NTNU
Kunnskap for en bedre verden

Analyse av metoder for å estimere kompresjonsdybde under hjerte-lunge-redning med smartklokke

Kristofer Tvedt Henrichsen

Master i kybernetikk og robotikk
Innlevert: juni 2018
Hovedveileder: Tor Engebret Onshus, ITK

Norges teknisk-naturvitenskapelige universitet
Institutt for teknisk kybernetikk

Oppgavetekst

Hjerte og lungeredning redder liv. Utfordringen i dag er at mange ikke tør å hjelpe til med HLR fordi de ikke vet hvordan det gjøres eller er usikre på rytme, kompresjonsdybde og rekkefølge på innblåsinger. En tidligere prosjektoppgave innenfor dette tema undersøkte mulighetene for å estimere rytmen man utfører HLR med basert på innebygde sensorer i en smartklokke. Målet for denne masteroppgaven er å benytte sensorene i smartklokken til å kunne estimere dybden man utfører brystkompresjoner med. Flere teknikker for dette skal utforskes, men modellen for dybdeestimering burde være robust og fungere uavhengig av bruker og rotasjon av smartklokken.

Forord

Denne masteroppgaven bygger videre på en prosjektoppgave om samme tema som ble skrevet høsten 2017 (1). Temaet for begge oppgavene er å undersøke mulighetene for en assistent til hjerte-lunge redning på en smartklokke. I prosjektoppgaven var fokus på å gjenkjenne rytmen man utfører HLR med, mens i denne oppgaven blir det forsøkt å estimere dybden til kompresjonene. En del av implementasjonen i denne oppgaven bygger på de samme fundamentene som i prosjektoppgaven, men ble skrevet på ny for det nye formålet i masteroppgaven. Dette blir nærmere diskutert i kapittel 1. Masteroppgaven er skrevet i et samarbeid med Laerdal medical, som står som oppgavestillere. Utenom to møter under prosjektoppgaven, hadde vi ett møte i januar før oppstart på masteroppgaven hvor problemstillingene for oppgaven ble diskutert. I tilegg hadde jeg 5 korte møter med veileder, hvor fremdrift på oppgaven ble gjennomgått og veileder ga råd for hvordan videre arbeid burde struktureres.

Laerdal har utlånt en gjenopplivingsdukke, og instituttet for Teknisk Kybernetikk ved NTNU har lånt ut en stasjonær PC. Resten av utstyret har undertegnede stått for selv. Programvaren brukt er åpent tilgjengelig for NTNU's studenter, og mesteparten er åpen-kildekode. Sett bort ifra eksterne programvare-biblioteker ble all kode, analyse og implementasjon utført av undertegnede.

Masteroppgaven kombinerer mye forskjellig teknologi som viste seg å være tidkrevende og utfordrende å implementere. Likevel, har det vært en utrolig interessant oppgave. Jeg vil gjerne takke Laerdal for å ha gitt meg denne spennende oppgaven, og takke veilederen min Tor Onshus. I tilegg vil jeg takke mine medstudenter for diskusjon og refleksjon for de uttalige problemstillingene som har oppstått i oppgavens løp.

Figur 1: Signert Kristofer Henrichsen

Oppsummering og konklusjon

Denne oppgaven har forsøkt å estimere kompresjonsdybden under Hjerte- Lunge-redning basert på sensormålingene fra en smartklokke. Presis estimering av kompresjonsdybde er viktig for å kunne a lage en assistent til HLR som kan veilede brukeren i sanntid. Formålet med en HLR-veileder er å sørge for at HLR blir utført så korrekt som mulig på den skadede for å maksimere sjansen for å redde den skadedes liv. For å få til dette ble det implementert en dybdemåler i en gjenopplivingsdukke som logget kompresjonsdybde i sanntid, som kommuniserte med en android-smartklokke. Data logget fra disse to enhetene ble behandlet og sammenlignet. Målet var å estimere referanse-dybden fra gjenopplivingsdukken basert på smartklokkens målinger. Sensormålingene fra smartklokken ble brukt for å lage et estimat for kompresjonsdybde basert på dobbeltintegrasjon av akselerasjon. Videre ble dette dybde-estimatet og sensormålingene analysert med PCA, en metode innen Multivariate Data-Analysis. Informasjonen fra denne analysen ble brukt for å lage en *Partial Least Square Regression*-modell for å kunne øke nøyaktigheten til dybde-estimatene.

Hjertekompresjoner ligger optimalt rundt 40-60mm, og en presisjon på under 3mm ansees som å være ønskelig for å kunne klassifisere kompresjoner som ”for grunne”, ”passe”, eller ”for dype”. Både PLSR-modellen og den tradisjonelle metoden ble testet på ny og ukjent data. På det nye testsettet estimerte PLSR-modellen kompresjonsdybde med et gjennomsnittlig avvik på 2.6mm. Det beste estimatet uten bruk av multivariat modell ga et gjennomsnittlig avvik på 3.4mm. Begge disse estimeringsteknikkene fremstår som robuste og burde være representative for fremtidige scenarioer. Det fremstod derfor av arbeidet i denne oppgaven som det er fullt mulig å implementere en HLR-assistent på smartklokke.

Abstract

The goal of this master thesis was to make an assistant for cardiopulmonary resuscitation (CPR). The motivation behind this was to encourage bystanders and help them perform better and more accurate CPR. The CPR-assistant was meant to be implemented on an android smartwatch, which is supposed to give live feedback of the rhythm and the compression depth of CPR while it is performed. Both the rhythm and the compression-depth therefore needs to be calculated by only using the internal sensors of the smartwatch. An earlier project by the writer of this thesis illustrated solutions for rhythm-estimation, so the focus in this thesis was to estimate the depth ofcompressions.

To estimate the compression-depth both the accelerometer and the gyroscope inside an android smartwatch were used. A rig was made inside a resuscitation manikin to measure the distance the breast was compressed. The depth measured in the manikin was used as the reference depth the smartwatch was supposed to estimate based on its own sensors. Traditional methods based on integrating acceleration two times to get the position traveled and multivariate data analysis were used to estimate the compression depth. The methods were created and trained on a varied dataset consisting of CPR performed on the manikin by several different people.

Later, the performance of the depth-estimation algorithms were tested on a completely new dataset. Usually compressions are around 30-60mm. The goal was to have a mean absolute deviation (MAD) under 3mm in order to measure compression-depth in a reliable and meaningful manner. With the new test-set, a Partial Least Squares Regression model (PLSR) performed best and had MAD of 2.6mm. The best result from a traditional model achieved a MAD of 3.4mm. The conclusion of this master thesis is that it is entirely possible to use a smartwatch as a CPR-assistant.

