudiet dataingeniør ved Høgskolen i Østfold da dette virket som et naturlig valg med tanke på hans interesser.

Morten Lindstad(født 10.05.1992) gikk tidligere på elektrofag med studiekompetanse(2008-2011) ved Glemmen VGS før han startet på dataningeniør på Høgskolen i Østfold. Dette viste seg å ha flere fordeler med tanke på at flere av fagene går igjen med hva han hadde på videregående. Han har alltid hatt en stor interesse for elektronikk og data, alt fra elektriske gitarer til dataspill. Dette er også en fordel fordi han og vil mest sannsynlig ha en enklere forståelse for bacheloroppgaven.

Det var ikke noe tilfeldighet at det var akkurat vi fire som havnet på samme gruppe. Vi har jobbet som en gruppe i flere annledninger tidligere. Vi har samarbeidet i fag som Datakommunikasjon, Bildebehandling og mønstergjenkjenning og Integrerte IT-systemer.

1.2 Oppdragsgiver

Oppdragsgiver er Erling Petter Strand, høgskolelektor ved avdeling for informasjonsteknologi på Høgskolen i Østfold. Han er sivilingeniør fra Norges teknisk-vitenskapelige universitet i Trondheim innenfor elektro med studieretning fysikalsk elektronikk og teleteknikk. Han har jobbet for

Standard Telefon og Kabelfabrikk, EDAS målesystemer og EDB- og Automatiseringsavdelingen på Østfold Ingeniørhøgskole i Sarpsborg som ble sammenslått med informatikk i Halden som nå er avdeling for informasjonsteknologi på Høgskolen i Østfold. Han er en av grunnleggerene av Project Hessdalen og er nå leder av dette prosjektet.

1.3 Oppdraget

Oppdragsgiver ønsker to værstasjoner som skal kunne måle temperatur, lufttrykk, luftfuktighet, vindhastighet og vindretning. Disse to stasjonene skal kobles opp på to forskjellige steder og værdataene skal sendes og lagres på serveren til Høgskolen i Østfold for deretter å bli presentert på hjemmesiden til Project Hessdalen.

1.4 Formål, leveranser og metode

1.4.1 Formål

Hovedmål : Lage en komplett værstasjon med datapresentasjon på hjemmesiden til Project Hessdalen.

- **Delmål 1:** Lage en værstasjon som skal måle temperatur, lufttrykk, luftfuktighet, vindretning og vindhastighet.
- **Delmål 2:** Lagre værdata på server til Høgskolen i Østfold.
- **Delmål 3:** Presentere værdata på hjemmesiden til Project Hessdalen.
- **Delmål 4:** Tilrettelegge for å kunne sammenligne værdata fra ulike tidsperioder.
- **Delmål 5:** Lage et kabinett til mikrokontrolleren, koblingsbrettet og strømforsyningene.
- **Delmål 6:** Tilpassning av hardware. Med tanke på lavpassfilter, spenningsdelere, festeanordninger for sensorer og kabinettet.

1.4.2 Leveranser

Resultatet av prosjektet vil være to værstasjoner med datapresentasjon og en rapport. Dataene som værstasjonene produserer vil være tilgjengelig på hjemmesiden til Project Hessdalen. Rapporten vil være en dokumentasjon av prosjektet.

1.4.3 Metode

Slik vi jobber går dette under den induktive metode, det vil si at vi lærer av å gjøre. Det vi vet er at mikrokontrolleren, Ethernut 2.1, skal brukes til å lese sensorene. Det vi må finne ut er hvilke sensorer som kan brukes på mikrokontrolleren. Dette gjøres ved å lese databladene til de sensorene vi tror er aktuelle. Lage en liste over de ulike kandidatene og sende denne til arbeidsgiver som velger ut de som passer best. Når værstasjonen er ferdig skal den testes på skolen før vi monterer den i Hessdalen. God arbeidsfordeling er viktig for oss siden vi er en gruppe på fire. Effektiv jobbing blir viktig for å klare å utføre alle arbeidsoppgavene.

Denne prosjektrapporten skal bli skrevet ved hjelp av LaTeX. Det er typesettingssystem for dokumentproduksjon. All kildekode og tekst kommer vi så til å legge på GitHub, som er en web-basert tjeneste for versjonskontroll og lagring av programvareprosjekter. Da vil alle i gruppen ha tilgang til nyeste versjon, og alle vil ha mulighet til å se og endre alt i prosjektet.

Da vi skal dele inn gruppen slik at alle får forskjellige ansvarsområder, har vi valgt å bruke en flat struktur. Sekretærrollen er den eneste rollen vi skal benytte, den skal gå på omgang blandt alle gruppemedlemmene. Sekretæren har ansvar for å skrive ned møtereferat og ukereferat.

1.5 Rapportstruktur

I kapittel 2 starter vi med bakgrunnsinformasjon på fenomenene til de brennende ildkulene og Project Hessdalen. Deretter mer informasjon om værstasjonen med spesifikke krav. Alt rundt værstasjonen må planlegges og dette tar vi for oss i kapittel 3. Her forklarer vi også hvordan hele systemet fungerer. Fra værdataene kommer inn til mikrokontrolleren og til de er tilgjengelig på nettsiden til Project Hessdalen. Hvordan selve produksjonen foregikk forklarer vi i kapittel 4. Deretter skal systemet testes og godkjennes før det skal monteres i målingstasjonen til Project Hessdalen. Hvordan dette ble gjort blir forklart i kapittel 5. Tilslutt har vi en diskusjonsfase, kapittel 6, hvor vi tar opp hva vi har lært, problemer vi har støtt på, hva vi er fornøyd med, ble arbeidsgiver fornøyd og hva ville vi har gjort annerledes dersom vi skulle ha gjort prosjektet på nytt. I kapittel 7 tar vi opp de viktigste punktene fra diskusjonskapittelet og hva som burde gjøres dersom dette produktet skal videreutvikles.

Kapittel 2

Analyse

2.1 Værstasjon i Hessdalen

2.1.1 Bakgrunn

Figur 2.1: Bilde tatt av Arne P. Thomassen 25. Oktober 1982, mellom klokken 19.00 og 20.30, fra Finnsåhøgda syd, mot øst.

I 1981 ble det observert flere lys flytende(se figur 2.1) over himmelen i bygden Hessdalen i Holtålen kommune, Sør-Trøndelag. De ble beskrevet som brennende ildkuler, og fenomenet ble vidt omtalt i medier. I 1981-1984 var det på det meste mellom 15-20 observasjoner i uka, og folk begynte å omtale det som UFO-observasjoner. Dette førte til at Hessdalen ble et attraktivt turistmål for de som ønsket å se fenomenet selv. Jevnligheten på observasjonene har sunket betraktelig etter dette, og er nå nede i 10-20 i året. [1]

2.1.2 Project Hessdalen

Project Hessdalen er et prosjekt startet i 1983 av organisasjonen ”UFO-Norge” for å finne ut mer om de stadig uforklarte lysfenomenene som blir observert i Hessdalen. I dag blir prosjektet ledet av Erling P. Strand og Høgskolen i Østfold. Det ble satt opp en automatisk målestasjon i 1998 for å filme fenomenene når de oppstår.

2.1.3 Værstasjon

I forbindelse med å utvide den automatiske målestasjonen, går prosjektet vårt ut på å bygge og installere to værstasjoner som skal innhente værdata regelmessig og lagre dette i en database på Høgskolen i Østfold. De to enhetene skal monteres på to bestemte punkter 171 meter fra hverandre. Deretter skal innhentet værdata presenteres visuelt på www.hessdalen.org. Etter ønske fra oppdragsgiver skal følgene data logges på timesbasis:

Temperatur Maksimum, minimum, gjennomsnitt, nåværende, tidspunkt for maksimum og tidspunkt for minimum.

Luftfuktighet Maksimum, minimum, gjennomsnitt, nåværende, tidspunkt for maksimum og tidspunkt for minimum.

Lufttrykk Maksimum, minimum, gjennomsnitt, nåværende, tidspunkt for maksimum og tidspunkt for minimum.

Vind Maksimum, minimum, gjennomsnitt, nåværende, vindretning for maksimum, tidspunkt for maksimum og tidspunkt for minimum.

2.1.4 Krav

Oppdragsgivers krav er at vi skal ha minst én operativ værstasjon i Hessdalen der minst én til to typer for værdata blir logget. Værstasjonene må bygges med mikrokontrolleren Ethernut 2.1. Data skal være visuelt presentert på www.hessdalen.org, og gi mulighet for brukerene å velge et gitt tidsintervall de vil se data fra. Data må kunne presenteres på både kurveform og tabellform.

2.1.5 Relatert arbeid

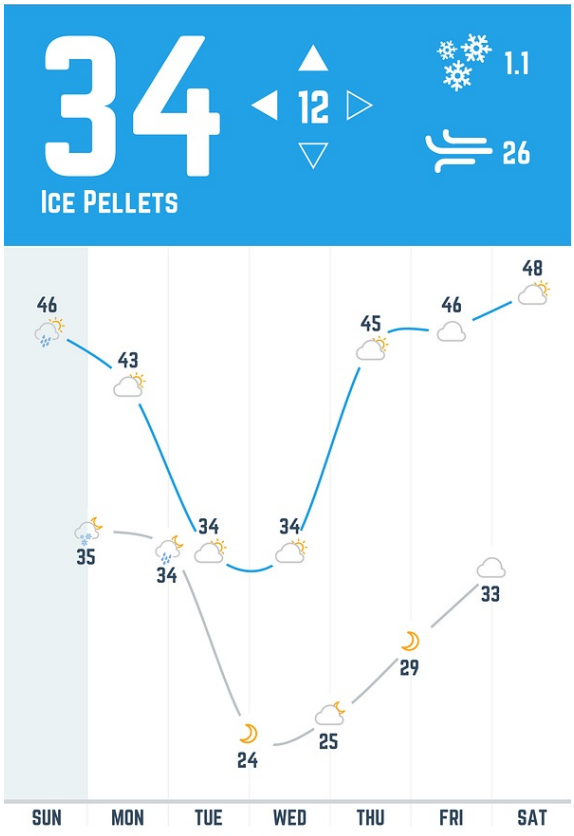
Det har vært mange tidligere prosjekter i forbindelse med Hessdalen, men ingen prosjekter der værdata skal representeres. I dag kan nesten hvem som helst kjøpe en egen værstasjon som et eget system. Det finnes systemer som er mer brukervennlig enn andre, hvis man ikke er erfaren på det området.

Dr. Bunson[2]

Det er mange eksempler på andre som har laget sin egen værstasjson. En av de er Dr. Seth Brown(på bloggen sin bruker han navnet Dr. Bunson). Han har laget en liknende værstasjon som det vi skal. Bakgrunnen for at han bygde sin værstasjon er at han bor på et område der det er dårlige representative værdata, så han valgte å ta saken i egne hender.

Han vurderte først å lage sin egen værstasjon fra bunnen, basert på enten på mikrokontrolleren Arduino eller Tessel, men fant ut at ingen av disse ville tåle påkjenningene av ekstremværet der han bor (Mount Washington). Derfor bestemte han seg for å kjøpe Davis 6250 Vantage Vue[3], en ferdigbygd kombinert vind-, regn- og temperaturmåler. Denne værstasjonen sender data trådløst, og serveren som tar i mot dette er en Raspberry Pi model B[4] med en Edimax EW-7811Un trådløs nettverksadapter[5].

Serveren programmerte han i Node.js, et JavaScript-bibliotek for nettverksservere. Han brukte også JavaScript for å få visualisert data på skjermer i huset. Han har også programmert et varslingssystem på serversiden ved bruk av Nodemailer som sender SMS-varslinger til telefonen hans når været når ekstreme verdier. Figur 2.2 viser presentasjonen av værdata som er laget i JavaScript.



Figur 2.2: Skjermdump av Dr. Bunsons JavaScript-presentasjon av værdata.

Arduino weather station[6]

En annen som har laget sin egen værstasjon, skrev en artikkel på hvordan han satt opp hele systemet, steg for steg. Han oppgir ikke navn på nettsiden, men han går under brukernavnet bram2202. Han bruker sensorer for trykk, temperatur, luftfuktighet og bevegelse. Han viser og skriver detaljert hva han bruker av utstyr og hvordan han koblet alt mot en Arduino mikrokontroller. Han har tatt med en sender og en mottaker mellom mikrokontrolleren og sensorene.

Værstasjonen ble bygd på en slik måte at han alltid kunne ha plass til utvidelser. Han viser ved et eksempel hvor han kobler opp og programmerer en sensor som måler nedbør. Alt han trengte var en måler for nedbør og en boks for å sikre denne måleren mot vann. Måleren kan bare kobles på en ledig port på mikrokontrolleren. Figur 2.3 viser værstasjonen etter den var ferdig oppkoblet.

Figur 2.3: Den ferdige værstasjonen til bram2202 som viser temperatur, luftfuktighet og trykk

2.1.6 Kan vi lære noe av disse prosjektene?

Dr. Bunson mente at mikrokontrollerkort ikke tåler nok til å brukes som værstasjon der han bodde, men det var hovedsaklig på grunn av ekstrem vind. I Hessdalen er det meste sannsynligvis kulde som kan bli det største problemet, da temperaturen kan krype under 30 minusgrader[7]. Siden det er påkrevd i oppgaven at vi bruker Ethernut 2.1, må vi beskytte kortet godt. Med denne informasjonen i bakhodet, vil en prioritet for oss være å bygge vår værstasjon inn i en godt isolerende boks med termostat og varmelement. Grensesnittet Dr. Bunson har brukt for fremvisning av data kan også være en inspirasjon når vi skal utforme vår side for datapresentasjon.

Bram2202 forklarer ikke koden han har skrevet for prosjektet, men han har vedlagt det som en link som man kan laste ned med hans instillinger for værstasjonen hvis man ville lagd en helt liknende værstasjon. Det er kun kretsskjemaer og oppkoblingen av målerne han har forklart. Det vi kan bruke av dette prosjektet er at vi kan lage en liknende boks, der vi beskytter alle kretsene for vann.

2.2 Værstasjonens historie

2.2.1 Fra observasjoner til datamodeller

Vi har alltid vært avhengig av været, og behovet for å forutsi været har eksistert i lang tid. Været har alltid spilt en stor rolle for blant annet landbruk, fisking og skipsfart. Derfor har vi mennesker i årtusener hatt et ønske om å kunne forutsi hvordan været kommer til å bli.

I begynnelsen bygde forutsigelsene bare på observasjoner og tilfeldige målinger, men på 1700-tallet ble det utviklet instrumenter som la grunnlaget for at den moderne meteorologien kunne bli en reell vitenskap. Dermed kunne forskere forstå de faktorene som bestemmer værets utvikling, og lage nyttige værmeldinger som for eksempel stormvarsler.[8] Figur 2.4 viser en typisk storm der det kan være gunstig å måle diverse ting som nedbør, lufttrykk og temperatur.

Figur 2.4: Uvær der det kan være gunstig å måle forskjellige parametere som for eksempel nedbør og lufttrykk.

2.2.2 Måling av lufttrykk

Ingen visste at luft har tyngde før italieneren Evangelista Torricelli oppfant kvikksølvbarometeret i 1643. Han la merke til at en kvikksølvsøyle sto høyere da det for eksempel var sol og pent vær enn da det var dårlig vær. I 1714 fant Gabriel Fahrenheit ut at det gikk an å måle temperaturen i luft og vann med et kvikksølvtermometer. Det teoretiske grunnlaget ble også lagt på 1700-tallet, da sveitseren Daniel Bernoulli ga ut verket ≪Hydrodynamica≫. [9]

2.2.3 Første ballongferd

I 1783 gjennomførte Jacques Alexandre Charles den første flygningen med en luftballong som var fylt med hydrogen. Det viste seg at oppfinnelsen var svært velegnet til å samle inn meteorologiske data fra atmosfæren. Fordi det kunne være livsfarlig å sende opp luftballonger i atmosfæren med tanke på lyn og dårlig vær, sendte de første ballongskipperne opp små testballonger før de lettet selv.[10] Figur 2.5 viser Jaques Alexandre Charles sin ballongferd.

Figur 2.5: Figuren viser den andre ballongferden gjort av Jacques Alexandre Charles 1. desember 1793

2.2.4 Utveksling av værdata

Den irske admiralen Francis Beaufort foreslo i 1806 å dele vindstyrken i en skala på 12 trinn – som han senere kalte beaufortskalaen. Dette ga meteorologene en felles referanse. Da telegrafen ble oppfunnet senere på 1800-tallet, ble det for første gang praktisk mulig å innhente værdata fra større områder, og eksempelvis sende ut stormvarsler. I 1849 opprettet Smithsonian Institution et meteorologisk nettverk på tvers av USA, og lignende samarbeidsprosjekter mellom meteorologer ble etablert i Europa på resten av 1800-tallet.[11]

2.2.5 Beregning av værdata

I 1922 fremla den britiske meteorlogen Lewis Fry Richardson verket ≪Weather Prediction by Numerical Process≫ som var en rekke formler som gjorde det mulig å beregne været fremover i tid. Problemet var at det tok omtrent 24 timer å foreta alle de kompliserte utregningene. Dermed var morgendagen over før værmeldingen kunne være klar. Richardson hadde store tanker og ville samle 64 000 meteorologer på ett sted, slik at de kunne løse hver sin del av utregningene, men ideen ble aldri noe av. [12]

2.2.6 Radiosonden et stort framskritt

I 1927 sendte to forskere opp luftballonger med måleinstrumenter, utstyrt med en radiosender. Dette resulterte i at meteorologene fikk tilgang til bedre data. Målingene ble da sendt trådløst hjem, og verdens første radiosonde var en realitet. Ti år senere opprettet USAs meteorologiske institutt en fast værtjeneste basert på radiosonder i ballonger. Under andre verdenskrig ble radioteknikken utviklet videre, og midt i 1950-årene gjorde bedre materialer det mulig å sende opp større luftballonger enn noen gang. Radiosondene kunne nå komme opp i 50 kilometers høyde og samle inn svært nøyaktige data.[13] Figur 2.6 viser en typisk radiosonde.