Innhold

KAPITTEL 1

Introduksjon

Hjerte og lungeredning redder liv. For å maksimere sjansen for å overleve hjertestans utenfor et sykehus, kreves følgende:

- *At HLR blir utført*
- At HLR blir utført med riktig frekvens
- *Dybden* på brystkompresjonene er riktig dybde
- At man utfører HLR i et mønster med 30 kompresjoner etterfulgt av 2 innblåsinger

Det første punktet på denne listen er et fundamentalt. Det viktigste ved en plutselig hjertestans er at HLR faktisk blir utført av forbipasserende. Norges befolkning er helt i verdenstoppen til å utføre HLR, hvor i situasjoner med hjertestans utenfor sykehus har HLR blitt utført på 83% av pasientene (2). I mange steder av verden er ikke disse tallene i nærheten like positive. Noen vegrer seg gjerne fra å utføre HLR, fordi de er usikre på hvordan det blir utført eller så har de aldri hatt skikkelig opplæring. Noe som kunne vært til hjelp er en sikker og robust assistent som man alltid kan ha med seg. En assistent man kan ha på seg hele tiden, og skulle man være vitne til en hjertestans kunne denne assistenten veileidet brukeren igjennom situasjonen frem til ambulansepersonell ankommer. Muligheten for å ha med seg en slik assistent på en smartklokke er tema for denne masteroppgaven og prosjektoppgaven om samme tema fra høsten 2017 (1). Smartklokker har innebygde sensorer som forhåpentligvis gjør dem i stand til å estimere rytmen og dybden man utfører hjertekompresjoner med. Med denne teknologien implementert på en klokke man alltid har med seg, kan brukere av klokken få sanntid tilbakemelding på at hjerte og lunge-redning de utøver er korrekt. En slik HLR-assistent kan både bidra til at flere

faktisk tør å utføre HLR, samt at HLR blir utført bedre. Den kan også bidra til å gjøre HLR-undervisning bedre i tilfeller hvor man begrenset utstyr.

1.1 Tidligere arbeid

Denne masteroppgaven er en fortsettelse av en prosjektoppgave utført av undertegnede om det samme temaet høsten 2017. Prosjektoppgaven gikk ut på å gjenkjenne rytmen HLR utføres med, basert på det interne akselerometeret i en Androidbasert smartklokke. En veileder-applikasjon til smartklokke ble laget som ga tilbakemelding av HLR-rytme til brukeren i sanntid, og fremstilte grafisk om man skulle utføre HLR raskere eller tregere. Dette viste seg til å vær tilfredstillende nøyaktig, og så ut til å fungere uavhengig av rotasjon på klokken. Arbeidet i prosjektoppgaven var ment som en introduksjon og mulighetsanalyse for å gi undertegnede en grunnleggende forståelse for problemet som utforskes i denne masteroppgaven.

Denne masteroppgaven går videre til neste punkt på listen, nemlig å estimere dybden til brystkompresjoner. Til dette blir både smartklokkens akselerometer og gyroskop benyttet. Å få på plass solid og nøyaktig dybde-estimering er siste viktige steg mot målet om en HLR-assistent. Med bra estimater for både rytme og dybde kan teknologiene kombineres for å lage en fullverdig assistent med grafisk brukergrensesnitt.

1.2 Mål og avgrensning

Et av premissene for HLR-assistenten som designes i denne oppgaven er at den skal være lett tilgjengelig, og det var derfor hensiktsmessig å teste denne teknologien på en platform som er lett å ha med seg rundt i hverdagen. Ettersom de færreste vil gå med proprietære armbånd var det bedre å benytte en teknologi folk gjerne allerede har og som blir stadig mer utbredt, altså smartklokker. Salget av smartklokker har hatt tilnærmet eksponentiell vekst de siste årene, og i 2018 forventes det at det skal selges 141 millioner smartklokker globalt (3).

I denne oppgaven har ikke implementasjon på smartklokke fokus, og mesteparten av analysen foregår på PC. Oppgaven skal prøve å belyse *hvor* nøyaktig man kan estimere kompresjonsdybde basert på en smartklokkes interne sensorer. Smartklokken brukes derfor til å logge sensordata, men resultatene av dette blir ikke vist frem grafisk i sanntid til brukeren. Likevel vektlegges det at alle modeller som utprøves og testes i denne oppgaven burde kunne overføres til en smartklokke og fungere uten problemer. Men, ettersom selve implementasjonen på smartklokke krever mye arbeid og ikke innfører noe verdiskapende nyvinninger i seg selv, står prosjektoppgaven (1) som et et eksempel på hva som kan utføres lokalt på en smartklokke. Under samtalene med Laerdal Medical før oppgaven startet ble det nevnt at en presisjon på dybdeestimater under $\pm 3mm$ eller er ønskelig for at det skal kunne brukes som en veileder. Målet for oppgaven er da å estimere kompresjonsdybde med et gjennomsnittlig avvik på under 3mm.

All analyse og diskusjon av resultater, samt implementasjon av hardware og applikasjoner i denne oppgaven er laget av undertegnede hvor annet ikke er spesifisert. Unntakene er programvare-biblioteker som blir benyttet i applikasjonene for smartklokken og dybdemåleren i annedukken. Applikasjonen til smartklokken bygger på de samme fundamentene som rytme-måleren i prosjektoppgaven, men applikasjonen ble laget på ny for det nye formålet i denne masteroppgaven.

KAPITTEL 2

Teori

For å lage en veileder til HLR er det nødvendig å ha riktige retningslinjer på plass. Hva er egentlig riktig kompresjonsdybde og rytme for hjerte og lungeredning? Dette ble nøye undersøkt i prosjektoppgaven (1). Resten av dette kapittelet er en oppsummering av prosjektoppgavens teori med et nytt fokus på hva som er problemstillingen i denne oppgaven.

2.1 Hjerte-Lunge-Redning

Hjerte-lunge-redning er en førstehjelpsprosedyre brukt for å opprettholde blodsirkulasjonen hos en person med hjertestans i et forsøk på å minimere sjansen for alvorlig hjerneskade. Dette gjøres vanligvis med en kombinasjon av brystkompresjoner og innblåsinger via den skadedes luftkanaler for å manuelt pumpe oksygenrikt blod til hjernen. Brystkompresjoner utføres ved å plassere hendene på den skadedes brystkasse og med strake armer komprimere brystkassen. Formålet med HLR er hovedsakelig å holde hjerneceller i live med oksygentilførsel, inntil en hjertestarter blir tilgjengelig. Det er veldig sjelden at HLR alene gjenoppliver et menneske. I en dansk studie av 7623 forekomster av hjertestopp utenfor sykehus, ble det vist at tilskuer-HLR mer enn doblet sjansen for å overleve, selv ved lange responstider (4). Premisset for en HLR-assistent er tross alt at HLR faktisk redder liv. I likhet med den danske artikkelen, forteller all forskning og retningslinjer fra Norges folkehelseinstitutt, USAs American Health Assosiation (AHA) og Storbritannias Resuscitation Council (RC UK) at HLR er essensielt ved hjertestans for å redde livet til den skadede.

2.2 Dybde og rytme under HLR

Tabell 2.1: Veiledende retningslinjer for HLR, hentet fra RC UK (5)

Egenskaper	Beskrivelse
Dybde: Voksne	50-60 millimeter
Dybde: Barn	Minst 1/3 av brystets dybde
	Ca. 40 millimeter for spedbarn
	Ca. 50 millimeter for eldre barn
Rytme	100-120 Kompresjoner i minuttet
Innblåsing	30 Kompresjoner + 2 Innblåsinger

En viktig faktor for kvaliteten av brystkompresjoner er at de må være dype nok. Tidligere internasjonal konsensus for HLR har vært at kompresjoner skal være minst 50mm. I prosjektoppgaven ble det oppdaget en trend som tyder på at gjennomsnittlige kompresjonsdybder på 45-55mm gir høyst overlevelsesrate. Utfordringen med studiene som ble undersøkt er at usikkerheten er stor, ettersom det er veldig krevende å samle inn store mengder data om kompresjonsdybde for hjertestanser. Dette er delvis fordi det er komplisert å samle inn pasientdata samt at personer som utfører HLR utenfor sykehus ikke har noen nøyaktig måte for å måle kompresjonsdybder. AHA og RC UK sine retningslinjer sier at høyest overlevningsrate skal bli oppnådd med kompresjoner i sjiktet mellom 50 og 60mm og rytme på mellom 100 til 120 slag i minuttet (5). Dette er retningslinjene denne oppgaven og prosjektoppgaven baserer seg på, og kan sees i tabell 2.1.