Figur 2.6: Et eksempel på en radiosonde

2.2.7 Første værsatellitt

Da de første datamaskinene ble bygd på 1950-tallet, kunne meteorologene beregne værets utvikling et helt døgn frem. Men ettersom de tidligste datamaskinene bare hadde en brøkdel av kapasiteten til moderne datamaskiner, tok beregningene fremdeles nesten 24 timer. Fordi meteorologiske beregninger er så kompliserte, egner de seg for databehandling. Meteorologien var derfor med på å videreutvikle datateknologien.[14]

Fra 1960-årene fikk meteorologene tilgang på enda mer data, fordi de startet å sende opp flere værsatelitter. Den første, Vanguard 2, ble sendt opp av den amerikanske marinen i 1959. Den skulle måle skydekkets tykkelse, men den kom ut av kurs og ble ubrukelig.[15] Det gikk bedre med den neste værsatellitten, TIROS-1, sendt opp av NASA i 1960[16]. Nå kunne all datainnsamling foregå automatisk. Figur 2.7 viser TIROS-1, den første vellykkende værsatellitten.

Figur 2.7: Den første vellykkede værsatellitten

2.3 Måling av vær

2.3.1 Lufttrykk

De meteorologiske målingene tar sikte på å finne den romlige fordelingen av det hydrostatiske trykk[17], som igjen bestemmer den storstilte luftsirkulasjonen. Lufttrykk angis i enheten pascal (forkortet Pa), men innen meteorologien brukes heller enheten hektopascal (hPa) som gir en tallverdi lik den tidligere enheten som brukes, millibar (mb). Lufttrykket blir angitt i atmosfærer eller mm kvikksølv (Hg). Én atmosfære (atm) er omtrent det lufttrykket som er ved havnivå. En atmosfære blir da 1013,25 hPa som tilsvarer 760 mm Hg (1000 hPa = 750 mm Hg). Apparater eller instrumenter for måling av lufttrykk kan være for eksempel et barometer. Figur 2.8 viser et enkelt barometer.

Figur 2.8: Et eksempel på et moderne barometer som måler lufttrykk

2.3.2 Vind

Vinden vil alltid variere. Derfor er internasjonal standard å måle middelverdien for vinden i 10 minutter, som også ligger til grunn ved bruk av Beauforts vindskala.

Vindens fart øker med høyden. Det kan være omtrent vindstille langs bakken, men ved noen få meters høyde kan det være kraftig vind. Vinden vil alltid bli større jo lengre opp man kommer, men etter et visst punkt vil hastigheten avta noe. Derfor er det er internasjonalt bestemt at vindmålinger for værvarslings- og klimaformål skal gjøres 10 meter over bakken. [18]

Vindfarten

Vindfarten oppgis i m/s, knop eller km/h. Begrepet vindhastighet beskriver både fart og retning, men brukes ofte bare om fart. Vindfarten bygger på Beauforts vindskala, som opprinnelig ble tatt i bruk på 1800-tallet av den britiske admiral Francis Beaufort. Skalaen ble i utgangspunket laget for seilfartøyer, og ble senere tilpasset instrumentmålinger av vindhastigheten. Vind angitt i skalaintervaller benevnes gjerne vindstyrke. I tabell 2.1 kan vi se en oppsummering av Beauforts vindskala. [18]

Beauforts skala	Navn på vindstyrken	Vindstyrke i knop	Vindstyrke i m/s
0	Stille	Mindre enn 1	0,0 - 0,2
1	Flau vind	1 - 3	0,3 - 1,5
2	Svak vind	4 - 6	1,6 - 3,3
3	Lett bris	7 - 10	3,4 - 5,4
4	Laber bris	11 - 16	5,5 - 7,9
5	Frisk bris	17 - 21	8,0 - 10,7
6	Liten kuling	22 - 27	10,8 - 13,8
7	Stiv kuling	28 - 33	13,9 - 17,1
8	Sterk kuling	34 - 40	17,2 - 20,7
9	Liten storm	41 - 47	20,8 - 24,4
10	Full storm	48 - 55	24,5 - 28,4
11	Sterk storm	56 - 63	28,5 - 32,6
12	Orkan	Over 63	Over 32,6

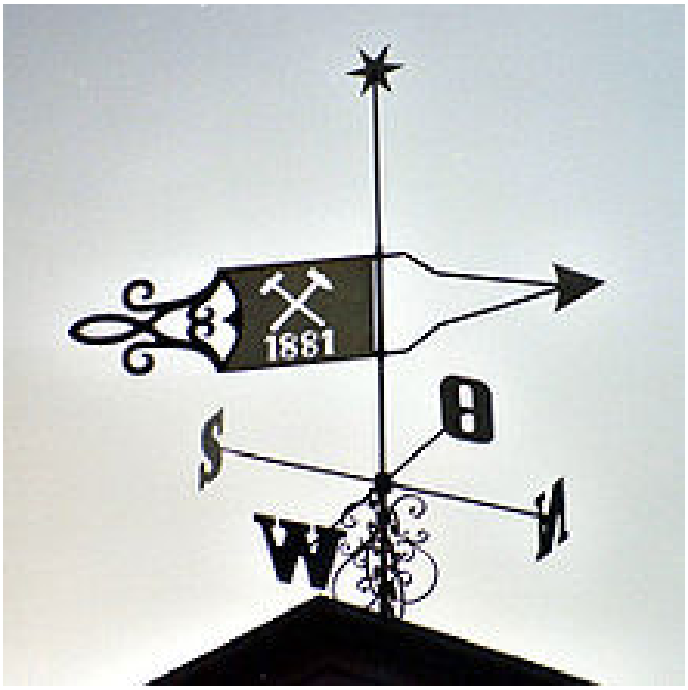
Tabell 2.1: Beauforts vindskala

2.3.3 Vindretning

De to vanligste måtene å måle vindretningen på er med en vindpølse eller en vindfløy.

Vindfløy

Vindfløy er et instrument som viser vindretningen. Vanligvis er en vindfløy utført som en usymmetrisk figur som kan dreie om en vertikal akse, som da kan vise vindens retning. Vindfløyens areal er laget slik at den er større på den ene siden, den siden som er tyngst. Dermed vil denne siden svinge fra vinden. Svært ofte er det festet til et kors eller en sirkel med angivelse av himmelretninger på fløystangen.[19] Figur 2.10 viser til en kjent vindfløy, som er plassert i Zeitz, Tyskland.



Figur 2.9: Vindfløy i smijern på industribygning i Zeitz, Tyskland

Vindpølse

En vindpølse er gjerne en stor konisk-formet sekk som er laget av tekstil. Vindpølsa er åpen i begge ender og den viser vindretning og vindhastighet. Vindpølser er mest brukt på flyplasser, helikopterplasser og på bruer.[20] Figur 2.10 viser en vindpølse på en liten, lokal flyplass.

Figur 2.10: En typisk vindpølse plassert på en liten flyplass

Vindpølsers generelle krav:[20]

- Vindpølsa skal gi presis indikasjon på vindretning og grov indikasjon på vindstyrke.
- Vindpølsa skal være traktformet og åpen i begge ender.
- Vindpølsa skal være oransje eller oransje og hvit. Dersom oransje og hvit velges, skal fargene være i form av 5 sirkulære bånd der første og siste bånd er oransje.

I tillegg for flyplasser:

- Lengden på vindpølsa skal være minst 3,6 meter.
- Vindpølsas største åpningen skal ikke være mindre enn 0,9 m, og den minste åpningen ikke mindre enn 0,3 m.

I tillegg for helikopterplasser:

- Lengden på vindpølsa skal være minst 2,4 meter,
- Vindpølsas største åpningen skal ikke være mindre enn 0,6 m, og den minste åpningen ikke mindre enn 0,3 m,

Kapittel 3

Planlegging

3.1 Oppstartsfasen

Bachelorprosjektet ble innledet ved at gruppen hadde et møte med veileder og oppdragsgiver. Da fikk vi vite litt mer detaljert hva vi skulle gjøre på selve oppdraget og litt om hva veilederens rolle er gjennom prosjektperioden.

Etter jul startet vi opp med ukentlige møter med oppdragsgiver og møte med veileder hver 14. dag. Vi hadde også internmøter i gruppa, en til to ganger i uka, etter behov. Etter vi hadde skrevet forprosjektrapporten, delte vi oss så vi fikk forskjellige arbeidsoppgaver. Det vil bli mer effektivt arbeid hvis vi jobber med hver vår del, enn at alle sitter over en PC og prøver å samarbeide.

3.2 Utforming av produktet

Vi fordelte arbeidsoppgavene slik at noen holder på med programmeringen mens noen andre kobler opp maskinvare. Vi benyttet oss av skolens verksted for å koble opp og montere værstasjonen.

Vi lagde først et par prototyper før vi bestemte oss for det endelige produktet. Vi ville gjerne få hele systemet til å gå før vi tenkte på hva slags design og fysisk beskyttelse vi skulle ha til de forskjellige komponentene.

3.3 Maskinvare

I dette delkapittelet vil vi forklare hvordan vi har tenkt under planleggningen av maskinvare.

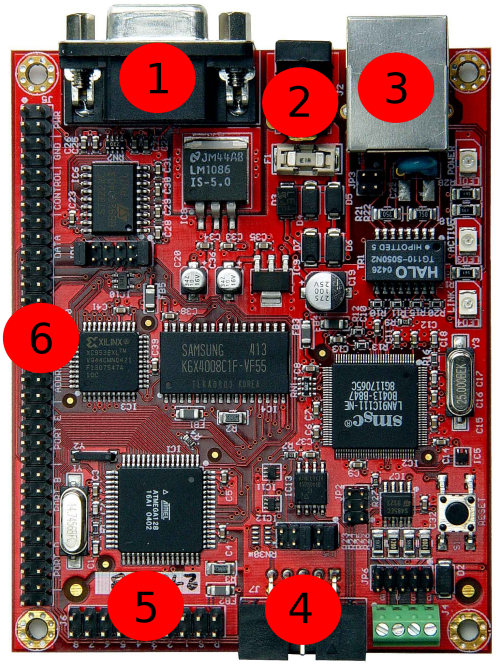
3.3.1 Ethernut 2.1

En av hovedkomponentene i værstasjonen skal være Ethernut 2.1. Det er et lite brett som består av en 8-bit AVR Atmel ATmega128 RISC mikrokontroller og SMCS' LAN91C111 kontroller. Den er en del av en serie med Open Source Hardware referansebrett. Noen av hovedfunksjonene er:

- Full duplex 10/100 Mbps ethernet kontroller.
- 128 kByte programerbar flash ROM(Read-Only Memory).
- 512 kByte Static RAM(Random-Access Memory).
- Inntil 28 programerbare digitale inn- og utganger.

- 10-bit ADC(Analog to Digital Converter) med 8 kanaler.
- To 8-bit og to 16-bit tidtakere.
- Watchdog tidtaker.
- Tåler temperaturer fra -40 ^{o}C til 85 ^{o}C

Den viktigste delen av Ethernut 2.1 er mikrokontrolleren ATmega128, denne er beskrevet i datablad levert av Atmel [21]. Programmering av mikrokontrolleren skjer ved å bruke JTAG-grensesnittet. Dette er en ekstern port som vi kobler til en datamaskin med en adapter og en RS-232 seriekabel. Ethernut 2.1 har også en RS-232 serieport som blant annet kan brukes til å se utskrift fra applikasjonen som kjører. Se figur 3.1 for plasseringer på brettet.



Figur 3.1: Ethernut 2.1. 1:RS-232 serieport, 2:Strømforsyning, 3:Ethernet-modul, 4:JTAG-grensesnitt, 5:ADC-innganger, 6:Digitale inn- og utganger

3.3.2 Sensorer

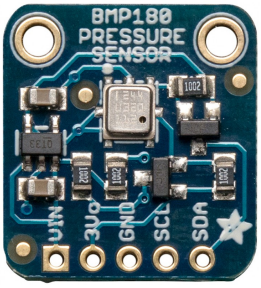
Når det kommer til valg av sensorer vi vil benytte så er det noen kriterier å ta hensyn til:

- Må være kompatibel med mikrokontrolleren.
- Tåle temperaturer helt ned til $-40\,^{\circ}\mathrm{C}$.
- Helst være ferdig utrustet for utemontering.

Vi har satt $-40\,^{\circ}\mathrm{C}$ som den laveste temperaturen sensorene må tåle da den laveste tempteratur i løpet av 2013 var -34,8^{o}C [7].

Ut ifra disse kriteriene har vi bestemt oss for disse sensorene:

(a) SHT10 - temperatur og luftfuktighet



(b) BMP180 - lufttrykk

(c) WG2/O50 - vindhastighet

(d) WRG2/O50 - vindretning

Figur 3.2: Sensorene vi skal bruke

Disse sensorene tilfredsstiller alle krav, bortsett fra BMP180 som ikke har beskyttelse. Til denne må vi få tak i en innkapslingsboks som er vanntett og har tette kabelgjennomføringer. Det må også være en form for luftgjennomstrøming slik at lufttrykket blir riktig målt.

3.3.3 Sensorer - Teknisk informasjon

SHT10[22]

- Måler temperatur og luftfuktighet.
- Digital.
- Operasjonsområde fra $-40\,^{\circ}\mathrm{C}$ til $120\,^{\circ}\mathrm{C}$.
- Fargekode på koblingspunkter hvor grønn:jord, blå:data, gul:klokke og rød:VDD.
- Spenningskilde: min 2.4V og maks 5.5V.

Ytterlig informasjon kan finnes i databladet[23] til SHT10. Hvordan SHT10 kommuniserer med mikrokontrolleren blir beskrevet i delkapittel 4.3.1

BMP180[24]

- Måler lufttrykk.
- Digital.
- Operasjonsområde fra $-40\,^{\circ}\mathrm{C}$ til $85\,^{\circ}\mathrm{C}$.
- Måler lufttrykk fra 300 til 1100 hPa. (9000 til -500m i forhold til meter over havet)

- Koblingspunkter: VIN, 3V0, GND, SCL(data) og SDA(klokkke).
- Klargjort for 5V logikk med integrert 3.3V regulator.

Denne versjonen av BMP180, som vi skal bruke, er allerede klargjort for 5V logikk fra produsenten. Det vil si at koblingspunktet VIN er beregnet for 5V og 3V0 er beregnet for 1.6V til 3.6V. Ytterlig informasjon finnes i databladet[25] til BMP180. Hvordan BMP180 kommuniserer med mikrokontrolleren blir beskrevet i delkapittel 4.3.1.

WG2/O-10[26]

- Måler vindhastighet.
- Analog utgang 0-10V.
- Operasjonsområde fra $-40\,^{\circ}\mathrm{C}$ til $70\,^{\circ}\mathrm{C}$.
- Tåler en vindstyrke på 80 m/s i opp til 30 minutter.
- Måleområdet fra 0-50 m/s.
- Varmeelement for å forhindre at bevegelige deler fryser fast.
- Spenningstilførsel 13-30 V DC eller 24V AC/DC, maks 50 mA.
- Spenningstilførsel for varmeelement 24V AC/DC, maks 20W.
- Det er syv koblingspunkter:
  1. Spenningstilførsel +
  2. Spenningstilførsel -
  3. Analog ut +
  4. Analog ut -
  5. Spenningstilførsel varmeelement +
  6. Spenningstilførsel varmeelement -
  7. GND

WRG2/O-10[27]

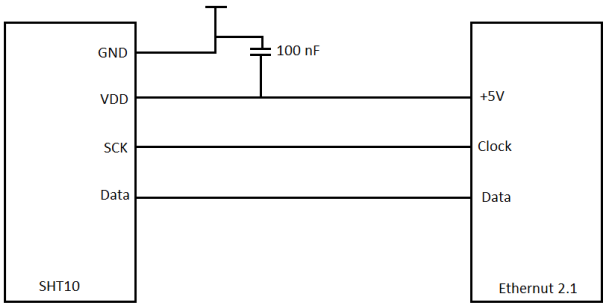
- Måler vindretning.
- Analog utgang 0-10V.
- Operasjonsområde fra $-40\,^{\circ}\mathrm{C}$ til $70\,^{\circ}\mathrm{C}$.
- Tåler en vindstyrke på 80 m/s i opp til 30 minutter.
- Måleområde fra 0-50 m/s.
- Varmeelement for å forhindre at bevegelige deler fryser fast.
- Spenningstilførsel 13-30 V DC eller 24V AC/DC, maks 50 mA.

- Spenningstilførsel for varmeelement 24V AC/DC, maks 20W.
- Det er syv koblingspunkter:
    1. Spenningstilførsel +
    2. Spenningstilførsel -
    3. Analog ut +
    4. Analog ut -
    5. Spenningstilførsel varmeelement +
    6. Spenningstilførsel varmeelement -
    7. GND

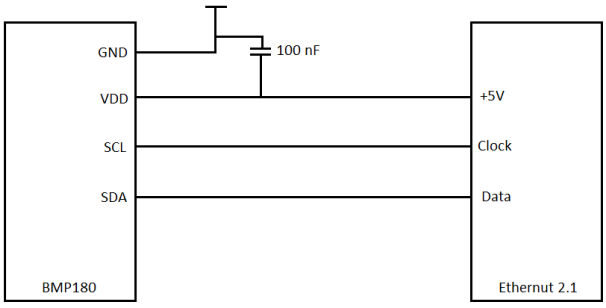
Hvordan avlesningen av WRG2- og WG2-sensoren blir gjort på mikrokontrolleren blir forklart i delkapittel 4.3.1.

3.3.4 Kretsskjema

I figur 3.3 og 3.4 ser du kretsskjema for SHT10 og BMP180. Mikrokontrolleren har egne utganger for tilførsler på 5V. Det vi trengte å vite for å lage kretsene mellom disse sensorene og mikrokontrolleren står godt beskrevet i databladene til sensorene. Disse sensorene skal bruke jordingspunktet på mikrokontrolleren.

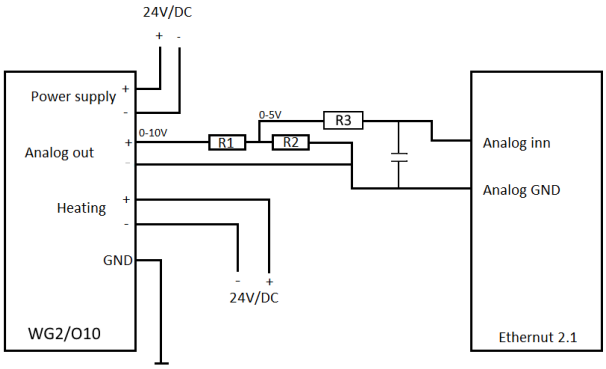


Figur 3.3: Kretsskjema for SHT10

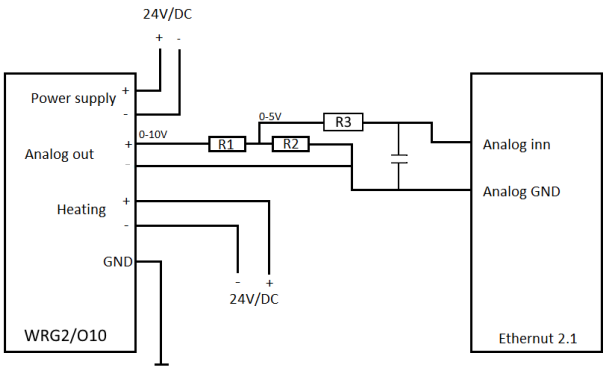


Figur 3.4: Kretsskjema for BMP180

I figur 3.5 og 3.6 ser du kretskjema for de analoge vindsensorene vi har valgt ut. Disse krever en strømforsyning fra 15-30V AC/DC. Vi har valgt å bruke en strømforsyning på 24V DC da denne også kan brukes på varmefunksjonen disse sensorene har.



Figur 3.5: Kretsskjema for WG2/O10



Figur 3.6: Kretsskjema for WRG2/O10

Disse sensorene sender ut en analog spenning på 0-10V og denne spenningen må vi redusere

fordi mikrokontrolleren ikke tåler mer enn 5V. Dette har vi løst ved å bruke en spenningsdeler til å dele spenningen ut fra sensorene på to. Dette gjøres ved at R1 og R2 har lik motstandsverdi. I vårt tilfelle valgte vi R1 og R2 lik 10KΩ. I utregning 3.1 ser du hvordan vi kom frem til dette.

$$U_{ut} = \frac{R_2}{R_1 + R_2} \times U_{inn} \tag{3.1}$$

Hvis $R_1 = R_2$ får vi (3.2)

$$U_{ut} = \frac{1}{2} \times U_{inn} \tag{3.3}$$

$$U_{ut} = \frac{1}{2} \times 10V \tag{3.4}$$

$$U_{ut} = 5V \tag{3.5}$$

Vi har også laget et lavpassfilter for å blokke signaler på mer en 10Hz. Med en grensefrekvens på 10Hz og en kondensator på 100nF kan vi regne ut motstanden vi trenger. I utregning 3.6 finner vi at motstanden til lavpassfilteret må være ca. 159KΩ.

$$f_g = \frac{1}{2\pi \times R_T \times C} \qquad f_g = 10Hz \tag{3.6}$$

$$R_T = \frac{1}{2\pi \times f_g \times C} \qquad C = 100nF \tag{3.7}$$

$$= \frac{1}{2\pi \times 10 \times 100 \times 10^{-9}} \tag{3.8}$$

$$= 159155 \approx 159K\Omega \tag{3.9}$$

For at de analoge og digitale sensorene ikke skal forårsake støy for hverandre har vi valgt å jorde de på forskjellige steder. Som nevt tidligere skal de digitale sensorene bruke jordingspunktet på mikrokontrolleren. Derfor velger vi å jorde de analoge sensorene til et jordingspunkt i kabinettet.

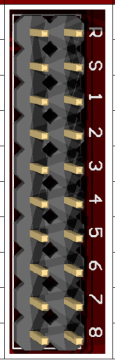
Tilkobling til Ethernut 2.1

SHT10 og BMP180 må kobles på de digitale inn-/utgangene på mikrokontrolleren. SHT10 kan kobles på hvilken som helst ledig port, vi har da valgt port B med klokke på bit 0 (pin 47) og data på bit 1 (pin 48). BMP180 må kobles på mikrokontrollerens TWI-grensesnitt. Vi må da bruke port D med klokke på bit 0(pin 55) og data på bit 1(pin 56). Strømforsyning til begge disse sensorene finner vi på pin 3 og 4, jord finner vi på port 5-8. På figur 3.1 kan vi se en oversikt over plassering til de forskjellige pins.

Pin	Signal	Description		Pin	Signal	Description
1	VCC3	+3.3V regulated		2	VCC3	+3.3V regulated
3	VCC5	+5V regulated		4	VCC5	+5V regulated
5	GND	Signal ground		6	GND	Signal ground
7	GND	Signal ground		8	GND	Signal ground
9	MR\	Reset input		10	DC	Unregulated supply
11	VCC5	+5V regulated		12	VCC5	+5V regulated
13	RD\	Memory read signal		14	WR\	Memory write signal
15	D0	Databus bit 0		16	D1	Databus bit 1
17	D2	Databus bit 2		18	D3	Databus bit 3
19	D4	Databus bit 4		20	D5	Databus bit 5
21	D6	Databus bit 6		22	D7	Databus bit 7
23	A0	Adressbus bit 0		24	A1	Adressbus bit 1
25	A2	Adressbus bit 2		26	A3	Adressbus bit 3
27	A4	Adressbus bit 4		28	A5	Adressbus bit 5
29	A6	Adressbus bit 6		30	A7	Adressbus bit 7
31	A8	Adressbus bit 8		32	A9	Adressbus bit 9
33	A10	Adressbus bit 10		34	A11	Adressbus bit 11
35	A12	Adressbus bit 12		36	A13	Adressbus bit 13
37	A14	Adressbus bit 14		38	A15	Adressbus bit 15
39	PE0	Port E bit 0		40	PE1	Port E bit 1
41	PE2	Port E bit 2		42	PE3	Port E bit 3
43	PE4	Port E bit 4		44	PE5	Port E bit 5
45	PE6	Port E bit 6		46	PE7	Port E bit 7
47	PB0	Port B bit 0		48	PB1	Port B bit 1
49	PB2	Port B bit 2		50	PB3	Port B bit 3
51	PB4	Port B bit 4		52	PB5	Port B bit 5
53	PB6	Port B bit 6		54	PB7	Port B bit 7
55	PD0	Port D bit 0		56	PD1	Port D bit 1
57	PD2	Port D bit 2		58	PD3	Port D bit 3
59	PD4	Port D bit 4		60	PD5	Port D bit 5
61	PD6	Port D bit 6		62	PD7	Port D bit 7
63	NC	Available		64	NC	ALE, if R34 stuffed

Tabell 3.1: Digitale inn-/utganger på Ethernut 2.1

De analoge vindsensorene må kobles på ADC. Her har vi valgt å koble vindhastighetssensoren på port F, bit 0(pin 5). Vindretning har vi koblet på port F, bit 1(pin 7). Oversikten over de analoge inngangene vises på figur 3.2.

Pin	Signal	Description		Pin	Signal	Description
1	AVCC5	+5V regulated		2	AREF	Voltage reference
3	AVCC5	+5V regulated		4	AGND5	Analog ground
5	PF0	Port F bit 0		6	AGND5	Analog ground
7	PF1	Port F bit 1		8	AGND5	Analog ground
9	PF2	Port F bit 2		10	AGND5	Analog ground
11	PF3	Port F bit 3		12	AGND5	Analog ground
13	PF4	Port F bit 4		14	AGND5	Analog ground
15	PF5	Port F bit 5		16	AGND5	Analog ground
17	PF6	Port F bit 6		18	AGND5	Analog ground
19	PF7	Port F bit 7		20	AGND5	Analog ground

Tabell 3.2: Analoge innganger på Ethernut 2.1

3.3.5 Kabinettet til værstasjonene

Det er behov for et kabinett som kan beskytte mikrokontrolleren, strømforsyningene og kretsene mellom sensorene og mikrokontrolleren.

Viktige kriterier for valg av kabinett:

- Vanntett
- Godt isolert
- Beregnet for veggmontering
- Mulighet for varmeelement og termostat
- Monteringsplate for å skru fast komponenter
- Størrelsesmessig så lite som mulig

Med dette tatt i betrakning har vi valgt å gå for et skap i metall med monteringsplate i stål og en dør som er fullt forseglet. Målene på skapet: høyde: 30cm, bredde: 20cm, dybde: 15,5cm.

Figur 3.7: Kabinett med dør

Dette skapet(3.7) har en beskyttelsesgrad på IP 66, *International Protection Marking*[28], hvilket vil si at det kvalifiserer meget godt som kabinett for utendørsmontering. Eneste vi må forbedre er isolasjonsevnen. Dette kan løses ved å bruke isopor innvendig. I tillegg til etterisolasjon skal vi også montere et termostatstyrt varmeelement for å være helt sikker på at det ikke oppstår kondens eller at komponentene fryser.

3.4 Programvare

3.4.1 Databasen

Databasen ment til å ta vare på værdata skal være lokalisert på en av skolens servere. Vi har valgt å bruke den populære MySQL(3.8)-databasen med InnoDB-motoren. Denne beslutningen er tatt på bakgrunn av at vi har god kjennskap til MySQL og InnoDB fra tidligere kurs og at det er en av

verdens meste brukte åpen kildekode database[29]. Vi kommer til å overholde minst første, andre og tredje normalform fra normaliseringsreglene i databasemodellen vår.

Figur 3.8: MySQL-logo

3.4.2 Visualisering av værdata

I vårt valg av hvilke teknologier vi skulle bruke for å presentere værdata på hessdalen.org, har vi tatt utgangspunkt i de webprogrammeringsspråkene vi har kunnskap om fra før. Det vil si CGI-basert Python(3.9), JavaScript(3.10) og PHP(3.11). For å best kunne vurdere hva vi skulle velge, satte vi opp en liste med fordeler og ulemper ved de forskjellige metodene.

CGI-basert Python[30] [31]

Fordeler

- Den fulle kraften av Python implementert i nettsiden.
- Mange muligheter, og en stor mengde biblioteker tilgjengelig.

Ulemper

- Hvis man ikke bruker alternative måter å kjøre CGI på, kan det fort bli treg utførelsetid på serveren. [32]
- All kode kjøres på serveren, noe som fører til høyere belastning.

Figur 3.9: Python-logo

JavaScript[33]

Fordeler

- Laget for webprogrammering.
- Stor mengde relevante biliotek for presentering av data (grafer, diagrammer, osv.)
- God samhandling med AJAX-teknologi for sømløs lasting av innhold.

- Utbredt språk, så det er mye god dokumentasjon. Kode kjøres av nettleser, noe som fører til mindre belastning på server.

Ulemper

- All kildekode er tilgjengelig for alle som leser nettsiden. Man kan altså ikke bruke JavaScript til sensitiv informasjon.
- Krever litt mer feilsøking, men Firebug (utvidelse for debugging til Firefox og Chrome) ordner dette relativt greit.

Figur 3.10: JavaScript-logo

PHP: Hypertext Preprocessor[34]

Fordeler

- Enkel syntaks og feilsøking.
- Tilgivende språk (det meste fortsetter å kjøre selv om man har kodefeil).

Ulemper

- Hatt mange problemer med sikkerhet i sin levetid.[35]
- All kode kjøres på serveren, noe som fører til høyere belastning.

Figur 3.11: PHP-logo

På bakgrunn av dette har vi bestemt oss for å presentere data ved å hovedsaklig bruke JavaScript med AJAX-teknologi og biblioteket 'Google Visualisation' for tegning av grafer. Som back-end for å hente værdata fra databasen og konvertere til JSON trenger vi et språk der kildekode blir kjørt på serveren, og ikke er synlig for bruker. Til dette har vi valgt PHP ettersom det er et utbredt og relativt raskt språk til bruk på serversiden.

3.4.3 Serversiden

Værstasjonen vil sende data som TCP-pakker ment for en bestemt IP og port på skolens server. For å ta imot disse pakkene og sørge for at data blir lagt inn i databasen riktig, må vi lage et program som håndterer dette. Her sto vi ganske åpent på hva slags platform vi skulle bruke, men vi valgte å bygge dette på det populære språket Java(3.12 av to grunner. Java versjon 1.6[36] er allerede installert på skolens server der dette programmet skal kjøre. Den andre grunnen er at Java er blitt veldig populært[37], og god mengde dokumentasjon er tilgjengelig på internett.

Figur 3.12: Java-logo

Serverprogramvaren vi skal bygge vil døgnet rundt ligge og vente på datapakker fra værstasjonen. Når den mottar en pakke med værdata vil denne informasjonen bli lagt inn i databasen med en gang. Vi planlegger å bygge dette med internett-sockets og en objektorientert løsning der andre klasser vil konvertere data fra JSON til spørringer som kan kjøres inn i databasen.

3.4.4 Programvare på Ethernut 2.1

Mikrokontrolleren Ethernut 2.1 skal spille en sentral rolle i systemet. Det er dens oppgave å lese av sensorene, og sende målingene til server for permanent lagring. På mikrokontrolleren kommer vi til å benytte det lille operativsystemet NutOS. Vår applikasjon kommer til å bli skrevet i programmeringsspråket C.

NutOS er et Open Source RTOS(Real Time Operating System) som er laget for mikrokontrollerfamilien Ethernut. Inkludert er også Nut/Net som er en implementasjon av TCP/IP-stakken. Funksjonalitet inkluderer støtte for tråder, hendelser, tidtakere og dynamisk allokering av minne. Nut/Net har støtte for protokoller som TCP(Transmission Control Protocol), IP(Internet Protocol), UDP(User Datagram Protocol), DHCP(Dynamic Host Configuration Protocol). Det inkluderer også et API for sockets. Vi har mulighet til å konfigurere operativsystemet for vårt bruk. Det følger med programvare som gjør det mulig å gjøre en rekke valg før man kompilerer operativsystemet.[38]

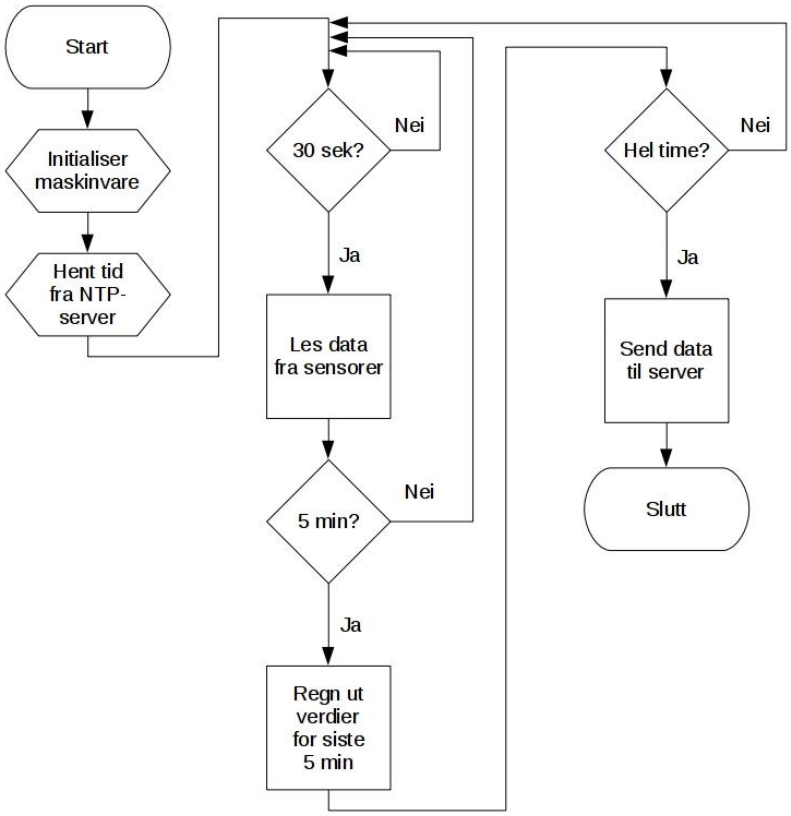
For å skrive vår applikasjon må vi benytte en kompilator, vi kommer til å bruke AVR-GCC(AVR-GNU Compiler Collection). Det er en kompilator som gir oss en rekke bibliotek som brukes med GCC. GCC er originalt skrevet for GNU operativsystemet.[39]

Under planlegging kom oppdragsgiver med følgende krav til funksjonalitet på mikrokontrolleren:

- Mikrokontrolleren skal gjøre ti målinger hvert femte minutt, og skal sende data for den siste timen til server. Fra disse målingene skal den høyeste og laveste verdi fjernes før det regnes ut gjennomsnitt. I tilfellet støy skulle ødelegge for målingene.
- Watchdog skal benyttes slik at mikrokontrolleren starter om hvis noe skulle gå galt, eller prosessoren skulle henge seg.

- Klokken på mikrokontrolleren skal stille seg selv ved å kontakte en NTP(Network Time Protocol)-server. Her har vi valgt å benytte NTP Pool Project sin server i Norge.