Et annet poeng som gikk igjen fra undersøkelsene i prosjektoppgaven er at det ikke er spesielt farlig om hjertekompresjonene er for dype. Kompresjoner dypere enn 60mm øker sannsynligvis ikke sjansen for overlevelse, men det medfører større sjans for brukne ribbein. For grunne kompresjoner derimot, kan føre til markant lavere sjanse for å overleve (1). Det viktigste er derfor at kompresjonene er dypere enn $\sim 45mm$. For dybdeestimeringen sin del i denne oppgaven så har ikke disse retningslinjene så veldig mye å si, fordi estimeringen må kunne fungere for både grunne og for dype målinger.

2.3 Lignende løsninger

Til smarttelefoner eksisterer det løsninger for assistanse til HLR. Disse baserer seg på å veilede brukeren i form av å fortelle dem tekstbasert hvordan HLR skal utføres og kan kontakte myndighetene for deg. Bare et fåtall av disse gir derimot sanntid tilbakemelding på HLR-kvaliteten man faktisk utøver. Eksempler på produkter som gir sanntid tilbakemelding på HLR er:

- Zoll sin "PocketCPR", android-applikasjon
- Sør-Koreanske MElabs "UCPR", android-applikasjon
- Samaid armbånd
- Laerdals CPRmeter2

Løsningene til Android smarttelefoner krever at telefonen legges mellom håndflaten og pasientens bryst mens HLR utføres, og bruker de interne sensorene i smarttelefonen til å gi sanntid tilbakemelding av dybde og rytme. Laerdal Medical sitt CPRmeter2 fungerer på en lignende måte, med et proprietært apparat. Ulempen med disse teknologiene er at man må holde apparatet/telefonen mens man utfører HLR, som kan være klumsete. Samaid har laget et eget proprietært armbånd som pares med en telefon, som så gir tilbakemelding i sanntid. Det er med andre ord et lignende prinsipp som ved å utnytte de interne sensorene i en smartklokke. Dette er dog ment for undervisning og båndet er ikke beregnet for å ha med seg rundt daglig.

Implementeres denne teknologien i en smartklokke istedenfor, vil mange ha med seg denne teknologien hvor enn de går. Under HLR ville man da ikke trenge å stresse med å holde mobilen korrekt, da klokken er noe man naturlig har på seg. I rapporten til A. Hove ble dette forsøkt med en smartklokke, der konklusjonen var at gjenkjenning av rytme og dybde med smartklokke ser ut til å ha stort potensial (6). Samme syn deler forskningen til Grünerbl et al. og Song et al. om samme tema (7) (8). De konkluderer i tillegg med at HLR-kvaliteten bedres etter opplæring med en smartklokke-assistent. Det har ikke blitt forsket mye på resultatene og nøyaktigheten til disse løsningene, og dette er noe som burde testes videre før det blir kommersielt. Ved lite testing er det usikkert hvor korrekte resultatene blir av dobbeltintegrasjon av akselerasjon som dybdeestimering hvis man bytter hånd eller i situasjoner hvor klokken endrer rotasjon. Mye av den tidligere forskningen baserer seg også på veldig kontrollerte og små treningsett, enten gjort av robotarmer eller av en person. Fokus i denne oppgaven vil være å grundig analysere de forskjellige mulighetene man har for dybdeestimering basert på de interne sensorene som finnes i en smartklokke. I tilegg skal det undersøkes hva slags nyttig informasjon som finnes i smartklokkens interne gyrometer. Oppgaven skal forsøke å gi et svar for hvor bra nøyaktighet en smartklokke-assistent har i praktiske situasjoner. *Multivariate Data Analysis* benyttes også for å undersøke hva slags informasjon smartklokkens sensorer forteller oss.

KAPITTEL 3

Verktøy

3.1 Utstyr

Som beskrevet i kapittel 1 er et av målene for dette prosjektet at det skal være mulig å ha med seg en HLR-assistent på en smartklokke. Selv om det er mulig å teste denne teknologien på proprietært utstyr først, var det interessant for oppgaven sin del å prøve å utvikle dette direkte på en smartklokke. Ved å gjøre det på denne måten var det mulig å se begrensningene som finnes på en smartklokke mens utviklingen foregikk. For å kunne utføre eksperimenter og teste algoritmer var også en gjenopplivningsdukke nødvendig. I dette kapittelet blir utstyret i rapporten introdusert. Formålet, den faktiske implementasjonen av utstyret og hvordan det ble brukt forklares nærmere i kapittel 4.

3.2 Smartklokke: LG G Watch R

Figur 3.1: Bilde av smartklokken brukt i oppgaven (9)

En smartklokke basert på Android ble valgt for å gjøre prosjektet så tilgjengelig som mulig. Smartklokken LG G Watch R har akselerometer og gyro i 3 akser. Sensorene har en maksimal oppdateringsfrekvens på 200hz. Denne ble valgt på grunn av tilfredstillende raske sensorer og fordi den hadde nyeste tilgjengelige utgave av Android Wear. Maskinvaren antas også til å være representativ for "den gjennomsnittlige smartklokke", som setter et passelig minimumkrav for ytelse. Dybde-estimeringsalgoritmene foreslått i oppgaven tar med andre ord utgangspunkt at skal kunne kjøre på akkurat denne smartklokken.

Tabell 3.1: Spesifikasjoner for LG G Watch R (9)

Egenskaper	**Beskrivelse**
Operativsystem	Android Wear 2.0
Skjerm	Rund @ 320x320
Prosessor	Quad core 1.2 Ghz Cortex A7
Minne	512MB RAM
Lagring	4GB
Akselerometer	3-Dimensjonal, 200hz frekvens, $\pm$ 2g rekkevidde
Gyrometer	3-Dimensjonal, 200hz frekvens
Tilkoblinger	WiFi, USB, Bluetooth 4.0

3.3 Gjenopplivningsdukke: Little Anne

Figur 3.2: Illustrasjonsbilde av gjennoplivingsdukken brukt (10)

En gjenopplivningsdukke av modellen *Little Anne* ble utlånt fra Laerdal Medical. I brystet på denne *annedukken* er det en fjær som simulerer kraften som må til for å utføre en brystkompresjon. Den gir også fra seg et hørbart klikk om brystet har blitt komprimert tilstrekkelig. Klikket oppstår rundt $50mm$ og kan skrues av og på. Dukken har originalt et system for å veilede brukeren mens man utfører HLR ved hjelp av en dybdemåler plassert under brystplaten, altså inni dukken. Dette systemet kan kommunisere med bluetooth til en applikasjon på smartmobil som grafisk fremstiller kompresjonsdybde og rytme. Etter en fullført HLR-sekvens har man mulighet til å lagre sekvensen lokalt på smartmobilen som en *.ssx-fil*. Applikasjonen finnes både til iOS og Android, og er skrevet i c#. Det som blir lagret for hver kompresjon i denne applikasjonen er fremstilt i figur 3.3.

```
<evt type="CprCompEvent" msecs="606016">
    <param type="endTime" value="606016" units="none"/>
    <param type="compDepth" value="47" units="mm"/>
    <param type="compPeakTime" value="605744" units="ms"/>
    <param type="compIncompleteRelease" value="False" units="percent"/>
    <param type="compMeanRate" value="110" units="cpm"/>
</evt>
```

Figur 3.3: Utsnitt av dataen som blir lagret under en HLR-sekvens i Laerdals applikasjon til smartmobil

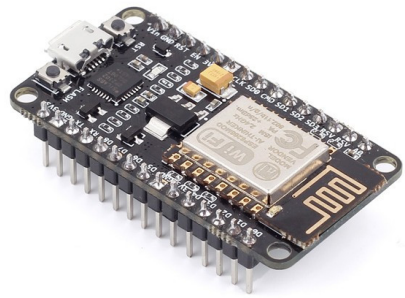
Som en kan se fra figur 3.3 er det relativt lite informasjon som blir lagret for hver kompresjon. Etter erfaringer fra tidligere arbeid hvor disse filene ble benyttet, ble det klart at det er tungvint og unøyaktig å bruke denne dataen (1). For det første er tidspunktet *CprCompEvent* gitt i *millisekunder siden appen ble startet på telefonen* som gjør at det er vanskelig å synkronisere dataen fra denne filen med sensor-data fra smartklokken. I

tilegg bidro det til en lite strømlinjeformet prosess ettersom ssx-filen med dukkens kompresjonsdata måtte overføres fra smartmobilen. Derfor var det nødvendig med en ny løsning for å måle den reele kompresjonsdybden til dukkens bryst. Følgende løsninger ble vurdert:

- Bruk av et kamera, hvor dukken og kamera hadde vært i en fast stilling. Dette er en løsning som krever mye behandling i etterkant og gjør synkronisering av måleserier vanskelig.
- Forsøke å *endre* applikasjonen til Laerdal. Uten å vite spesifikasjonene til sensoren fra Laerdals innebygde teknologi var det ikke sikkert det var mulig. Det ville også vært tidkrevende å forstå og manipulere et svært c#-prosjekt uten tidligere erfaring.
- Installere en egen proprietær dybdemåler basert på rimelige sensorer og open-source biblioteker til en populær og enkelt programmerbar mikrokontroller.

Avgjørelsen falt på det siste alternativet. Det ble derfor implementert en egen løsning for måling av kompresjonsdybde som blir detaljert beskrevet i kapittel 4. Med en arduino-lignende mikrokontroller finnes det masse ferdige biblioteker, og dette er en av de enkleste og rimeligste måtene tilgjengelig for å kunne interagere med sensorer. I tilegg finnes det bluetooth-moduler for mikrokontrollere, som betydde at det var mulig å sette sammen en fullverdig løsning på innsiden av annedukken. En sensor for å måle distanse, en bluetooth-modul for å sende denne dataen til smartklokken og en mikrokontroller for å binde alt sammen. Implementasjonen av dette kommer i kapittel 4.

3.4 Mikrokontroller - NodeMCU ESP8266



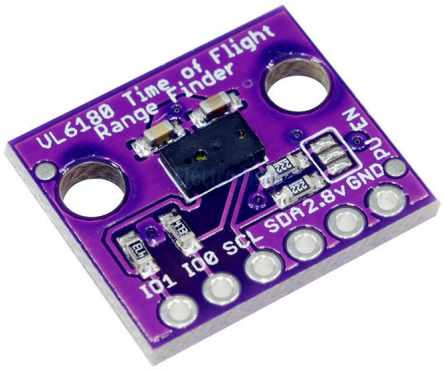
Figur 3.4: Bilde av mikrokontrolleren ESP8266 brukt i oppgaven (11)

Mikrokontrolleren (MCU) er hjernen av dybdemåleren som implementeres i annedukken, og måtte være i stand til å kommunisere med utvalgte sensorer og bluetooth-moduler. Valget for mikrokontroller falt dermed på en mikrokontroller av typen NodeMCU. NodeMCU er en *open source* IoT platform som blir drevet av brikken ESP8266 med WiFi-tilkobling. Spesifikasjonene til MCU'en er gitt i 3.2. Denne MCU'en er mye utbredt og det finnes kompilatorer for en rekke programmeringsspråk. Tilgang til eksisterende biblioteker var en grunn til at denne mikrokontrolleren ble valgt, samtidig som den ble antatt til å være rask nok for de forholdsvis enkle oppgavene den skulle utføre. En slik enkel mikrokontroller ble valgt over for eksempel raspberry pi, da slike mer avanserte maskiner ofte har begrensede sanntids-egenskaper og er mer kompliserte enn nødvendig.

Tabell 3.2: Spesifikasjoner for NodeMCU (12)

Egenskaper	**Beskrivelse**
Prosessor	80 eller 160 MHz
Minne	128 KiB RAM og 4MB Flash
Strøm	3.3-5v
Kommunikasjon	I^2C, SPI, WiFi
Serial	921600 Baud rate
Språk	c, c++, arduino, Lua, MicroPython

3.5 Distanse-sensor i annedukke



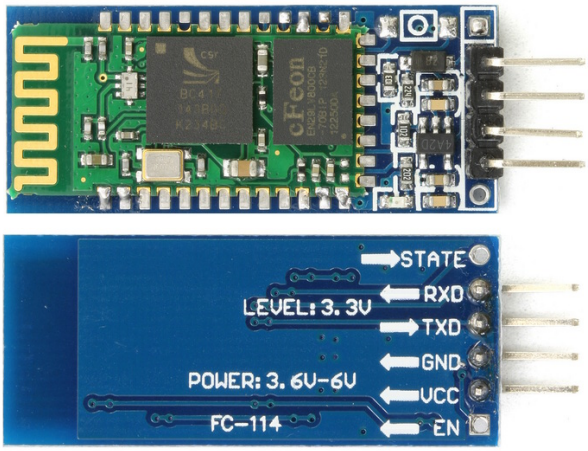
Figur 3.5: Bilde av distansesensoren VL6180 brukt i oppgaven (11)

For å måle kompresjonsdybden på innsiden av annedukken var det nødvendig med en distanse-sensor. Til dette formålet finnes det både optiske og aukustiske målere med forskjellige oppdateringsfrekvenser og presisjon. Valget falt på den I^2C-baserte sensoren VL6180X, produsert av STMicroelectronics. VL6180X er en *Time of Flight*-modul som måler tiden det tar før lyset fra en laser blir reflektert tilbake til kilden. Sensoren har begrenset synsfelt og klarer bare å se i retning, men ettersom distanse-sensoren er fast montert på innsiden av en dukke medførte ikke det noe problemer. Siden modulen baserer seg på lys som kilde betyr dette også at målingene kommer omtrent uten forsinkelse i motsetning til akustiske. Frekvensen man kan måle med avhenger av flere parametre som blant annet bestemmer hvor lenge den maksimalt skal vente på å få lys returnert tilbake til sensoren. Hvordan disse parametrene er satt bestemmer samplings-rate, nøyaktighet, støy og maksimal lengde som er mulig å måle. I denne rapporten ble de anbefalte verdiene gitt i databladet brukt (13). Med disse verdiene ble den praktiske samplings-frekvens rundt 80 hz som viste seg å være tilstrekkelig.