Ut ifra disse spesifikasjonene kan vi lage et flytskjema som beskriver hvordan applikasjonen skal fungere. Se figur 3.13



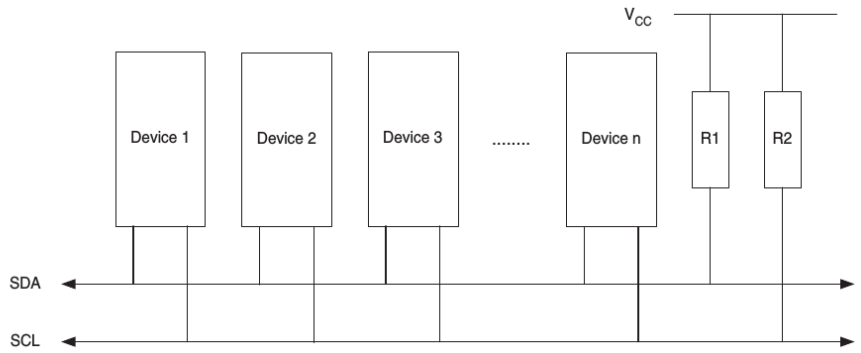
Figur 3.13: Flytskjema for applikasjon på mikrokontrolleren

3.4.5 Grensesnitt mot sensorene

BMP180

BMP180 benytter den kjente kommunikasjonsbusen I^2C (Inter-Integrated Circuit), også kalt TWI (Two-wire Serial Interface). Et TWI maskinvaregrensesnitt er tilgjengelig på mikrokontrolleren.

I^2C er en flerbruker seriell databuss som benytter seg av et master-slave system. Vår implementasjon krever kun en master, Ethernut 2.1, men alle enheter har mulighet til å være master i en slik buss. Som vi ser på figur 3.14 kobles alle enheter til to kabler, som er SCL (klokken) og SDA (data). Klokken styres av master når det foregår kommunikasjon. Hver enhet koblet til bussen har en adresse på 7 bit pluss en bit som bestemmer om man ønsker å skrive til, eller lese fra enheten.

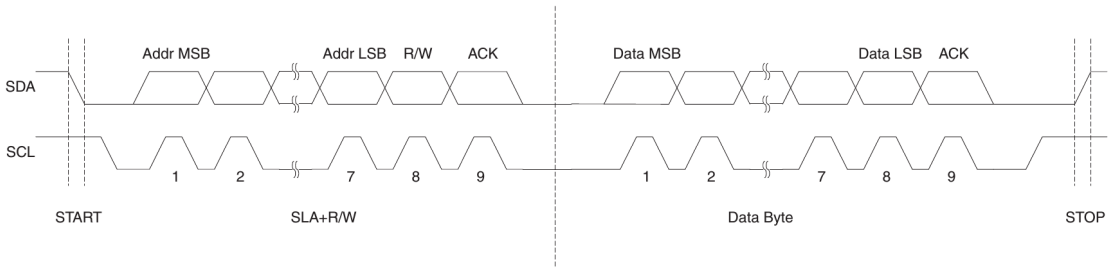


Figur 3.14: I^2C databuss

Det er fire forskjellige moduser for hver enhet tilkoblet bussen:

- Master transmit, hvor master sender data til slave.
- Master recieve, hvor master mottar data fra slave.
- Slave transmit, hvor slave sender data til master.
- Slave recieve, hvor slave mottar data fra master.

Som vi ser på figur 3.15 starter typisk kommunikasjon mellom to enheter koblet til bussen med at en master sender et START-signal, da er master i transmit-modus. Dette etterfulges med å sende en 7-bit adresse(SLA), som er adressen til enheten den ønsker å kommunisere med. Deretter en bit som bestemmer om vi skal skrive til sensoren(SLA+W) eller motta fra sensoren(SLA+R). Slaven svarer da master med en acknowledge(ACK). Hver databyte sendt vil da bli svart på med en ACK, unntatt siste byte. Etter ferdig sending, kan master igjen velge å sende et (RE)START-signal hvis den ønsker å kommunisere mer. Det må gjøres hvis for eksempel master ønsker å motta informasjon fra slaven. Da må man først sende adressen til registeret hos slaven man ønsker å lese fra, før man sender (RE)START. Slaven vil da sende data som ligger i det registeret. Master avslutter kommunikasjon ved å sende et STOP-signal.[25][40]



Figur 3.15: Typisk I^2C dataoverføring

Fra datablad til BMP180 ser vi at mikrokontrolleren først må lese kalibreringsparametere som er lagret på sensoren. Deretter må sensoren få beskjed om å måle lufttrykket, før mikrokontrolleren leser av råverdien. Mikrokontrolleren må da bruke både råverdien og kalibreringsparameterene til å kalkulere lufttrykket. Implementasjonen er forklart i detalj i kapittel 4.3.1

SHT10

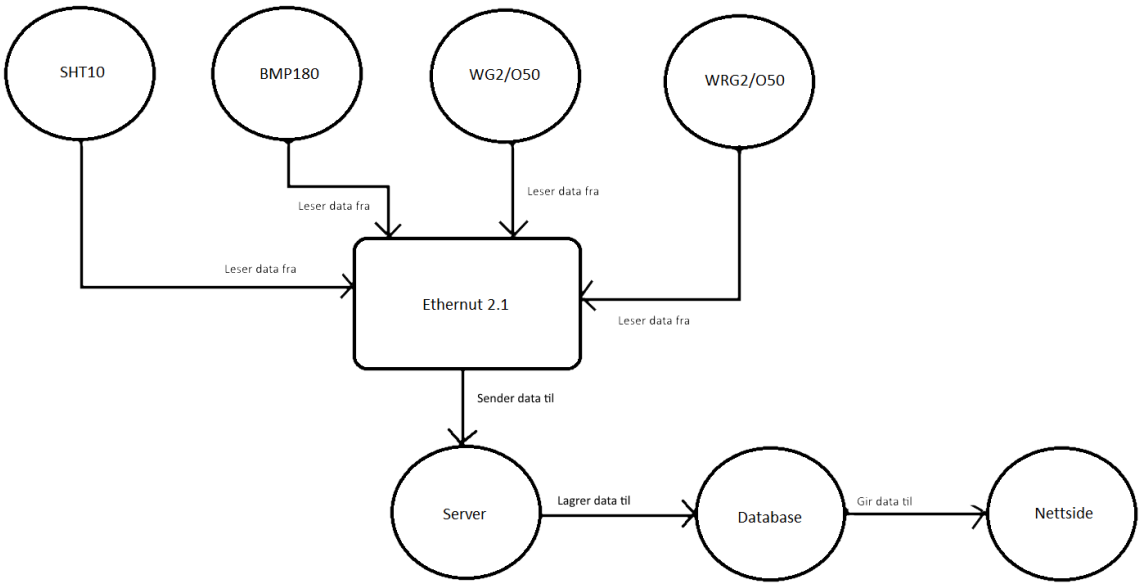
SHT10 benytter sin egen kommunikasjonsprotokoll som er relativt lik I^2C. Protokollen er dokumentert i sensorens datablad. Kommunikasjon med sensoren må implementeres ved å bruke såkalt bit banging, en teknikk for seriekommunikasjon. Det går ut på å manuelt veksle de digitale utgangene i programvare, istedet for å bruke dedikert maskinvare. På grunn av mangel på dedikert maskinvare vil dette påvirke ytelsen, prosessoren må gjøre all jobb ved kommunikasjon. SHT10 må styres med to utganger, SCK(klokke) og DATA.

Mikrokontrolleren må først sende en kommando til sensoren som bestemmer hva som skal måles. Før sensoren sender råverdien til mikrokontrolleren. Denne råverdien brukes sammen med kalibreringsparametere, som er dokumentert i datablad, for å regne ut temperatur og luftfuktighet.[23]

Vindretning- og hastighet

Vindretning- og hastighetsensorene er analoge. Disse vil vi lese av ved hjelp av mikrokontrolleren ADC (Analog to Digital Converter). Dette er en maskinvare-modul som er en del av mikrokontrolleren. ADC-en er dokumentert i mikrokontrollerens datablad. Her må vi kun lese av verdien på inngangene og konvertere de til respektive verdier.

3.5 Oversikt av hele systemet



Figur 3.16: Systemoversikt

I figur 3.16 ser du en oversikt over systemet og hvilke komponenter som kommuniserer med hverandre. Mikrokontrolleren leser av sensorene, behandler dataene og sender disse til serveren. På serveren blir dataene lagret i en database. Disse dataene blir hentet ut og visualisert på Project Hessdalen sin nettside.

Kapittel 4

Produksjon

4.1 Arbeidsprosessen

Vi har valgt å dele ansvaret mellom gruppemedlemmene på følgende måte:

- Mikael Grimstad
  - Serverside.
  - Visualisering av værdata.
- Kristian Karlsen
  - Planlegge fysisk beskyttelse av værstasjon og sensorer.
  - Lage maskinvarekretser.
  - Utstyrsbestilling.
- Kristoffer Jensen
  - Programmering av mikrokontroller.
  - Lage maskinvarekretser.
- Morten Lindstad
  - Bakgrunnsanalyse.
  - Hjelpe til der det trengs.

Utover dette har alle i gruppen vært med på å sette sammen værstasjonene og installere de i Hessdalen.

4.2 Maskinvare

4.2.1 Tilpassing for montering i felt

Vi har laget to værstasjoner som skal monteres på to forskjellige steder. Den ene skal festes til en mast ved hovedstasjonen og den andre skal festes til et tre ved sekundærstasjonen.

Vindsensorene

Som tilbehør til vindsensorene fikk vi tak i en monteringsstang, vist i figur 4.1, fra samme produsent som er beregnet for disse sensorene. Sensorene festes og løsnes enkelt fra monteringsstangen ved å skru på to låseskiver som sitter rett over ledningen ut fra sensorene. Selve monteringsstangen festes til ønsket plass ved hjelp av to metallbånd og tilhørende låsemekanisme.

Figur 4.1: Monteringsstang for vindsensorene

BMP180

Lufttrykksensoren BMP180 ble vi nødt til å montere i en koblingsboks da den ikke har noen form for beskyttelsesetui. Dette ble gjort ved å lodde fast sensoren til et koblingsbrett som ble skrudd fast inne i koblingsboksen. For å sikkre en tett kabelgjennomføring benyttet vi kabelnipler. Sensoren er også avhengig av luftgjennomstrøming i boksen for at måling av lufttrykket skal bli korrekt. Dette ble løst ved å lage et lite hull i siden og et i bunnen for drenering av vann som kan renne inn.

Festeanordningen til luftrykksensoren ved hovedstasjonen lagde vi av et bøybart metallbånd med hull i. Dette ble festet til boksen og deretter til masten ved å bøye metallbåndet rundt stolpene i masten og skru det fast med maskinskruer. Ved sekundærstasjonen ble boksen festet rett til treet ved bruk av treskruer. Dette ser du på figur 4.2 i avsnitt 4.2.2.

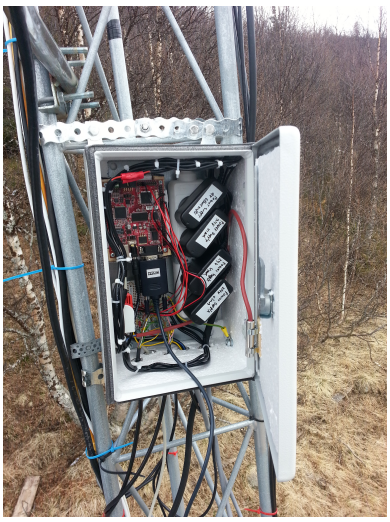
SHT10

For at temperaturmålingene ikke skal bli påvirket av direkte sollys har vi valgt å feste sensoren i en hvit koblingsboks. Koblingsboksen har tillstrekklig med åpninger for gjennomlufting slik at luften inne i boksen holder samme temperatur som på utsiden. Måten den er montert ved de to stasjonene er gjort på samme vis som ved BMP180, beskrevet i avsnittet ovenfor.

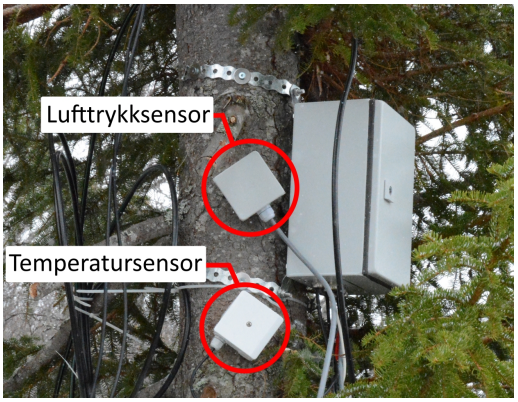
4.2.2 Tilpassing av kabinettet

Det vi valgte å bruke som kabinett på værstasjonen er et skap av stål med beskyttelsesgrad IP 66. Dette er mer en tilstrekklig for utendørs montering. Vi har også gjort ett par tillegsmodifikasjoner. Innvendig har vi isolert det med isopor og montert en termostat som styrer et varmeelement for å forhindre kondens og frost. Termostaten slår på varmeelementet ved $5\,^{\circ}\mathrm{C}$ og av ved $15\,^{\circ}\mathrm{C}$.

Komponentene som skal huses i kabinettet er skrudd fast i tilhørende monteringsplate som fulgte med kabinettet. Selve festeanordningen til kabinettetene for å feste de ved hovedstasjon og sekundærstasjonen er laget av metallbånd med skruehull. På figur 4.2 ser du hvordan dette ble gjort.



(a) Hovedstasjon



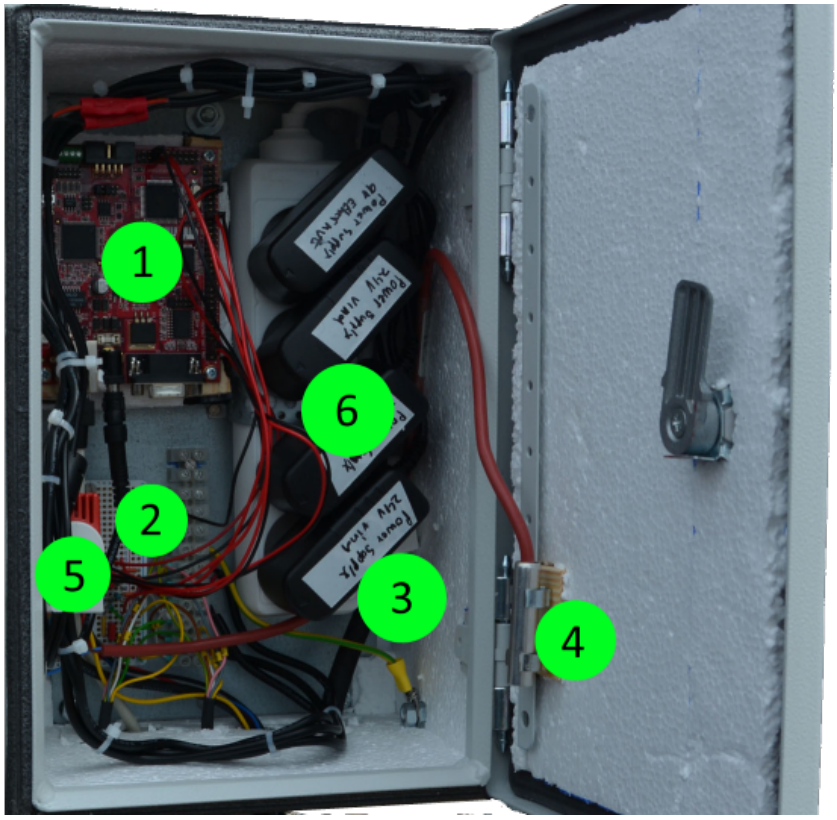
(b) Sekundærstasjon

Figur 4.2: Værstasjonene montert ved målestasjonene

Innredning i kabinettet

I figur 4.3 ser du hvordan vi har innredet kabinettet. Vi har tatt høyde for at kablene til strømforskyningene kan forårsake støy som kan påvirke avlesningen av sensorene. Derfor har vi lagt disse mest mulig utenom mikrokontrolleren, koblingsbrettet og sensorledningene.

1. Mikrokontroller (Ethernut 2.1)
2. Koblingsbrett og sukkerbit
3. Hovedinntak 220V
4. Varmeelement
5. Termostat
6. Strømforsyninger



Figur 4.3: Innredning i kabinettet

Varmeelementet er festet til en trekloss i lokket fremfor direkte til monteringsstripa i lokket. Slik unngår vi varmetapet som hadde oppstått om vi hadde festet det direkte til monteringsstripa. Denne plasseringen var også den mest taktiske for å unngå at strålingsvarmen fra elementet kan komme til å smelte i stykker komponenter rundt, og for å få plassert det så lavt som mulig.

Termostaten, som styrer varmeelementet, er festet lavt i sideveggen på motsatt side av varmeelementet og strømforsyningene. Dette for å unngå at den blir påvirket av andre faktorer enn luften inne i kabinettet og fordi det er her det blir lavest temperatur. Det gjør også at varmeelementet blir slått på så rask temperaturen faller under $5\,^{\circ}\mathrm{C}$.

På koblingsbrettet i kabinettet finner du kretsene til sensorene. Alt av ledninger fra mikrokontrolleren og sensorene er koblet sammen på dette koblingsbrettet, bortsett fra strømforsyningene til vindsensorene. Disse kobles sammen i sukkerbiten. Dette for å skille mellom kretsene på 5V og 24V.

Ledningene fra sensorene kommer inn i bunnen av kabinettet. Isolasjonen rundt ledningene følger helt inn i kabinettet og er ikke avisolert mer enn at ledningene når sine innganger på koblingsbrettet og i sukkerbiten . Alt av komponenter og ledninger på koblingsbretter er loddet fast.

Kabelen som leder 230V inn i kabinettet er ført inn helt til høyre. Dette for å isolere den best mulig bort fra sensorledningene. Her følger isolasjonen et godt stykke inn. Som du ser på figur 4.3 går den inn bak grenuttaket før den avmantles og kobles til grenuttaket. Dette har vi gjort ved å koble de sammen i en sukkerbit som er beskyttet av en kapsel.

4.2.3 Problemer underveis

Før vi isolerte kabinettet med isopor, brukte vi et materiale av elastisk plastikk. Grunnet isolasjonens elastiske egenskap gjorde det enkelt å få de utskjærte platene hele inn i kabinettet. I motsetning til isoporen som vi var nødt til å dele i mindre biter for å få på plass. Den isolerte heller ikke godt nok for vår bruk. Dette medførte at vi valgte isopor som isolasjon selvom den var litt mer trøblete å få inn i kabinettet.

Før vi reiste opp til Hessdalen for å montere værstasjonen opplyste oppdragsgiver oss om at nettverksforbindelsen mellom hoved- og sekundærstasjonen er brutt. Nettverksforbindelsen går i en fiberkabel. Årsaken til brudd på forbindelsen var at et dyr hadde tygget i stykker kabelen. Oppdragsgiver fikk tak i en reparatør, men grunnet dårlige værforhold måtte reparasjonen utsettes. Dette gjør at værstasjonen vi monterte ved sekundærstasjonen ikke får sendt målingene sine til serveren før netterverksforbindelsen har blitt reparert.

4.3 Programvare

4.3.1 Ethernut 2.1

Implementasjon av grensesnitt mot BMP180

BMP180 benytter I^2C/TWI for kommunikasjon med mikrokontrolleren. Dette er en egen maskinvaremodul på mikrokontrolleren, som aksesseres gjennom dataregister. Disse registrene er dokumentert i databladet til mikrokontrolleren, og vi må benytte oss av disse fire:

- TWI Bit Rate Register (TWBR)(Figur 4.4)

  Brukes sammen med TWSR for å sette klokkefrekvensen i master-modus.

Bit	7	6	5	4	3	2	1	0	
	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	TWBR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figur 4.4: TWBR fra datablad

- TWI Control Register (TWCR)(Figur 4.5)

  Brukes for å kontrollere TWI-modulen. Her er vi interessert i bit 5, som sender START ut på bussen. Bit 4, som sender STOPP og bit 2 som aktiverer TWI-modulen.

Bit	7	6	5	4	3	2	1	0	
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figur 4.5: TWCR fra datablad

- TWI Status Register (TWSR)(Figur 4.6)

Bit 7-3 brukes for å sjekke status på TWI-modulen. Bit 1-0 er brukt til å skalere klokkefrekvensen.

Bit	7	6	5	4	3	2	1	0	
	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

Figur 4.6: TWSR fra datablad

- TWI Data Register (TWDR)(Figur 4.7)

  I transmit-modus, holder dette registeret på neste byte som skal sendes. I recieve-modus vil dette registeret inneholde data som ble mottatt.

Bit	7	6	5	4	3	2	1	0	
	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	1	

Figur 4.7: TWDR fra datablad

Før vi benytter oss av TWI-modulen må vi først initialisere den. Dette gjøres ved å først sette klokkefrekvensen(SCL). Klokkefrekvensen regner vi ut med formelen 4.1

$$SCL = \frac{CPU\ Klokkefrekvens}{16 + 2 \times TWBR \times 4^{TWPS}} \tag{4.1}$$

Mikrokontrollerens CPU kjører på 14,7456 MHz[21]. Her velger vi TWBR til 17 og TWPS til 1:

```
TWSR = (0<<TWPS1) | (0<<TWPS0); //Setter prescalar til 1.
TWBR = (1<<TWBR4) | (1<<TWBR0); //Setter bit rate til 17.
```

Frekvensen havner da på cirka 97 kHz. Vi valgte denne frekvensen fordi kommunikasjon ved høyere frekvenser kan bli påvirket av støy og avstanden kan også bli redusert[41]. Vår applikasjon er heller ikke tidskritisk, vi skal kun benytte sensoren hvert 30. sekund.

Deretter starter vi TWI-modulen ved å skrive til TWCR:

```
TWCR = (1<<TWEN); //Aktiverer TWI.
```

Når vi skal kalkulere lufttrykket må vi først lese av kalibreringsparametere som er lagret på sensorens EEPROM(Electrically Erasable Programmable Read-Only Memory). Som vi ser i tabell 4.1 er dette elleve 16-bits registere som befinner seg på adresse 0xAA til og med 0xBF.

	BMP180 reg adr	
Parameter	**MSB**	**LSB**
AC1	0xAA	0xAB
AC2	0xAC	0xAD
AC3	0xAE	0xAF
AC4	0xB0	0xB1
AC5	0xB2	0xB3
AC6	0xB4	0xB5
B1	0xB6	0xB7
B2	0xB8	0xB9
MB	0xBA	0xBB
MC	0xBC	0xBD
MD	0xBE	0xBF

Tabell 4.1: Kalibreringsparametere på BMP180

For å lese ut disse dataene må vi først sende ut START på bussen, det gjør vi med registeret TWCR:

```
TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN); //Sender START.
//Venter på at TWINT er satt.
while((TWCR & (1<<TWINT)) == 0) {
    ...
}
//START har blitt send.
```

Så må vi sende adressen til hvilken enhet vi vil kommunisere med. BMP180 befinner seg på adresse 0xEE for skriving, og 0xEF for lesing[25]. Vi sender adressen ved å laste den inn i TWDR, for så å starte sending med TWCR:

```
//Laster byten inn i data-registeret. Klar for sending.
TWDR = data;

//Starter sending av data i TWDR.
TWCR = (1<<TWINT) | (1<<TWEN);

//Venter på at TWINT er satt.
while((TWCR & (1<<TWINT)) == 0) {
    ...
}
//Data har blitt sendt.
```

Deretter sender vi hvilken adresse vi vil lese av. BMP180 vil da svare med verdien i registeret.

STOP sender vi når vi er ferdig med å kommunisere, ved å skrive følgende:

```
TWCR = (1<<TWINT) | (1<<TWSTO) | (1<<TWEN); //Sender STOP.
```

Etter vi har mottatt og lagret alle kalibreringsparametere må vi sende en kommando til BMP180 som starter lesing av lufttrykk og temperatur. Grunnen til at vi også må lese temperatur er at den er en faktor i utregningen av lufttrykk. Fra tabell 4.2 kan vi se at verdien 0x2E må skrives til kontrollregisteret 0xF4 for å måle temperatur. BMP180 vil da bruke minst 4.5ms på å fullføre målingen. Når BMP180 har gjort seg ferdig, vil rådata ligge i registeret 0xF6.

Når vi skal lese av rådata for lufttrykket må vi også velge hvilken nøyaktighet vi ønsker. Dette er bestemt av OSS(Oversampling Setting) som er et tall mellom 0-3. Som vi ser i tabell 4.2 bruker BMP180 lengre tid på å måle når vi ønsker bedre nøyaktighet.

Measurement	Control register value (register address 0xF4)	Max. conversion time [ms]
Temperature	0x2E	4.5
Pressure (oss = 0)	0x34	4.5
Pressure (oss = 1)	0x74	7.5
Pressure (oss = 2)	0xB4	13.5
Pressure (oss = 3)	0xF4	25.5

Tabell 4.2: Tidsbruk for BMP180

Vi har valgt å benytte OSS=3 for best nøyaktighet. Det innebærer av vi må skrive verdien 0xF4 til kontrollregisteret 0xF4 og vente i minst 25.5 ms før vi leser 2 byte fra registeret 0xF6. Siden vi har valgt best nøyaktighet er det også en ekstra byte tilgjengelig i registeret 0xF8. Denne råverdien blir en del av en rekke utregninger som er beskrevet detaljert i datablad. [25]

Implementasjon av grensesnitt mot SHT10

SHT10 benytter sin egen protokoll for kommunikasjon som må implementeres med bit banging, det innebærer å styre SCK(klokken) og DATA manuelt i programvaren. For å styre de to utgangene, og ha mulighet til å endre DATA til inngang/utgang i programmet. Kommer vi til å benytte oss av et GPIO(General-purpose input/output)-bibliotek inkludert i NutOS.

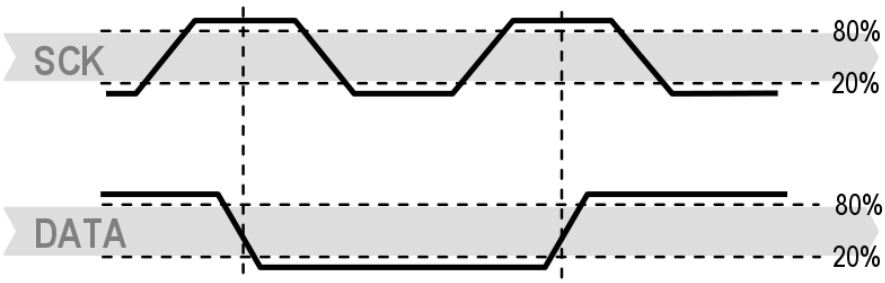
Biblioteket brukes med enkle funksjonskall, hvis vi ønsker å sette klokkeutgangen høy gjøres det slik:

```
GpioPinSetHigh(PORT_B, SCK_PIN);
```

eller sette datautgangen til inngang:

```
//Setter data til input og aktiverer pullup-motstand.
GpioPinConfigSet(PORT_B, DATA_PIN, GPIO_CFG_PULLUP);
```

I databladet finner vi hvordan vi skal kommunisere med SHT10. Før en kommando sendes må vi først initialisere kommunikasjon med en transmission-start som visst på figur 4.8.



Figur 4.8: Transmission-start

Dette implementeres i C på følgende måte:

```
#define SCK_LOW GpioPinSetLow(PORT_B, SCK_PIN)
#define SCK_HIGH GpioPinSetHigh(PORT_B, SCK_PIN)
#define DATA_LOW GpioPinSetLow(PORT_B, DATA_PIN)
#define DATA_HIGH GpioPinSetHigh(PORT_B, DATA_PIN)

void start_transmission(void)
{
    SCK_LOW;
    set_data_output();
    DATA_HIGH;
    PULSE_SHORT; //Venter i 3ms.

    SCK_HIGH;
    PULSE_SHORT;

    DATA_LOW;
    PULSE_SHORT;

    SCK_LOW;
    PULSE_SHORT;

    SCK_HIGH;
    PULSE_SHORT;

    DATA_HIGH;
    PULSE_SHORT;

    SCK_LOW;
    PULSE_SHORT;

    set_data_input();
```

}

Etter transmission-start sendes en 8-bits kommando hvor de første 3 er adresse-bits og de siste 5 er kommando-bits. I tabell 4.3 ser vi en oversikt over kommandoene.

Command	Code
Reserved	0000x
Measure Temperature	**00011**
Measure Relative Humidity	**00101**
Read Status Register	00111
Write Status Register	00110
Reserved	0101x-1110x
Soft reset, resets the interface, clears the status register to default values. Wait minimum 11 ms before next command	**11110**

Tabell 4.3: Kommandoer til SHT10

Etter vi har sendt ut en kommando, må vi vente i maks 80/320ms for en 12/14-bit måling[23]. Fuktighet måles med 12-bit og temperatur med 14-bit nøyaktighet. SHT10 signaliserer at den er ferdig ved å dra DATA-linjen lav. Da kan vi lese verdien fra sensoren ved å sette SCK høy og styre den til vi har lest all data.

For å finne temperaturen bruker vi formel 4.2, hvor SO_T er målt råverdi.

$$T = d_1 + d_2 \times SO_T \tag{4.2}$$

Parameterene får vi fra datablad som vist i tabell 4.4. Vi benytter VDD=5V og 14-bit nøyaktighet.

VDD	d_1 (°C)	d_1 (°F)
5V	-40.1	-40.2
4V	-39.8	-39.6
3.5V	-39.7	-39.5
3V	-39.6	-39.3
2.5V	-39.4	-38.9

SO_T	d_2 (°C)	d_2 (°F)
14bit	0.01	0.018
12bit	0.04	0.072

Tabell 4.4: Parametere for temperatur

Når vi skal måle relativ luftfuktighet må vi benytte formelen 4.3, hvor SO_{RH} er målt råverdi.

$$RH_{linear} = c_1 + c_2 \times SO_{RH} + c_3 \times {SO_{RH}}^2 \tag{4.3}$$

Parametere for formelen finner vi i datablad og er vist i tabell 4.5. Vi måler med 12-bit nøyaktighet.

SO_{RH}	c_1	c_2	c_3
12 bit	-2.0468	0.0367	-1.5955E-6
8 bit	-2.0468	0.5872	-4.0845E-4

Tabell 4.5: Parametere for lufttrykk

Siden vi også har målt temperatur kan vi nå regne ut relativ luftfuktighet kompansert med temperatur. Dette gjør vi i formel 4.4. Her er $T_{^oC}$ temperaturen vi regnet ut tidligere. Med parametere i tabell 4.6.

$$RH_{true} = (T_{^oC} - 25) \times (t_1 + t_2 \times SO_{RH}) + RH_{linear} \tag{4.4}$$

SO_{RH}	t_1	t_2
12 bit	0.01	0.00008
8 bit	0.01	0.00128

Tabell 4.6: Parametere for lufttrykk kompansert med temperatur

Implementasjon av ADC

For å bruke ADC må man benytte følgende dataregister:

- ADC Multiplexer Selection Register(ADMUX)(Figur 4.9)

  Brukes til å bestemme referansespenning og hvilken kanal vi vil lese av. Her er vi interessert i bit 7-6 som velger referansespenning, og bit 4-0 som bestemmer hvilken kanal vi vil benytte.

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figur 4.9: ADMUX-registeret

- ADC Control and Status Register (ADCSRA)(Figur 4.10)

  Brukes til å kontrollere ADC. Her er vi interessert i bit 7 som starter ADC, og bit 6 som starter lesing på analog inngang. Bit 2-0 benyttes til å bestemme skalering på klokkefrekvensen til ADC.

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figur 4.10: ADCSRA-registeret

- ADCH og ADCL(Figur 4.11)

  Disse holder på den innleste verdien.

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Figur 4.11: ADCH og ADCL

Når vi vil lese av verdien på en analog inngang må vi først bestemme referansespenning og klokkehastighet. Referansespenningen bestemmer blant annet høyeste spenning som kan måles. Vi har valgt å benytte mikrokontrollerens interne spenning på 5 volt. Det gjør vi ved å skrive til ADMUX:

```
ADMUX = (1 << REFS0); //Setter Vref til AVCC = 5V.
```

Fra datablad har vi at klokkefrekvensen til ADC må settes til lavere enn 200kHz hvis vi ønsker en nøyaktighet på 10 bit[40]. Klokkefrekvensen regner vi ut i formel 4.5, hvor klokkefrekvensen til CPU er 14,7456 MHz[21].

$$Klokkefrekvens = \frac{CPUklokkefrekvens}{ADCPrescaler} \tag{4.5}$$

ADC Prescaler setter vi til 128 i registeret ADCSRA, det fører til en klokkefrekvens på ca. 115 kHz. For å initialisere ADC skriver vi til ADCSRA:

```
//Skrur på ADC og setter ADC-klokkehastighet til 115.2kHz.
ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
```

Nå må vi velge hvilken kanal vi vil lese av. Dette gjør vi ved å skrive verdi vist i tabell 4.7 til ADMUX.

MUX4..0	Single Ended Input
00000	ADC0
00001	ADC1
00010	ADC2
00011	ADC3
00100	ADC4
00101	ADC5
00110	ADC6
00111	ADC7

Tabell 4.7: Kanalvalg på ADC

Eksempelvis kan vi velge kanal ADC1 og lese fra denne ved å skrive til ADCSRA. Så venter vi på at konverteringen er ferdig og leser verdien fra ADCL og ADCH:

```
//Endrer til kanal ACD1.
ADMUX = (1 << REFS0) | (1 << MUX0);

//Starter lesing
ADCSRA |= (1 << ADSC);

//Venter på at lesing er ferdig. Da er ADSC satt til 0.
while (ADCSRA & (1 << ADSC));

//Leser verdien på ADC.
lsb = ADCL;
msb = ADCH;

//Innlest verdi.
raw = (msb << 8) | (lsb);
```

Råverdien vi leser fra ADC er et tall mellom 0-1023. For å finne spenningen bruker vi formel 4.6.

$$Spenning = \frac{Råverdi \times Ref.spenning}{1023} \tag{4.6}$$

Nå som vi har regnet om til spenning kan vi konvertere dette til vindhastighet med formel 4.7 og vindretning med formel 4.8.

$$Vindhastighet = \frac{Spenning \times Makshastighet}{Ref.spenning} \tag{4.7}$$

Maks hastighet fra vindhastighetsensoren er 50 m/s. [42]

$$Vindhastighet = \frac{Spenning \times Maksretning}{Ref.spenning} \tag{4.8}$$

Maks hastighet fra vindretningsensoren er 360 grader. [43]

Implementasjon av nettverkskommunikasjon

Nettverksfunksjonalitet er implementert i NutOS, og vi trenger kun enkle funksjonskall for å benytte den. Før vi begynner å bruke ethernet-modulen må den først initialiseres. Vi skal også benytte DHCP for å få tak i IP-adresse. Dette gjøres slik:

```
//Registrerer ethernet-kontroller.
NutRegisterDevice(&DEV_ETHER, 0, 0);

//Konfigurerer DHCP.
NutDhcpIfConfig(DEV_ETHER_NAME, mac, 0);
```

Vår applikasjon er avhengig av å holde orden på tiden. For å hente tid fra en NTP server gjør vi slik:

```
//Holder på tiden
time_t ntp_time = 0;

//Adresse til NTP-server.
uint32_t timeserver = inet_addr("85.252.162.7");

//Henter tid fra server.
NutSNTPGetTime(&timeserver, &ntp_time);

//Setter klokken på Ethernut.
//Ethernut holder nå orden på tiden.
stime(&ntp_time);
```

Mikrokontrolleren teller nå tiden selv, og vi trenger kun å gjøre et funksjonskall for å få nåværende tid. Når vi skal sende data til serveren gjøres dette ved hjelp av TCP. NutOS har innebygde funksjoner for TCP, og vi trenger da å bruke en socket. For å benytte en socket gjør vi følgende:

```
//Lager socketen som vi skal bruke.
TCPSOCKET *sock = NutTcpCreateSocket();

//Kobler til server.
NutTcpConnect(sock, inet_addr(FREJA_IP), FREJA_PORT);

//Sender data med bytes-lengde.
NutTcpSend(sock, data, bytes);

//Lukker socket.
NutTcpCloseSocket(sock);
```

Mikrokontrolleren har et begrenset buffer, og kan sende maksimalt 1460 byte på en gang. Når vi ønsker å sende flere bytes i en socket må vi løse dette i en løkke:

```
//Sender data til server. Kan sende maks 1460 bytes av gangen.
//Kjører til alle bytes har blitt sendt.
while ((sent = NutTcpSend(sock, args->data, bytes)) != bytes) {
    //Regner ut resterende bytes som må sendes.
    bytes -= sent;

    //Flytter resterende bytes til begynnelsen i bufferet.
    memcpy(args->data, &args->data[sent], bytes);
}
```

Watchdog

Watchdog er en maskinvaremodul som teller ned fra gitt tid. Hvis applikasjonen ikke starter om nedtelleren vil modulen starte om applikasjonen. I kode implementeres det ved å benytte biblioteket ”avr/wdt.h”:

```
//Setter watchdog-nedteller til 1.8s
wdt_enable(7);

//Starter watchdog på nytt.
wdt_reset();
```

Implementasjon av hovedløkke

Som vi så på figur 3.13 i kapittel 3.4.4 er vår applikasjon avhengig av å gjøre oppgaver på forskjellige tidspunkt. Da vi skal gjøre ti målinger i minuttet, gjør vi en måling hvert 30. sekund. For å sjekke dette i kode gjør vi det enkelt med en løkke:

```
for(;;) { //Evig løkke
//Henter nåværende tid.
    datetime = get_current_time();

    //Dette er en trettisekunder.
    //F.eks 13:31:30, 14:32:00.
    if((datetime->tm_sec % 30) == 0)
        break;

    ...
}
```

Når vi er på en trettisekunder henter vi en måling fra alle sensorer. Disse målingene lagrer vi i arrayer og holder orden på hvor mange målinger vi har tatt. For hver trettisekunder sjekker vi også om dette er en femminutter(f.eks 14:05, 15:40). Da skal vi regne ut gjennomsnitt, maksimum og minimum. Dette gjøres enkelt ved å traversere målingene vi har gjort, for så å lagre de utregnede verdiene i enda et array.