Tabell 3.3: Spesifikasjoner for VL6180X Time of Flight sensor (13)

Egenskaper	**Beskrivelse**
Rekkevidde	5mm til 100mm
Kommunikasjon	I^2C
Egenskaper	Distansemåler, lys-styrke-måler
Strøm	2.8V
Oppløsning	1mm

3.6 Bluetoothmodul HC-06



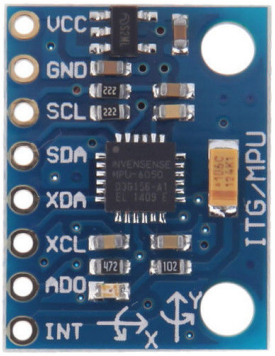
Figur 3.6: Bilde av bluetoothmodulen brukt for at smartklokken og gjennopplivingsdukken skulle kunne kommunisere. (14)

Bluetoothmodulen HC-06 er en mye brukt kommunikasjonsmodul for mikrokontrollere. Modulen opererer som en seriell-port det er mulig å skrive og lese data fra. HC-06 kan bare være slave i en bluetooth-tilkobling, så smartklokken vil alltid være master i dette prosjektet. Navnet, pin-kode og seriell hastighet kan settes ved hjelp av *AT-commands*. Seriell hastighet kan settes stegvis fra en baudrate på 1200 til 1382400. Merk at dette bare påvirker hastigheten informasjonen flyter mellom bluetooth-modulen og mikrokontrolleren, og har derfor ikke noe direkte effekt på overføringshastigheten mellom smartklokken. Likevel vil en for lav baud-rate over denne serielle tilkoblingen føre til en flaskehals for bluetoothoverføringen, der bufferet i bluetoothmodulen vil fylles opp med meldinger (15).

Tabell 3.4: Spesifikasjoner for HC-06 (15)

Egenskaper	**Beskrivelse**
Seriell tilkobling	1200 til 1382400 Baud rate med UART
Strøm	3.3v
Bluetooth	Bluetooth 2.0 @ 2.4GHz
Bluetooth hastighet	1Mbps/1Mbps(Synchronous)

3.7 Akselerometer og gyroskop



Figur 3.7: Bilde av akselerometer installert i gjennopplivingsdukken (11)

I denne oppgaven benyttes det et akselerometer og gyroskop i smartklokken samt et i systemet inni i dukken. Felles for begge akselerometrene er at de måler *proper akselerasjon* $[m/s^2]$. Proper akselerasjon er kraften et legeme blir utsatt for et gitt tidspunkt. Ligger sensoren helt i ro vil sensoren måle en kraft tilsvarende 1g, ettersom dette er en kraft gjenstanden alltid blir utsatt for. Fritt fall vil på andre siden gi målinger på 0 m/s^2. Både akselerometrene og gyroskopene i dukken og smartklokken er kapasitiv-baserte Micro Electro Mechanical Systems (MEMS). Gyroskopene måler vinkelhastighet i radianer per sekund, i tre akser.

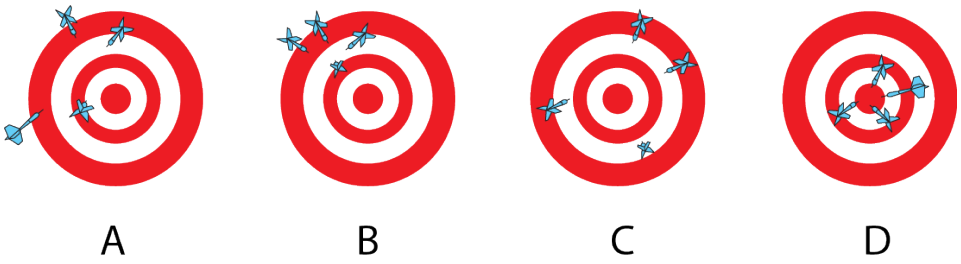
Akselerometeret som ble installert i annedukken er en sensor kalt MPU-6050, som er et lavkost MEMS kombinert akselerometer og gyroskop. Sensor-modulen kommuniserer med mikrokontrollere med I^2C. Oppløsningen på målingene avhenger av rekkevidden valgt (16). Grunnen til at det var ønskelig med et akselerometer i annedukken var for å se om dette kunne brukes til å forbedre dybde-estimatene basert på smartklokkens akselerometre. Tanken var å prøve å finne sammenhenger og forskjeller mellom den akselerasjonen klokken utsettes for i forhold til akselerasjonen som brystplaten opplever. Dette ble utforsket uten at det ga noe spesielt interessant informasjon og ble derfor utelatt av oppgaven. Akselerometeret i annedukken ble imidlertid brukt for å kalibrere tidsforskjellen mellom smartklokken og mikrokontrolleren.

Tabell 3.5: Spesifikasjoner for MPU6050

Egenskaper	**Beskrivelse**
Sensor	Akselerometer og Gyrometer i x-, y- og z-retning
Rekkevidde	$\pm 2g$ til $\pm 16g$
Strøm	3.3-5v
Kommunikasjon	I^2C
Samplings-rate	Maksimalt 8kHz for gyro og 1Khz for akselerometer

Akselerometeret i smartklokken er litt mer begrenset enn kontrollen som er tilgjengelig på MPU6050. I android er det mye større begrensinger på sanntid ytelse enn for mikrokontrolleren. Akselerometeret i smartklokken har en maks rekkevidde på $\pm 2g$ og maksimal oppdateringsfrekvens på 200hz. Android sensorer blir registrerte som egne objekter, som bakgrunnstråder i Android oppdaterer med en gitt frekvens. En hake er at forsinkelsen fra en måling blir tatt til den blir levert til trådene som lytter etter endringer fra sensorene ikke er konstant. Oppdateres smartklokkens sensorer med $\sim$100Hz, betyr bare det at det ca blir gjort 100 målinger per sekund. Det kan i realiteten være alt fra 95-105, hvor hver måling kommer med 5-15ms mellomrom. Det samme gjelder gyrometer-målingene på Android. Dette er ikke nødvendigvis et problem, men det er noe som gjerne må tas hensyn til (17).

3.8 Utstyrets nøyaktighet og presisjon



Figur 3.8: Illustrasjon av presisjon og nøyaktighet. B fremstiller høy presisjon mens C fremstiller høy nøyaktighet. For A er begge deler lave, mens D har både høy presisjon og nøyaktighet (18)

I denne oppgaven blir det skilt mellom *nøyaktighet* og *presisjon*. Nøyaktigheten til en sensor angir hvor like eksperimentelle målinger blir i forhold til en teoretisk referanse. Med presisjon menes *spredningen* av resultater, altså hvor godt de eksperimentelle målingene stemmer med hverandre. Figur 3.8 illustrerer denne forskjellen. 3.8(B) fremstiller høy presisjon, ettersom alle målingene er tett samlet, men disse målingene har lav nøyaktighet. 3.8(C) har høy nøyaktighet fordi snittet av målingene ligger på origo, men har stor spredning (lav presisjon). Høy presisjon og nøyaktighet blir fremstilt i 3.8(D) hvor samlingen av målinger både er tette og på origo.

Muligens den viktigste sensoren i denne oppgaven er dybdemåleren VL6180. Målingene fra denne sensoren blir brukt som fasit/refreanse for alle dybde-estimater, og det er derfor viktig med så presise målinger som mulig. Det er denne dybden som skal estimeres basert på målingene fra smartklokken. Presisjonen til VL6180x oppgis til å være avhengig av refleksjonsfaktoren til materialet den måler avstand mot, målefrekvens og lysstøy. Ettersom målingene til sensoren bare har en dimensjon, kan nøyaktigheten til de eksperimentelle målingene bli justert for ved å kalibrere med en forskyvning. Inni annedukken er det en helt hvit halvblank og flat overflate som kan antas til å ha en høy refleksjonsfaktor som skal være fordelaktig for å høyest mulig presisjon. Etter at sensoren ble installert i dukken ble nøyaktigheten og presisjonen testet og resultatene av $\sim$14000 målinger vises i tabell 3.6.