Når vi er på en hel time, skal vi sende data til serveren. Dette gjør vi ved å lage en tråd som håndterer nettverksoperasjonen. Vi er avhengig av å benytte en tråd fordi nettverksoperasjonene tar så lang tid at vi ikke rekker å starte om watchdog-telleren.

Tråder er implementert i NutOS og lages enkelt med THREAD-makroen:

```
//Deklarerer en tråd som håndterer sending av data.
THREAD(DataThread, args)
{
    ...
}
```

Tråden startes ved å gjøre følgende funksjonskall:

```
//Starter tråden som sender data med 512 bytes stack.
NutThreadCreate("DataThread", DataThread, args, 512);
```

Under testing av systemet kom vi over en feil som vi ikke klarte å fikse. Feilen var at nettverksmodulen på mikrokontrolleren sluttet å fungere etter 2-3 sendinger. På grunn av tidsmangel løste vi denne feilen ved å starte om mikrokontrolleren etter at den sendte data hver hele time. Dette er en relativt dårlig løsning, men det løser feilen.

Feilhåndtering

- Ved oppstart vil mikrokontrolleren prøve å få tid fra NTP-server. Den vil prøve 6 ganger før den gir opp og starter om programmet.
- Vi skanner hele TWI-bussen for å se om vi finner BMP180, hvis ikke vil den bli ignorert den neste timen. Da setter vi bare målt trykk til 0.
- SHT10 prøver vi å kontakte ved hver avlesning. Får vi tidsavbrudd setter vi temperatur og luftfuktighet til 0.
- Nettverksrelaterte feil blir funnet ved tidsavbrudd. Vi prøver å sende data 6 x 10 sekunder. Hvis vi ikke får koblet til serveren vil mikrokontrolleren starte om og vi vil miste data fra den siste timen.

4.3.2 Database

Slik beskrevet i avsnitt 3.4.1, bruker vi MySQL med InnoDB for lagring av værdata. I modelleringen av databasen, var hovedfokuset å lage en modell(se figur 4.12) som overholdt første, andre og tredje normalform fra normaliseringsreglene.

Første normalform (1NF)

- Hva: Krever at på hvert krysningspunkt mellom rader og kolonner i tabellen finnes det én enkelt verdi, og aldri en liste over verdier.
- Hvordan: Dette har vi overholdt ved å ha alle ulike verdier i egne kolonner, og kun ha èn måling i hver rad.

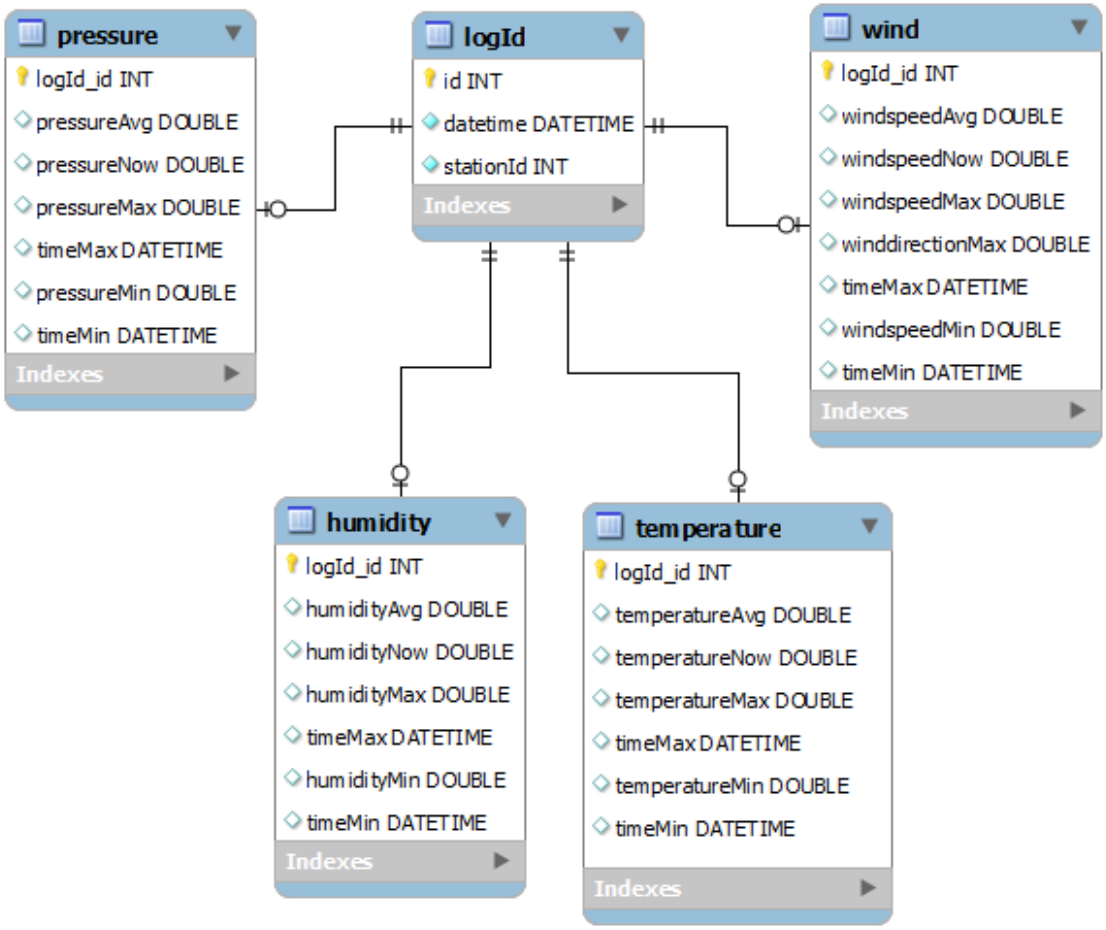
Andre normalform (2NF)

- Hva: Krever at hver ikke-nøkkelkolonne er helt avhengig av hele primærnøkkelen, ikke bare en del av nøkkelen.
- Hvordan: I denne modellen er dette irrelevant, da vi ikke har delte primærnøkler. Tabellen 'logId' kunne vært laget med en delt primærnøkkel på stasjonsnummer og tid, men for enkelhets skyld, valgte vi å lage en egen teller for dette.

Tredje normalform (3NF)

- Hva: Krever at hver ikke-nøkkelkolonne må være avhengig av primærnøkkelen, og bare den.
- Hvordan: Overholdt ved at hver verdi i tabellene kun er avhengige av primærnøkkel.

For at en modell skal overholde 3NF, må den også oppfylle kravene til 1NF og 2NF.



Figur 4.12: Databasemodellen laget for å lagre målingsdata.

4.3.3 Serverside

For å ta imot TCP-pakkene med JSON-formatert data og legge det inn i databasen, var vi avhengig av å ha en server kjørende på skolens nettverk. Denne serveren valgte vi som beskrevet i 3.4.3 å bygge på Java.

Pakkemottak og tråder

For å kunne motta datapakkene værstasjonen sender fra Hessdalen og ned til skolens server, bruker vi internett-sockets. Nærmere bestemt 'TCP Stream Sockets'. Disse tillater pålitelig to-veis kommunikasjon mellom både værstasjonen og serveren. Dette fungerer i praksis slik at serveren venter til en tilkobling er gjort, og data mottatt før den behandler data slik beskrevet under, og sender ≪Done≫ tilbake til værstasjonen. Når serveren er ferdig med dette, er den klar til å motta ny data. Dette måtte imidlertidig bli gjort trådbasert for at begge stasjonene skulle kunne sende på likt. Dermed gjorde vi det slik at så fort en ny tilkobling blir opprettet til serveren, lages det en egen tråd for å behandle denne tilkoblingen. Prosessen er den samme som beskrevet ovenfor, bortsett ifra at serveren kan håndere alle tilkoblinger selv om de kommer på likt.

Konvertere JSON til SQL-spørringer

Vi har laget en egen klasse kalt ≪DbManager≫ for å håndere den mottatte JSON-formaterte datastrengen og bygge SQL-spørringer innsetting i databasen. Her valgte vi å bruke et lite ressurskrevende bibliotek kalt Json-lib. Med dette biblioteket bygger vi et JSON-objekt av strengen, og kan slik lett hente ut målingsdata. Med dette bygger vi SQL-spørringer som kjøres ved hjelp av en klasse laget for å håndere databasen. Det er mer om denne prosessen i neste punkt.

Databasebehandling

Klassen ≪Db≫ har to oppgaver. Den første er å kjøre SQL-spørringene generert i ≪DbManager≫, og den andre er å returnere den autogenererte nøkkelen MySQL lager ved innsetting av data. Denne nøkkelen er svært viktig når det kommer til å få koblet riktige rader i databasen sammen og overholdt restriksjonene for fremmednøkler. Det er mer om databaseoppbygningen i avsnitt 4.3.2.

For at Java-serveren skal kunne koble seg til og håndere databasetilkoblinger, var vi avhengig av å bruke en driver fra MySQL kalt JDBC. Dette er et bibliotek som utvider Java med muligheten for nettopp dette.

Kompilert JAR-fil

Vi har valgt å kompilere den ferdige serveren som en JAR-fil. Dette er gjort i den hensikt å slippe å ha en relativt stor mengde klasser og biblioteker liggende løst der programmet skal kjøres fra. Ved å JAR-kompilere kan vi få med alle klasser og biblioteker i samme fil, og det blir dermed et mye mer ryddig produkt.

Sikkerhet

For at nettverk-sockets skal kunne ta imot tilkoblinger og data utenfor lokalnettet, må den relevante porten åpnes i router/brannmur for tilkoblinger utenfra. I enkelt tilfeller kan det være en sikkerhetsrisiko å ha porter åpne, men det er kun dersom programmet som mottar disse pakkene kan feiltolke innholdet og gjøre noe uforventet. Programmet vi har laget avviser alle pakker som ikke kommer med korrekt formatert JSON etter vår mal, så dette skal ikke være noe problem. Allikevel er det en dobbelsikring på plass. Porten vi bruker er nemlig kun åpnet for data fra de relevante IP-adressene værstasjonene sender fra. Det vil si at tilkoblinger på denne porten blir avvist fra alle andre adresser på internett. Av sikkerhetsårsaker nevner vi verken hvilken port eller hvilke IP-adresser dette gjelder.

Cron

Vi ønsket å garantere at serveren kjører til en hver tid. Vi har derfor tatt i bruk programmet cron. Cron er et program på *nix-operativsystemer som gjør det mulig å planlegge kjøring av et program/skript. Dette gjøres ved å skrive inn når og hva i en crontab(Cron tabell). Systemet vil da gjøre gitt oppgave på gitt tid.[44] Vi har ført opp følgende linje i vår crontab på serveren:

```
59 * * * * /local/hessdalen/hessdalenweather/hws-make-run.sh
```

Skriptet hws-make-run.sh vil da bli kjørt hver hele time på det 59. minuttet. Dette passer bra da mikrokontrolleren sender data hver hele time. Skriptet sjekker om vår server kjører, og starter den dersom den ikke gjør det:

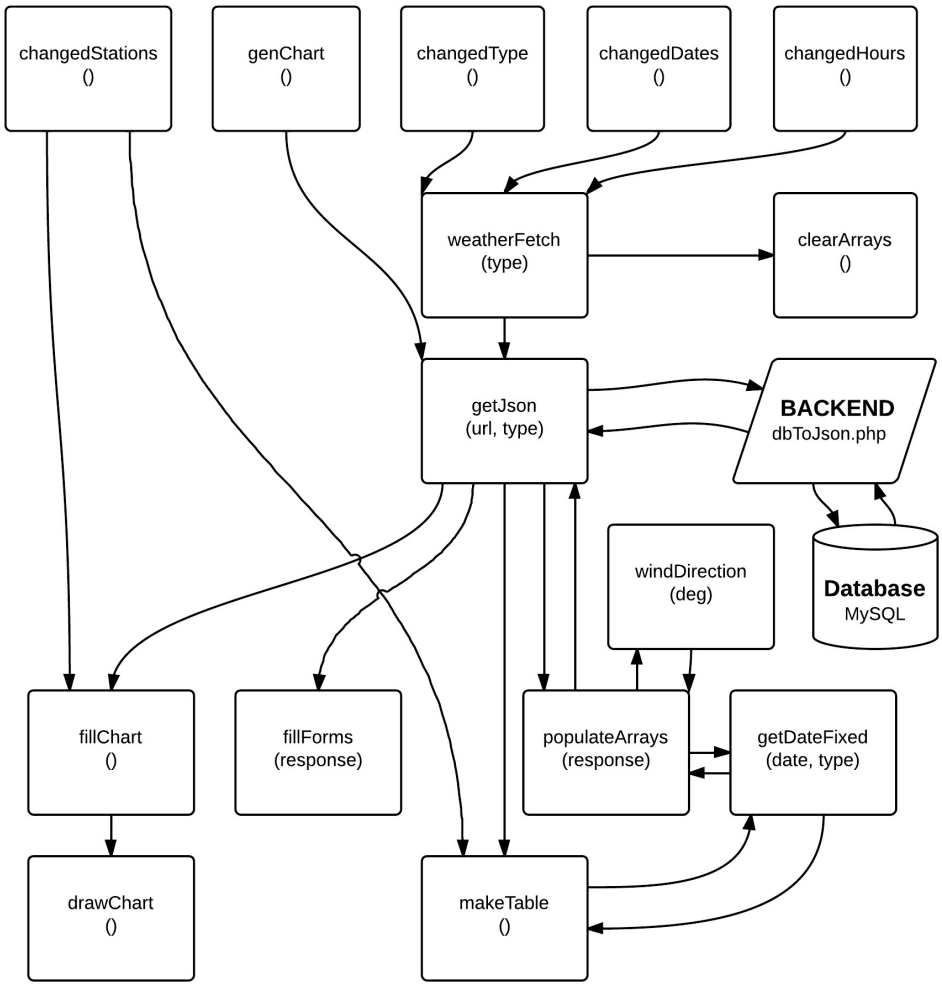
```
#!/bin/bash
#hws-make-run.sh
#Passer på at HessdalenWeatherServer kjører.

program=hessdalen16.jar

if ps aux | grep -v grep | grep $program > /dev/null
    then
        exit
else
        java -jar $program & #Programmet kjører ikke, starter på nytt.
fi
exit
```

4.3.4 Datapresentasjon

For å presentere måledata på hessdalen.org, valgte vi slik beskrevet i avsnitt 3.4.2 å lage front-end på JavaScript, og back-end på PHP. Her går vi litt i dybden på hvilke ulike teknikker og metoder vi har brukt når vi bygde denne datapresentasjonen. I figur 4.13 kan du se hvordan de ulike funksjonene i JavaScriptet kommuniserer sammen, og hvordan back-end og databasen jobber sammen med dette.



Figur 4.13: Oversikt over hvordan de ulike funksjonene i JavaScriptet kommuniserer sammen, og med back-end.

Det er mange skjermdumper av nettsiden for å beskrive dens oppbygging i dette avsnittet, men for å få den fulle opplevelsen og fortståelsen for hvordan siden fungerer, anbefales det at du besøker www.hessdalen.org og velger 'Været'('The weather' i den engelske versjonen) i menyen til venstre.

Backend

Selv om den største delen av fremvisningen ble bygd på JavaScript, så måtte vi også lage en back-end for å kommunisere med databasen. Grunnen til at dette er som beskrevet i avsnitt 3.4.2 at JavaScript er et språk som blir kompilert der og da av nettleseren. Med andre ord hadde tilkoblingsinformasjonen til databasen vært fritt tilgjengelig for alle som besøkte siden. For å omgå

dette, måtte vi bruke et språk som kompileres på serversiden, og kildekoden dermed ikke fritt tilgjengelig. Vi valgte slik også beskrevet i avsnitt 3.4.2 å lage back-end i PHP.

Det denne koden gjør er å lese av de superglobale HTTP-variablene sendt gjennom linken, hente ut relevant måledata fra databasen og skrive ut JSON-formatert data på siden. Dette blir mye enklere å forstå hvis vist i praksis, så her er et eksempel:

Åpner ≪dbToJson.php≫ med denne URL:

```
dbToJson.php?type=temperature&from=2014-5-8%2000:00:00
&to=2014-5-8%2023:59:59&hoursOverride=false
```

Her ser vi at datatype er satt til å være temperatur, fradato til 08.05.14 kl 00:00 og tildato til 08.05.14 kl 23:59.
Verdien ≪hoursOverride≫ er her satt til å være falsk, da vi velger å bruke visningens standardinstillinger for restriktering av data. Ettersom dette kun er èn dag her, er standard å vise alle målinger i tidsrommet satt. Dersom brukeren har valgt å kun vise måledata fra kl 00 og 06 ville det sett slik ut:

```
hoursOverride=true&hours00=true&hours06=true
&hours12=false&hours18=false
```

Den første linken ovenfor henter dermed ut fra databasen og konverterer til JSON slik beskrevet ovenfor. Deretter skrives det ut og blir slik:


```
[
  {
    "logId_id":"12","time":"2014-05-08 00:00:00",
    "station":"2",
    "valueAvg":"0.6212805",
    "valueNow":"0.5",
    "valueMax":"1.220001","timeMax":"2014-05-07 23:01:00",
    "valueMin":"0.5","timeMin":"2014-05-07 23:57:30"
  },
  ...
  ...
  ...
  {
    "logId_id":"35","time":"2014-05-08 23:00:00",
    "station":"2",
    "valueAvg":"1.312188333333333",
    "valueNow":"1.18",
    "valueMax":"2.100002","timeMax":"2014-05-08 22:01:00",
    "valueMin":"1.18","timeMin":"2014-05-08 22:47:30"
  }
]
```


Merk: Dette ville egentlig blitt veldig mange målinger, så de er kuttet ned til å vise første og siste for lesbarhet. Prikkene mellom målingene indikerer der de andre ville vært.

Det blir altså vist som en array av JSON-elementer. Dette blir senere tolket og brukt til presentering i JavaScript.

Back-end brukes også i første lasting av siden for å hente ut når første og siste måling ble gjort. Dette brukes for å begrense tilgjengelige datoer i kalenderene der brukeren velger tidsrom for datapresentasjon.

Biblioteker brukt

Dette er JavaScript-bibliotekene vi har tatt i bruk for å lage datapresentasjonen.

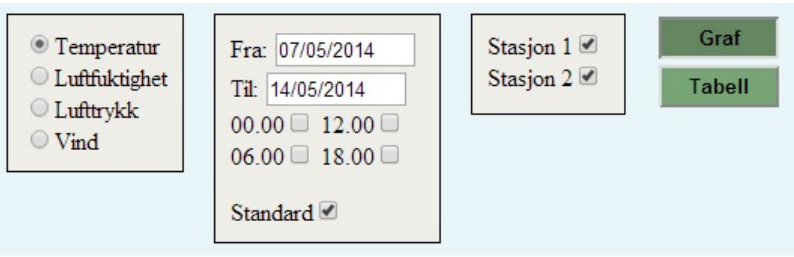
jQuery[45]
jQuery er et bibliotek laget for å simplifisere programmeringsprosessen. Mottoet deres er 'write less, do more', og gjengir ganske godt hva det går ut på. I vår kode brukes primært jQuery for å forenkle AJAX-basert datalasting (beskrevet dypere i senere punkt), men også i enkelte for-løkker og andre småfunksjoner.

jQuery UI[46]
jQuery UI er et bibliotek som samhandler med jQuery for å tilby ulike visualiseringsløsninger. I vår kode brukes dette for bygge kalenderen der bruker kan velge tidsrom for datapresentasjon.

Google Visualisation[47]
Google Visualisation er et bibliotek laget for visualisering av data gjennom grafer og diagrammer. I vår kode brukes dette for å generere linjediagrammet/linjegrafen for datapresentasjonen.

Brukervalg

I figur 4.14 vises de ulike valgene brukeren kan ta for å påvirke datapresentasjonen.



Figur 4.14: Valgene presentert for brukeren.

Disse brukervalgene har blitt valgt på bakgrunn av to kriterier:

- Hovedfokus på å ikke overvelme brukeren, og heller vise de viktigste valgene uten at det skal gå for mye på bekostning av funksjonalitet. Et eksempel på dette, er at istedenfor at brukeren kan velge hvilke type verdier som skal vises (gjennomsnitt, maks og minimum), vises kun gjennomsnittsverdien grafisk, og de andre verdiene dersom man flytter musepekeren over en måling på grafen(slik vist i figur 4.15).
- Det andre vi har tatt hensyn til er hvilke ønsker arbeidsgiver hadde. Spesifikt innebar det muligheten til å vise måledata både grafisk og på tabellform, og å kunne spesifisere hvilke timer i døgnet det skal hentes ut fra.

For å sikre at ikke brukeren skal kunne gjøre noen feil når de velger, blir alle feltene sjekket hver gang det blir gjort en endring. Det vil si at brukeren kan for eksempel ikke huke bort alle timesvalgene uten at ≪Standard≫ blir automatisk huket på igjen. Lignende sikring er laget for stasjonsvalget. I kalenderen som åpnes når tekstfeltene markert ≪Fra≫ og ≪Til≫ trykkes, er det ikke mulig å verken velge datoer der det ikke er målinger (Altså før stasjone ble montert, og etter nåtid). Det er heller ikke mulig å sette en senere fradato enn den valgte tildatoen. Alle disse sikringene kombinert, gjør at det er tilnærmet umulig å utføre brukerfeil.

Dynamisk innholdslasting

Siden er bygd opp slik at brukeren aldri trenger å trykke noen dedikert knapp for å laste data. Så fort et brukervalg blir endret, oppdaterer datapresentasjonen seg selv for å reflektere de nye valgene. Dette sørger for en veldig dynamisk følelse på hele siden, og forbedrer brukeropplevelsen.

Måten dette har blitt oppnådd på, er ved å bruke en teknologi kalt AJAX. Dette gjør det altså mulig for nettleseren å hente ny data fra back-end uten at selve siden må lastes på nytt. Det er mulig å lage AJAX-basert datalasting i JavaScript uten noen ekstra biblioteker, men vi har valgt å bruke jQuery for å forenkle prosessen. Dette kombinert med å ha funksjonskallene til JavaScriptet når brukeren endrer en av brukervalgene, sørger for denne dynamiske opplevelsen.

Grafisk fremvisning

For å ha muligheten til å vise frem måledata på graf-/diagramform, bestemte vi oss altså for å bruke biblioteket 'Google Visualisation'. I dette biblioteket var det mange ulike måter å presentere data på, men vi bestemte oss for å bruke linjedigram da dette er den meste konvensjonelle og oversiktlige måten å presentere værdata på. Slik beskrevet i punktet om 'Brukervalg' ovenfor, står brukeren ganske fritt til å velge hvilke målinger som skal vises i diagrammet. X-aksen på diagrammet viser tid, og Y-aksen viser målingsverdi. Google Visualisation sørger for at akseverdiene justerer seg til de mest praktiske verdiene. Med det menes det at dersom man for eksempel kun har temperaturmålinger mellom 20 og 25 grader i tidsrommet valgt, vil verdiene på Y-aksene justere seg for å best reflektere dette. Dersom brukeren velger et stort tidsrom, reduseres antall målinger vist om dagen slik:

1-10 dager

- Standard: Alle målinger vises(24 i døgnet).
- Brukervalg: Kan vise alle fire (kl 00, 06, 12 og 18).

11-30 dager

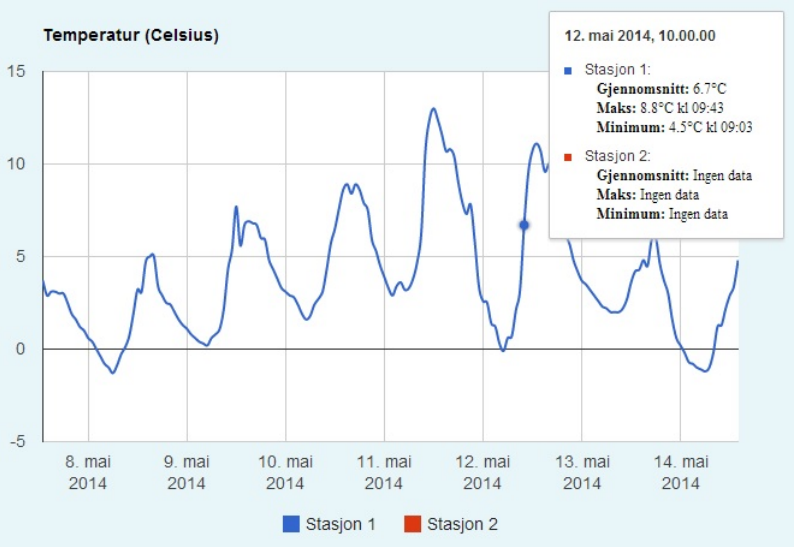
- Standard: To målinger i per døgn vises (kl 06 og 12).
- Brukervalg: Kan vise alle fire (kl 00, 06, 12 og 18).

Over 30 dager

- Standard: En måling per døgn (kl 12).
- Brukervalg: Kan vise èn av fire (kl 00, 06, 12 og 18).

Hovedgrunnene til at intervallet mellom målingene må begrenses i grafisk visning er at lesbarheten vil gå ned, og datakraften nødvendig vil gå opp ved for mange punkter vist.

På figur 4.15 kan du se hvordan det ser ut når alle temperaturmålinger mellom 8. mai og 14. mai vises.



Figur 4.15: Måledata for temperatur presentert med linjediagram.

Tabellvisning

Tabellvisningen (vist i figur 4.16) er bygd dynamisk av JavaScriptet på vanlige HTML-tabeller basert på hvilken måledata som skal vises. Denne tabellen erstatter diagrammets plass på siden når brukeren velger å vise måledata på tabellform.

Å vise data på denne måten, begrenser ikke antall målinger hentet ut fra databasen dersom ≪Standard≫-verdien er satt, uansett tidsrom. Vi har valgt å gjøre det slik, da det å vise en tabell ikke krever nevneverdig mer av maskinen eller reduserer lesbarheten, samme hvor mange målinger som vises. Du kan forøvrig fremdeles velge å begrense hvilke timer måledata skal vises fra selv.

Temperatur (Celsius)

Dato	Kl	Stasjon 1 Gjennomsnitt	Stasjon 2 Gjennomsnitt	Stasjon 1 Maks	Stasjon 2 Maks	Stasjon 1 Min	Stasjon 2 Min
14/05/2014	16:00	4.7°C	Ingen data	5.1°C kl 15:54	Ingen data	4.2°C kl 15:07	Ingen data
14/05/2014	15:00	4.8°C	Ingen data	5.4°C kl 14:01	Ingen data	4.3°C kl 15:00	Ingen data
14/05/2014	14:00	4.8°C	Ingen data	5.7°C kl 13:19	Ingen data	4°C kl 13:04	Ingen data
14/05/2014	13:00	3.4°C	Ingen data	4°C kl 12:01	Ingen data	2.8°C kl 12:11	Ingen data
14/05/2014	12:00	2.9°C	Ingen data	4.3°C kl 11:55	Ingen data	2.2°C kl 11:46	Ingen data
14/05/2014	11:00	2.2°C	Ingen data	4.2°C kl 10:56	Ingen data	1.2°C kl 10:09	Ingen data
14/05/2014	10:00	1.3°C	Ingen data	1.8°C kl 09:01	Ingen data	1.1°C kl 09:28	Ingen data

Figur 4.16: Måledata for temperatur presentert på tabellform.

Praktisk informasjon

For å gi brukeren litt praktisk informasjon om værstasjonene, har vi laget en egen boks (se figur 4.17) på siden dedikert til nettopp dette. Her er det beskrevet når stasjonene ble montert, hvor de står og litt informasjon om hvordan den grafiske fremvisningen begrenses av størrelsen på tidsrommet.

Figur 4.17: Praktisk informasjon om værstasjonene vist på siden.

Feilhåndering

Her var målet å skrive koden med så god feilhåndtering at ingenting skal stoppe opp selv om det mangler hele målinger eller er uhåndterbare verdier i databasen. I tillegg til dette har fokuset vært å gjøre opplevelsen av presentasjonsdataen god, selv om noe slikt skulle oppstå. Ettersom stasjon 2 ikke er i stand til å sende værdata før den får nettverk igjen (se avsnitt 4.2.3), har vi et godt eksempel på dette i figur 4.15.

Dersom JavaScriptet skulle få problemer med å hente ut data produsert av back-end pga. av treg databasetilkobling, eller lignende, får brukeren beskjed om det. Denne beskjeden lyder som følger ≪Tidsavbrudd. Fortsetter å prøve...≫. Så fort tilkoblingen er gjenopprettet, vises data slik

brukeren ba om.

Design

Hovedfokuset her har vært å lage en side som er oversiktlig og lett å bruke. Slik beskrevet i punktet om 'Brukervalg' ovenfor, skal det heller ikke være nødvendig å måtte bruke tid på å forstå hvordan siden skal brukes.

Videre har vi valgt et design som skal være enkelt, men appellerende for brukeren. Slik vist på figur 4.18, er siden delt inn i tre områder:

Brukervalg

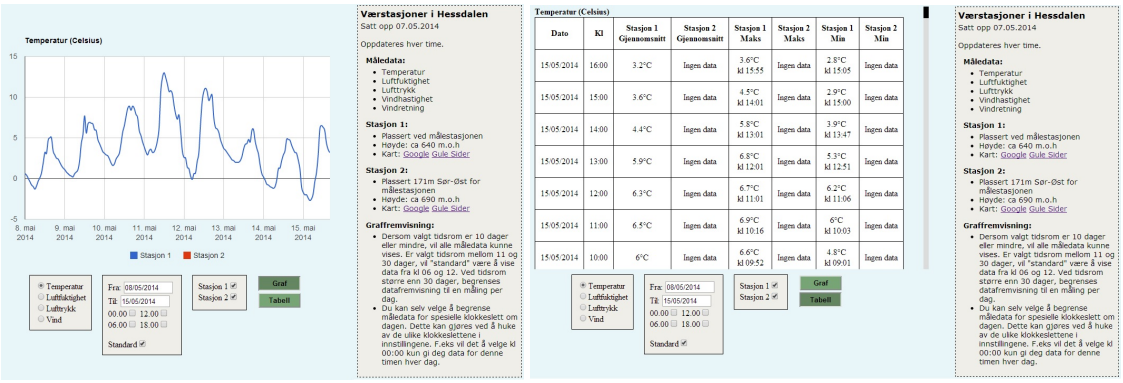
- Plassering: Nederste på siden.
- Formål: La brukeren påvirke hvilke data som skal vises.

Datafremvisning

- Plassering: Til venstre på siden.
- Formål: Fremvise måledata enten via diagram eller tabell.

Informasjon

- Plassering: Til høyre på siden.
- Formål: Vise praktisk informasjon i sammenheng med værstasjonene.



(a) Med linjediagram (b) Med tabell

Figur 4.18: Det ferdige designet på siden med de ulike metodene for å vise data.

For å få til dette i praksis, har vi brukt CSS3, som er en stilarkteknologi. Hele designet er bygd ved hjelp av denne teknologien.

Engelsk versjon

Oppdragsgivers nettside www.hessdalen.org er tilgjengelig på både Norsk og Engelsk. Dette innbar at vi måtte lage en engelsk versjon av datafremvisningen også. For å lage denne, ble vi nødt til å oversette flere variabler og strenger i JavaScriptet, og teksten skrevet i HTML-filen.

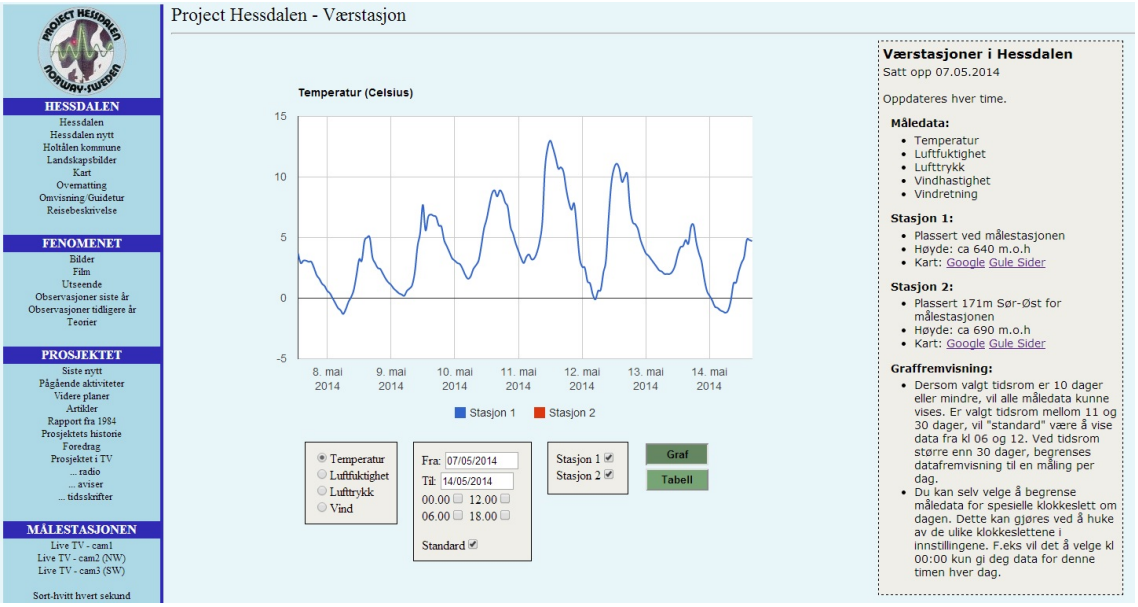
Alle figurene i dette avsnittet er hentet fra den Norske versjonen av siden, men den Engelske versjonen ligger også fritt tilgjengelig på www.hessdalen.org.