Tabell 3.6: Testresultater av nøyaktigheten til VL6180X

Referanse	Median	Gjennomsnitt (MSD)	MAD	MSD	RMSE
0mm	0mm	0.069mm	0.926mm	1.595mm	1.253mm

Testen ble utført med oppvarmet og kalibrert sensor hvor dukken lå helt i ro. I dette tilfellet ble sensoren kalibrert på forhånd, slik at alle målingene teoretisk sett skulle bli 0mm. Resultatene viser at sensoren er relativt nøyaktig når den er riktig kalibrert ettersom gjennomsnittsmålingen ligger veldig nærme referansen. Med andre ord vil målingene i gjennomsnitt bli det samme. *Mean Signed Deviation* henviser til det gjennomsnittlige avviket for hver måling, og er et tegn for nøyaktigheten til et sett med målinger. I testen

er referansen 0, så i dette tilfellet vil MSD være det samme som gjennomsnittet av alle målingene. Hadde 0.069mm blitt trukket fra hver måling hadde altså MSD blitt 0. MSD beskriver nøyaktigheten til måleserien og kan enkelt justeres for med et *offset*.

I et forsøk på å kvantifisere presisjonen og spredningen av målingene ble i tilegg *Mean Absolute Deviation* og *Root Mean Square Error* kalkulert. MAD forteller hva det gjennomsnittlige avviket i absoluttverdi er for alle eksperimentelle målinger, som i dette tilfellet er tilnærmet lik $\pm 1mm$. RMSE kvadrer hver måling og gir derfor større straff jo større avviket er. Presisjonen på disse målingene fikk ikke undertegnede til å forbedre på noen måte, og presisjonen vil være tilnærmet lik under alle tester som blir utført. Men, nøyaktigheten avhenger av temperatur og lys fra omgivelsene rundt, og det er viktig at dybdesensoren blir riktig kalibrert før hver test for å få best mulig kontinuitet og nøyaktige målinger fra dukken. Med tanke på presisjonen til denne sensoren kan det ikke forventes at resultatene til en utviklet algoritme for dybde-estimering kan ha lavere RMSE eller MAD enn for denne dybdesensoren.

Det oppstod et lite problem under testing med smartklokkens akselerometer. Under kompresjoner med stor akselerasjon, ble smartklokkens akselerometer "makset" ut, altså at målingene ble det maksimale av hva de kunne bli. Større akselerasjon enn $\pm 19,3 m/s^2$ blir bare målt som 19.3. For at dette skal oppstå må både smartklokken ha stor akselerasjon, og vinkelen til smartklokken må være slik at mesteparten av kraften bare er i en akse. Holdes smartklokken vannrett og beveges rett opp, vil følgelig all akselerasjon bli målt i smartklokkens y-akse. Er det litt vinkel på klokken blir akselerasjonen "fordelt" utover x-, y- og z-målingene av akselerasjon som kan forhindre problemet. Videre i oppgaven blir dette kalt *clipping*. Dype kompresjoner er de som blir utsatt for størst akselerasjon. Det viste seg at en del av disse kompresjonene ble tolket som grunnere enn de egentlig var på grunn av at akselerasjonen ble clippet. Dette er selvsagt ikke ønskelig, og effekten av det diskuteres i kapittel 7.

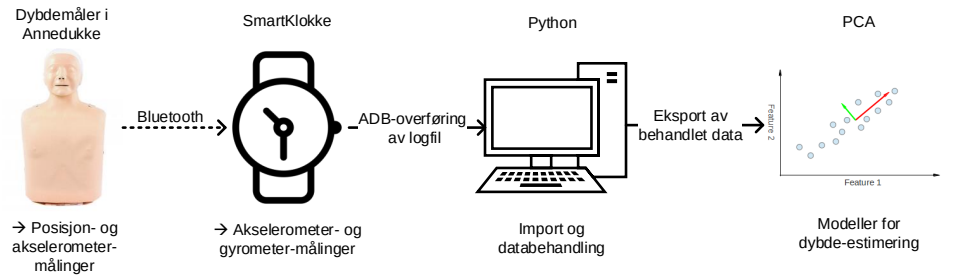
KAPITTEL 4

Implementasjon

Hittil er det etablert hva slags utstyr som er nødvendig for å kunne gjøre en analyse av dybdeestimering. Det var ønskelig med en enhet som måler dybden internt i Anne-dukken, samt det var nødvendig med en enhet for å lese akselerasjonen fra håndleddet mens man utfører hjerte-lunge-redning. Dybdemåleren i Anne-dukken, har både et akselerometer og en dybdemåler som diskutert i kapittel 3. Før dette systemet ble designet, var det et par problemstillinger å ta hensyn til for å få nøyaktige målinger. Det første var det fysiske oppsettet og hvordan dette skulle løses i praksis, og det andre problemet var hvordan målingene gjort av dukken og smartklokken skulle sammenlignes. Det praktiske og ferdige oppsettet vises først i 4.1, og deretter blir koden og den faktiske implementasjonen gjennomgått i resten av kapittel 4.

4.1 Oppsett

For å få samtidige målinger fra dybdesensoren i annedukken og akselerasjonsmålinger fra smartklokken ble det implementert et system hvor disse enhetene kommuniserer med bluetooth. Under en HLR-sekvens blir sensordataen fra annedukken og smartklokken logget internt på smartklokken. En HLR-sekvens er en "treningsøkt" av hjertekompresjoner utført på annedukken. Teknikk under slike sekvenser er egentlig ikke så viktig, ettersom dybdeestimeringen må kunne fungere på både unormalt dype og grunne kompresjoner og med lav og høy rytme. Etter en HLR-sekvens er over blir denne sensordataen lagret i en logfil *(LogFile.txt)* og deretter overført fra smartklokken til en datamaskin.



Figur 4.1: Oversikt over dataflyt mellom enheter (19) (20) (21)

Hver linje i *LogFile.txt* inneholder et tidspunkt og en rekke medfølgende målinger. Hver unike måling/variabel skilles med en tab, og målingene som blir logget hvert tidspunkt er gitt i tabell 4.1. Etter dette blir logfilen importert til Python for analyse. All dataen fra en HLR-sekvens lastes inn før måleserien blir delt opp slik at hver kompresjon er separate. Deretter blir kompresjonene behandlet og analysert før det blir eksportert som en ny fil *(cprList.txt)* hvor hver *kompresjon* er en linje. Smartklokken lager altså en fil der hver linje er tidssteg og deres målinger, mens den behandlede dataen i python gir en fil hvor hver linje er en kompresjon og dens egenskaper. Denne filen har separert kompresjonene og kalkulert en rekke informasjon om hver kompresjon. Dette er et nødvendig steg for å kunne analysere forholdene bak hver kompresjon og for å forbedre dybde-estimater med maskinlæringsmetoder. En oversikt av denne flyten vises i figur 4.1.