Sikkerhet

Det eneste sikkertsaspektet vi har måtte ta hensyn til i denne koden, er såkalt 'SQL-injection'. Det vil si at noen kan prøve å bruke back-end til å sende inn ondsinnede spørringer til databasen. Som for eksempel å avslutte en streng (semikolon) sendt til databasen og følge opp med en 'DROP TABLE...'-setning. Dette har vi sikret oss mot ved å ta i bruk funksjonen 'mysql_real_escape_string'[48] i PHP. Denne sørger for at det blir lagt på en '\' foran tegn MySQL kan tolke. Slik blir de gjort ubrukelige, og 'SQL-injection' umulig.

Implementasjon på hessdalen.org

Da vi var ferdig med produksjonen av datafremvisningen, skulle den implementeres på www.hessdalen.org. Dette ble gjort i tett samarbeid med oppdragsgiver, og det eneste utfordringen vi støtte på var en CSS-konflikt der siden vår overstyrte stilsettet på www.hessdalen.org sine tabeller. Dette ble raskt løst ved å sette spesifikke ID'er på tabellene i datapresentasjonen.



Figur 4.19: Den norske versjon av datapresentasjon implementert på hessdalen.org

Samarbeid med oppdragsgiver

Ved flere anledninger har vi latt oppdragsgiver prøve ut prototyper av datapresentasjonen. Dette for å kunne få tilbakemelding på hva som var bra, og hva som kunne vært gjort annerledes. På grunn av et slikt samarbeid, har vi kunne lage et mye bedre produkt, som oppdragsgiver er veldig fornøyd med(se avsnitt 6.1.2).

Kapittel 5

Testing

I dette kapittelet vil vi ta for oss hvordan vi har testet produktet. Forklare hvordan de ulike delene av produktet har blitt testet. Forklare kort om hva vi var ute etter med testen og om produktet holdt mål.

5.1 Testing av Programvare

5.1.1 Testing av mikrokontroller med sensorer

Hver sensor ble testet individuelt med mikrokontrolleren til vi fikk kommunisert med de. Etter alle sensorene var fungerende begynte vi å teste nettverksfunksjonalitet. Dette gjorde vi ved å sette opp en midlertidig server som kun tok imot data og printet de ut på skjerm. Da fikk vi raskt oversikt over hva som var feil.

En feil som dukket opp var at nettverksmodulen på mikrokontrolleren sluttet å fungere etter 2-3 sendinger. Dette løste vi ved å starte om mikrokontrolleren hver hele time, noe som løste feilen.

Vi testet også hva som skjedde hvis en sensor sluttet å fungere. Dette simulerte vi ved å plugge sensoren fra mikrokontrolleren. Vi implementerte løsninger som håndterer dette hvis det skulle skje med det ferdige produktet.

5.1.2 Testing av serveren

Hovedfokuset under testingen av serveren, har vært å se at riktig måledata blir plassert riktig i databasens tabeller. Dette har vi gjort ved å sammenligne mottatt måledata med det som blir lagt inn i databasen.

En annen svært viktig del, har vært å se hvordan programmet håndterer ulike feil. Med dette mener vi tidsavbrudd, korrupte pakker, feilformatert JSON, o.l. Ved alle disse feilene, har målet vært at programmet ikke krasjer, men heller går tilbake til en tilstand der det er klart for nytt datamottak.

Det har også blitt testet for å se at den trådbaserte delen av programmet fungerer. Det vil si at vi har prøvd å sende flere tilkoblinger til serveren helt på likt.

Alle disse testene var vellykket, og ingen feil ble funnet.

5.1.3 Testing av datapresentasjonen

I testingen av datapresentasjonen har fokuset vært todelt.

Den ene delen har gått på å sjekke at riktig data blir hentet ut fra databasen og presentert på korrekt sted. Dette har vi gjort ved å sammenligne databasens verdier med det som blir presentert på nettsiden.

Den andre delen har gått ut på å se om alle brukervalg blir håndert riktig i presentasjonen, og gir tilbake forventet resultat. Koden er skrevet på en slik måte at det ikke skal være mulig å gjøre brukerfeil på siden (slik beskrevet i avsnitt 4.3.4). Dette har også blitt testet ved å se om vi fant noen vei vi kunne lure presentasjonen til å vise noe vi ikke ønsket.

Alle disse testene var vellykket, og ingen feil ble funnet.

5.2 Testing av kabinett

Da vi skulle kvalitetsteste kabinettet var vi ute etter å finne ut om det det holder på temperaturen når det blir kaldt ute. Det var også interessant å finne ut om termostaten fungerer og faktisk slår på varmeelementet når temperaturen synker under $5\,^{\circ}\mathrm{C}$.

Testen ble utført ved å legge kabinettet i en dypfryser som vi forhåndsmålte til å holde rundt $-25\,^{\circ}\mathrm{C}$. For å kunne sjekke temperaturen i kabinettet la vi inn et termometer.

Etter tre timer sjekket vi tilstanden inne i kabinettet. Som forventet har termostaten reagert på temperaturfallet og skrudd på varmeelementet. Nå var temperaturen inne i kabinettet på $2\,^{\circ}\mathrm{C}$ som også tyder på at isolasjonen vi satt inn hjelper en god del.

Testen var vellykket og viser at de tiltakene vi har tatt for å forhindre alt for lav temperatur fungerer.

5.3 Testing av systemet

Før systemet skulle settes i drift, langtidstestet vi det ved å la alt kjøre i to fulle døgn. Målet her var å finne feil som kunne oppstå under lengre kjøringer.

Måten dette ble utført på er at vi koblet systemet opp mot en datamaskin som lagret de feilmeldingene serveren og mikrokontrolleren sender ut dersom en av de skulle feile.

Denne testingen var svært vellykket. Vi mistet ingen målinger i løpet av disse to døgnene.

Kapittel 6

Diskusjon

Her kommer vil til å ta opp hva vi har lært av prosjektet. Se om vi klarte å nå målene fra avsnitt 1.4.1, leverte vi det forventede resultatet i avsnitt 1.4.2 og fungerte metoden i avsnitt 1.4.3. Dertetter diskutere hva vi er fornøyde med, hva som kunne vært bedre og forklare hvorfor det ble bra eller dårlig. Ta opp eventuelle problemer vi har støtt på og tilslutt si noen ord om hva vi ville ha gjort anderledes om vi skulle ha gjort dette på nytt.

6.1 Erfaringer vi har gjort oss

Gruppen har tilegnet seg mye kunnskap. Vi har fått jobbet med utvikling av maskinvare, programvare og prøvd oss i feltarbeid. Prosessen har ført til at vi nå sitter med ny kunnskap om hvordan prosjekter bør planlegges og utføres.

6.1.1 Måloppnåelse

Dette er målene fra kapittel 1.

Hovedmål Lage en komplett værstasjon med datapresentasjon på hjemmesiden til Project Hessdalen.

- **Delmål 1** Lage en værstasjon som skal måle temperatur, lufttrykk, luftfuktighet, vindretning og vindhastighet.
- **Delmål 2** Lagre værdata på server til Høgskolen i Østfold.
- **Delmål 3** Presentere værdata på hjemmesiden til Project Hessdalen.
- **Delmål 4** Tilrettelegge for å kunne sammenligne værdata fra ulike tidsperioder.
- **Delmål 5** Lage et kabinett til mikrokontrolleren, koblingsbrettet og strømforsyningene.
- **Delmål 6** Tilpassning av hardware. Med tanke på lavpassfilter, spenningsdelere.

Vi nådde alle målene vi satt oss i planleggningsfasen.

6.1.2 Opp til forventningene?

Resultatet vi leverte levde opp til både våre egne og oppdragsgivers satte mål. Under en samtale vi hadde med han spurte vi om han er fornyød med prosjektet. Han sa seg meget godt fornyød med arbeidet vi hadde gjort, spesielt med tanke på de spesifikasjoner som ble gitt oss i forkant av prosjektet.

6.1.3 Arbeidsmetoden

Den induktive metoden fungerte. Siden vi har begrenset erfaring innefor værstasjoner og hvordan disse skal lages, ble det til at vi måtte lære oss mye underveis. Hvordan sensorene kommuniserer med mikrokontrolleren var en av tingene vi måtte tilegne oss kunnskaper om på grunn av manglene erfaring på området. Derav har den indutive metoden fungert bra.

6.2 Fornøyd, misfornøyd?

6.2.1 Hva vi er fornøyde med

Når vi ser på systemet i sin helhet er vi meget fornøyd med det vi har fått til. Værstasjonene er montert og har vært i drift i snart to uker. I denne perioden har det ikke vært noen problemer med hverken avlesninger, sendinger eller visualisering av data.

Vi er meget fornøyd hvor variert oppgaven har vært. Den har tatt for seg hele utviklingsprosessen. Alt fra å kjøpe inne materiale og utstyr, programmering og kobling til å dra ut å montere. Så vi sitter igjen med en del ny kunnskap og erfaring.

Både vi og oppdragsgiver er fornøyd med hvordan kommunikasjonen oss i mellom har vært. Vi har hatt faste møter med arbeidsgiver hvor vi har oppdatert han på hvordan vi ligger ann og han har hatt mulgihet til å komme med innspill på ting som burde gjøres anderledes.

6.2.2 Hva vi er misfornøyde med

Som sagt er vi fornøyd med prosjektet i sin helhet, men en ting som kunne vært løst på en annen måte er hvordan ledningene fra sensorene er koblet til koblingsbrettet. Slik vi endte opp å gjøre det var at vi loddet ledningene fast til koblingsbrettet. I ettertid ser vi at dette er en tungvidt løsning dersom det skulle bli nødvendig å bytte en av sensorene. Det vi heller burde ha gjort er og tilpasset ledningene slik at de enkelt kan kobles av og på koblingsbrettet.

Kapittel 7

Konklusjon

Vi er fornøyd med måten vi har løst oppgaven og produktet vi har laget. Værstasjonene er montert og i drift. Dataene kan du se på hjemmesiden til Project Hessdalen. Vi har oppfylt de kravene som oppdragsgiver har stilt. Oppdragsgiver har selv sagt seg meget fornøyd med de resultatene av prosjektet han har fått sett. Dette er innredning i kabinettet til værstasjonen, festeanordningene til sensorene og kabinettet, fysisk beskyttelse av komponentene og visualiseringen av værdataene. Vi tør påstå at oppdragsgiver syns prosjektet og resultatene er bedre enn forventet.

Hvis prosjektet skulle bli videreført, har vi noen tips til utbedringer:

- Måten ledningene fra sensorene er koblet til koblingsbrettet bør gjøres mindre permanent, slik at det blir enklere å bytte ut sensorer som ikke lenger fungerer.
- Lage en løsning som sjekker vindhastigheten oftere slik at det er mulig å registrere vindkast og lagre disse til databasen.
- Legge til mer funksjonalitet rundt visualisering av data. For eksempel statestikk på varmeste dag, kaldeste dag osv.

Gruppen er tilfreds med oppgaven fordi det har vært utfordringer innenfor maskinvare og programvare. Det at den ga oss mulighet for å være med ut i felt for installasjon av værstasjonene var også en lærerik opplevelse.

Bibliografi

[1] E. Strand, "Project Hessdalen." http://www.hessdalen.org/index_n.shtml, 2000.

[2] S. Brown, "Building A Weather Station." http://www.drbunsen.org/building-a-weather-station/, 2013.

[3] D. I. Corp, "Davis 6250 Vantage Vue." http://www.vantagevue.com/products/product.asp?pnum=06250, 2014.

[4] T. R. Pi, "Raspberry Pi." http://www.raspberrypi.org/, 2014.

[5] E. T. Co., "EDIMAX EW-7811Un." http://www.edimax.com/en/produce_detail.php?pd_id=347&pl1_id=1, 2014.

[6] bram2202, "Arduino weather station." http://www.instructables.com/id/Arduino-weather-station/?ALLSTEPS, 2013.

[7] M. i. NRK, "Yr.no Hessdalen 15.03.13." http://www.yr.no/sted/Norge/S%C3%B8r-Tr%C3%B8ndelag/Holt%C3%A5len/Hessdalen/statistikk.html, 2013.

[8] P. Vibe, "Viten om vær og vind." http://historienet.no/naturvitenskap/viten-om-vind-og-vaer, 2012.

[9] P. Vibe, "Viten om vær og vind." http://historienet.no/palle-vibe/1643-1738-maling-av-lufttrykk-0, 2012.

[10] P. Vibe, "Viten om vær og vind." http://historienet.no/palle-vibe/1783-ballongferd-viste-vaergudenes-luner-0, 2012.

[11] P. Vibe, "Viten om vær og vind." http://historienet.no/palle-vibe/1806-1849-utveksling-av-vaerdata-0, 2012.

[12] P. Vibe, "Viten om vær og vind." http://historienet.no/node/49705, 2012.

[13] P. Vibe, "Viten om vær og vind." http://historienet.no/palle-vibe/1927-1955-radiosonden-var-et-stort-fremskritt-0, 2012.

[14] P. Vibe, "Viten om vær og vind." http://historienet.no/palle-vibe/fra-1950-arene-vaerfolk-og-datamaskiner-gikk-hand-i-hand-0, 2012.

[15] NASA, "Vanguard 2." http://nssdc.gsfc.nasa.gov/nmc/spacecraftDisplay.do?id=1959-001A, 2014.

[16] NASA, "TIROS 1." http://science1.nasa.gov/missions/tiros/, 2014.

[17] K. E. Harstveit, "lufttrykk." http://snl.no/lufttrykk, 2007.

[18] K. E. Harstveit, "vind." http://snl.no/vind, 2011.

[19] Ukjent, "vindfløy." http://snl.no/vindfl%C3%B8y, 2012.

[20] Ukjent, "vindpølse." http://www.rantex.no/?PageID=143&ItemID=169, 2011.

[21] E. Project, "Ethernut Hardware Manual." http://ethernut.de/pdf/ethernut21b_ehwm_2_0.pdf, 2008.

[22] Ukjent, "Temperaturmåler." http://www.adafruit.com/products/1298, 2014.

[23] SENSIRION, "Datasheet SHT1x." http://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/Humidity/Sensirion_Humidity_SHT1x_Datasheet_V5.pdf, 2011.

[24] Ukjent, "Trykkmåler." http://www.adafruit.com/products/1603, 2014.

[25] BOSCH, "Datasheet BMP180." http://www.adafruit.com/datasheets/BST-BMP180-DS000-09.pdf, 2013.

[26] Ukjent, "Måler for vindhastighet." http://www.fuehlersysteme.de/wind-speed-transmitter-compact.html, 2014.

[27] Ukjent, "Måler for vindretning." http://www.fuehlersysteme.de/wind-direction-transmitter-compact.html?___store=de_en&___from_store=de_en, 2014.

[28] B. Profab, "IP ratings explained?." http://www.bisonprofab.com/ip-ratings-explained.htm, 2014.

[29] solid IT, "DB-engines Ranking." http://db-engines.com/en/ranking, 2014.

[30] P. S. Foundation, "Python." https://www.python.org/, 2014.

[31] P. S. Foundation, "Common Gateway Interface support." https://docs.python.org/2/library/cgi.html, 2014.

[32] K. Seidler, "Good Idea: Python with FastCGI." https://blogs.oracle.com/oswald/entry/good_idea_python_with_fastcgi, 2010.

[33] M. D. Network, "JavaScript." https://developer.mozilla.org/en-US/docs/Web/JavaScript, 2014.

[34] T. P. Group, "PHP: Hypertext Preprocessor." http://www.php.net/, 2014.

[35] A. Munroe, "PHP: a fractal of bad design." http://me.veekun.com/blog/2012/04/09/php-a-fractal-of-bad-design/, 2012.

[36] Oracle, "Java.com." http://www.java.com/en/, 2014.

[37] P. Krill, "Java tops C as most popular language in developer index." http://www.infoworld.com/t/java-programming/java-tops-c-most-popular-language-in-developer-index-224781, 2013.

[38] E. Project, "Ethernut Hjemmeside." http://www.ethernut.de/index.html, 2014.

[39] I. Free Software Foundation, "GCC, the GNU Compiler Collection." https://gcc.gnu.org/, 2014.

[40] Atmel, "Atmel128 Datablad." http://www.atmel.com/Images/doc2467.pdf, 2011.

[41] E. S. Academy, "I2C FAQ." http://www.esacademy.com/en/library/technical-articles-and-documents/miscellaneous/i2c-bus/frequently-asked-questions/i2c-faq.html, 2014.

[42] FuehlerSysteme, "Datasheet WG2/O-10." http://www.fuehlersysteme.de/wind-speed-transmitter-compact.html, 2003.

[43] FuehlerSysteme, "Datasheet WRG2/O-10." http://www.fuehlersysteme.de/wind-direction-transmitter-compact.html?___store=de_en&___from_store=de_en, 2003.

[44] P. Vixie, "Man Page for crontab." http://www.unix.com/man-page/linux/5/crontab/, 2010.

[45] T. jQuery Project, "jQuery." http://jquery.com/, 2014.

[46] T. jQuery Project, "jQuery UI." https://jqueryui.com/, 2014.

[47] Google, "Google Chart API." https://developers.google.com/chart/, 2014.

[48] T. P. Group, "PHP: mysql_real_escape_string." http://www.php.net/mysql_real_escape_string/, 2014.