Tabell 4.1: Informasjon som blir lagret for hver linje i *LogFile*. Logfile er filen med måledata som blir lagret på smartklokken etter en HLR-sekvens

Egenskaper	**Beskrivelse**
dateInputMilliSec	Tidspunkt i unix-tid
mag	Magnituden av akselerasjon
x, y, z	3 Verdier for akselerasjons-måling i x, y, og z retning
gx, gy, gz	3 Verdier for gyrometer-måling i x, y, og z retning
rz	z-komponenten av smartklokkens rotasjonsmatrise
mm	Dybdemåling fra Annedukken
x_a, y_a, z_a	3 akselerasjons-målinger for x, y, og z retning fra Annedukken

4.2 Kommunikasjonsproblemet

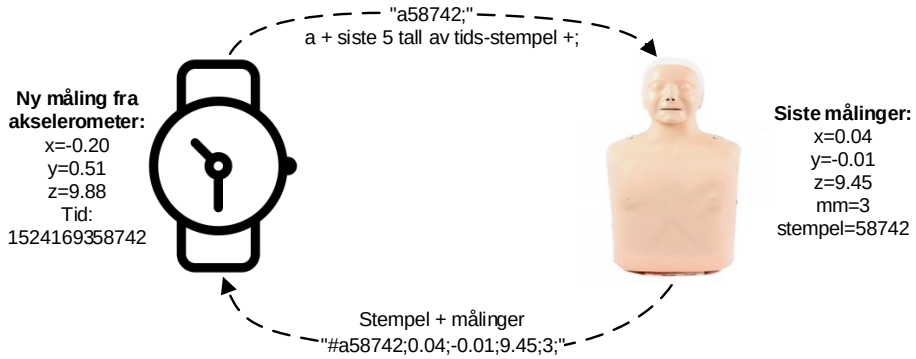
Det var ønskelig å sammenligne akselerometerdata fra smartklokken med dybdemålinger fra annedukken så detaljert som mulig for å få best mulig innsikt i problemstillingen. Derfor var det nødvendig å få *samtidige* målinger fra klokken og dybdemåleren i dukken. Problemet med å måle med to forskjellige enheter slik det blir gjort i dette arbeidet er at målingene er vanskelige å korrekt synkronisere i tid ettersom det ikke er mulig å ha en kablet tilkobling mellom dem. En mulighet hadde vært å skrive dybdemålingene fra Anne-dukken til en seriell-port på en PC og logget disse til en fil. Å skrive til seriell med en Android smartklokke er ikke like enkelt, så den mest realistiske løsningen her å logge dataen til en fil på smartklokken som så overføres til pcen i etterkant. Disse filene kunne blitt forsøkt synkronisert med behandling i ettertid på PC, men dette medfører en rekke utfordringer:

- Ulike tidsstempler på målinger fra smartklokke og dybdemåler ettersom målingene ikke skjer samtidig.
- Forskyvning i tid mellom smartklokke og dybdemåler. Det er ingen enkel måte å synkronisere klokkene til de to enhetene nøyaktig likt.
- Forskjellig frekvens på målinger fra de to enhetene: Sanntid-begrensninger i Android gjør det umulig å ha en spesifikk tidsforskjell mellom hver nye måling. En målefrekvens på 100hz gir ca 100 målinger i sekundet, men målingene kan komme uregelmessig med 5-15 millisekund mellomrom.

Et problem som vanskeliggjør synkroniseringen av tid er at akselerasjons-målingene var ønskelige å ha på 100hz, som betyr at en relativt liten tidsforskyvning på 100ms utgjør mer enn 10 målinger i forskjell. Dette viste seg til å være nok til å forårsake uønsket oppførsel for algoritmene som gjenkjenner kompresjoner. Noen av problemene kan til dels rettes opp med post-prosessering, men det hadde krevd en del arbeid og muligens ikke gitt tilfredstillende resultater. Ettersom det er veldig vanskelig å ha en definitiv måte å synkronisere disse måleseriene i tid ble det foreslått en løsning basert på bluetooth diskutert i 4.3. Ideen med bluetooth ble altså introdusert for å ha kontroll over tidsforskjellen i målinger mellom smartklokken og Anne-dukken, samtidig som man da får samme tidsstempler for hver måling.

Å få samtidige målinger kunne også blitt løst ved å droppe smartklokke helt og brukt en proprietær løsning hvor annedukken og et akselerometer-amrbånd var koblet til en felles mikrokontroller. Dette var ikke ønskelig ettersom friheten og muligheten til en MCU ikke er representativ for hva som faktisk går an i virkeligheten med Android- eller iOS-baserte smartklokker.

4.3 Bluetooth kommunikasjon



Figur 4.2: Bluetooth-kommunikasjon mellom dybdemåleren i Anne-dukken og smartklokken (19)(20)

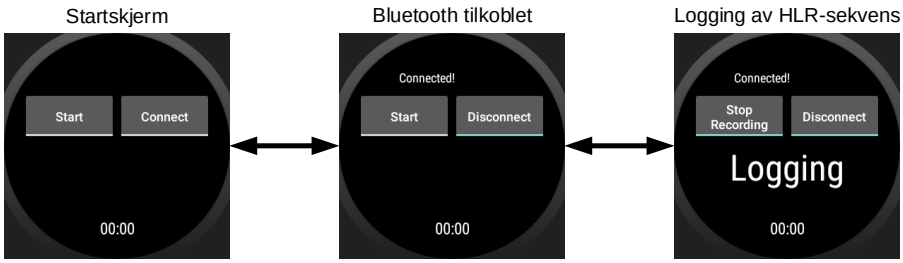
Den overordnede ideen med overføring av målinger over Bluetooth vises i figur 4.2. Smartklokken mottar kontinuerlig nye målinger i x-, y- og z-retning fra dets interne akselerometer, og mottar i tilegg tidspunktet for målingen. Tanken er at i det smartklokken mottar en ny måling, sendes den siste delen av tidspunktet for denne målingen via Bluetooth til dybdemåleren. Dybdemåleren mottar dette klokkeslettet, og kombinerer det med sin siste måling. Deretter sender dybdemåleren tidspunktet den fikk av smartklokken sammen med målingene tatt internt i Anne-dukken. Meldingen som blir overført fra Anne-dukken tilbake til smartklokken er fremstilt i formel 4.1. Smartklokken lagrer så denne mottatte verdien sammen med målingene fra smartklokkens eget akselerometer.

$$\underbrace{a58742}_{\text{Tids-stempel}}\ ;\ \underbrace{0.04}_{Acc_x};\ \underbrace{-0.01}_{Acc_y};\ \underbrace{9.45}_{Acc_z};\ \underbrace{3}_{\text{mm}}\ ; \tag{4.1}$$

Ettersom Bluetooth-modulen til dybdemåleren er koblet med UART til mikrokontrolleren, har man en begrensning på hastigheten det er mulig å overføre med. Dette er grunnen til det bare ble valgt å sende en del av tids-stempelet, for å begrense muligheten for pakke-tap. En melding for overførsel inneholder *32 chars*, der hver char består av et 8bit ASCII-tegn. En melding er da 256 bits, og for for å opprettholde en frekvens på 100hz krever det derfor at bluetooth-tilkoblingen til dybdemåleren minst kan sende i 25600 bits/sekund. I tilegg kommer datastrømmen som går fra smartklokken til Anne-dukken som også tar en del av båndbredden. Selv om en baud-rate rundt 100000 bits/s burde være nok, viste det seg under praktisk bruk at en tilkobling på 921600 bits/s var nødvendig for å opprettholde stabil kommunikasjon. Dette kan skyldes at Android-enheten kan finne på å sende en bulk med meldinger som medfører at dybdemåleren må kunne både motta informasjon og sende raskere enn under normal operasjon. Baud-rate på 921600 er forresten det maksimale en NodeMcu kan kommunisere med over seriell tilkobling, og skulle dette prosjektet utvides av en eller annen grunn måtte kanskje valget av mikrokontroller blitt revurdert.

4.4 Android Smartklokke applikasjon

I dette prosjektet trengtes det en applikasjon til en Android-basert smartklokke som skulle logge sensordata og motta målinger fra en mikrokontroller over bluetooth. Applikasjonen ble laget spesifikt for denne oppgaven, og er basert på kunnskapen fra prosjektoppgaven om samme tema (1). Applikasjonen har en relativt enkel oppbygging og et begrenset brukergrensesnitt som vises i figur 4.3.



Figur 4.3: Brukergrensesnittet for Android-applikasjonen LittleAnne til smartklokke.

Før det er mulig å starte å logge, må bluetoothenheten i dybdemåleren i klokken kobles til smartklokken. Denne bluetoothtilkoblingen initailseres av at knappen ”connect” presses inn. Da vil smartklokken bruke noen sekunder på å etablere en tilkobling til dybdemåleren. Dette gjøres ved at det opprettes en *bluetooth-socket* for tilkoblingen, som er et klasseobjekt fra et Android-bibliotek som oppretter en TCP-lignende oppkobling mellom bluetooth-enhetene. Mac-adressen til dybdemålerens bluetooth-modul er kjent, for å sørge for at smartklokken kobler seg til riktig enhet. Ved bruk av en slik bluetooth-socket tar Android seg selv av vedlikehold av tilkoblingen og muliggjør data-overføring igjennom socketen. Denne socketen har en *ouputStream* og en *inputStream*. OutputStream er en funksjon til Androids bluetooth-socket man kan bruke for å skrive *byteArrays* til. Det vil si at en streng med tekst kan transformeres til et byteArray med ASCII-karakterer, som så kan skrives til bluetooth-socketens outputStream. Bluetooth-socketen åpner med andre ord en seriell tilkobling mellom enhetene ved bruk av outputStream og inputStream. Android vil selv håndtere denne outputStreamen i bakgrunnen og videresende tillagte meldinger til den tilkoblede bluetooth-enheten.

For lesing av data fra bluetooth-socketen anbefales det av Android å lage en egen tråd for dette. Tråden *listenForData* blir opprettet for dette formålet. ListenForData-tråden ligger i bakgrunnen og leser det som kommer på bluetooth-socketens inputstream. Dataflyten som kommer her blir gjort om til en string og lagt til et buffer. Når en hel melding er mottatt, basert på start (’a’) og stopp-tegnet (’\n’) som sendes fra dybdemåleren, blir den logget til en liste.

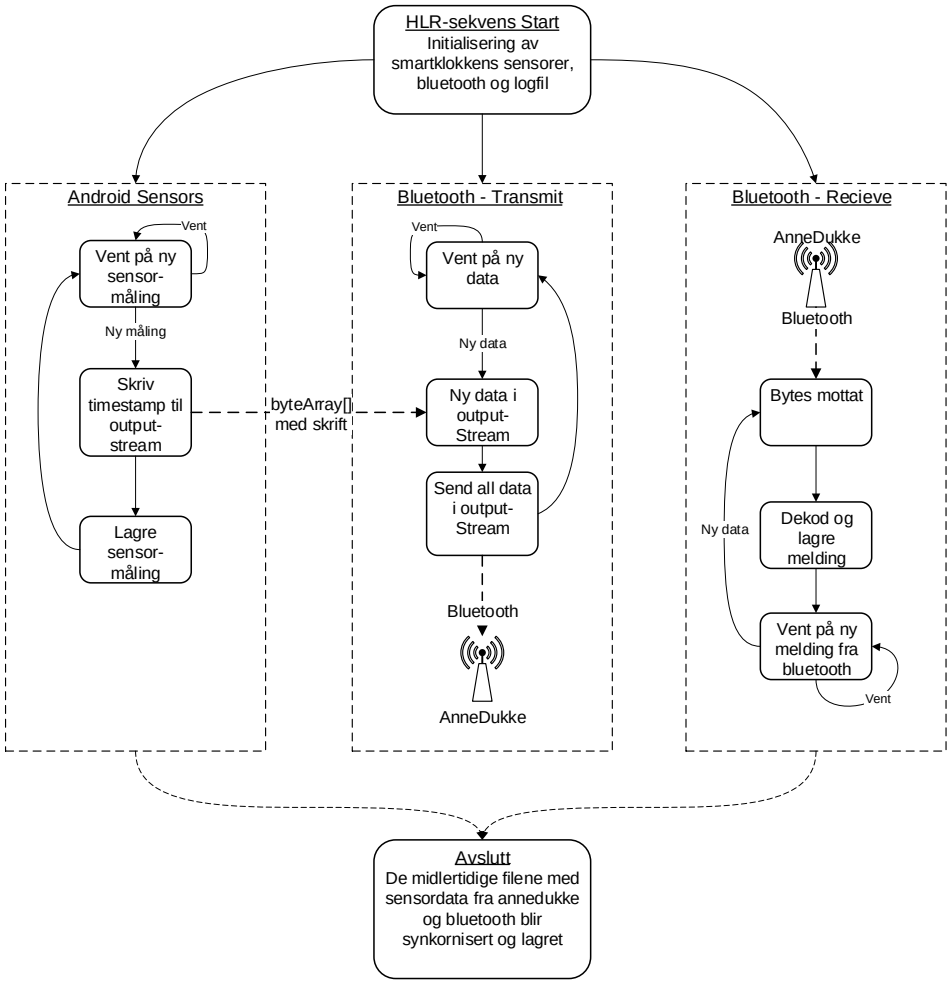
Etter fullført bluetooth-oppkobling kan en logge-sekvens startes ved at ”start”-knappen trykkes inn. Så lenge start-knappen er trykket inne vil smartklokken kontinuerlig lese fra sine interne sensorer. Flyten av data for når applikasjonen er i ”logge-modus” vises i figur 4.4. Hver stipla boks i figuren henviser til hver sin tråd for applikasjonen. En tråd leser kontinuerlig sensorene i smartklokken. Når det kommer en ny akselerasjonsmåling fra sensoren, blir tidspunktet for denne målingen skrevet til outputStreamen for bluetoothtilkoblingen. En tråd i bakgrunnen leser det som er på outputStreamen og videresender dette over bluetooth seriell til dybdemåleren. En tredje tråd ligger uberørt i bakgrunnen og logger alle meldingene smartklokken får i retur fra dybdemåleren i annedukken. Mens applikasjonen er i loggemodus vil sensordataen fra smartklokken samt meldingene den mottar fra bluetooth bli logget til hver sine separate *arrayLists*, forholdsvis *accData* og *btData*. ArrayLists er en Android-implementasjon av dynamiske lister. Det er to gode grunner til at data blir mellomlagret i lister først:

- Android I/O håndtering er ikke alltid like raskt og stabilt, det oppstår problemer om det blir forsøkt å skrive til en fil 100 ganger i sekundet.
- Målingene fra smartklokkens sensorer og målingene fra dybdemåleren må *synkes* i tid.

Etter endt logging vil disse 2 listene, accData og btData, bli synket i tid den grad det er mulig, for å så bli skrevet til en fil som blir lagret internt på smartklokken. Denne filens navn er *”LogFile.txt”*. Når klokken går ut av loggemodus vil smartklokken stoppe å lese fra sensorene sine, og derav stoppe å sende meldinger over bluetooth. Tråden som lytter etter bluetooth-meldinger fortsetter imidlertid en liten stund til, for å sørge for at alle meldinger blir mottatt. Trykkes ”Disconnect” inn vil bluetooth-socketen bli stengt.

Til tross for kompleksiteten, gjør egentlig ikke denne applikasjonen mer enn å kommunisere med dybdemåleren og logge sensordata. Analysen av den målte sensordataen blir gjennomført på PC, og derfor måtte logfilen *(LogFile.txt)* overføres til en PC fra smartklokken. ADB - *Android Seriell Tilkobling* brukes for å overføre filen til stasjonær pc for videre behandling. Dette krever at smartklokken er tilkoblet USB, og var den eneste reele løsningen for å overføre filer fra en android smartklokke til en annen enhet (1). For å henholdsvis hente eller slette denne filen fra smartklokken brukes følgende ADB-kommandoer:

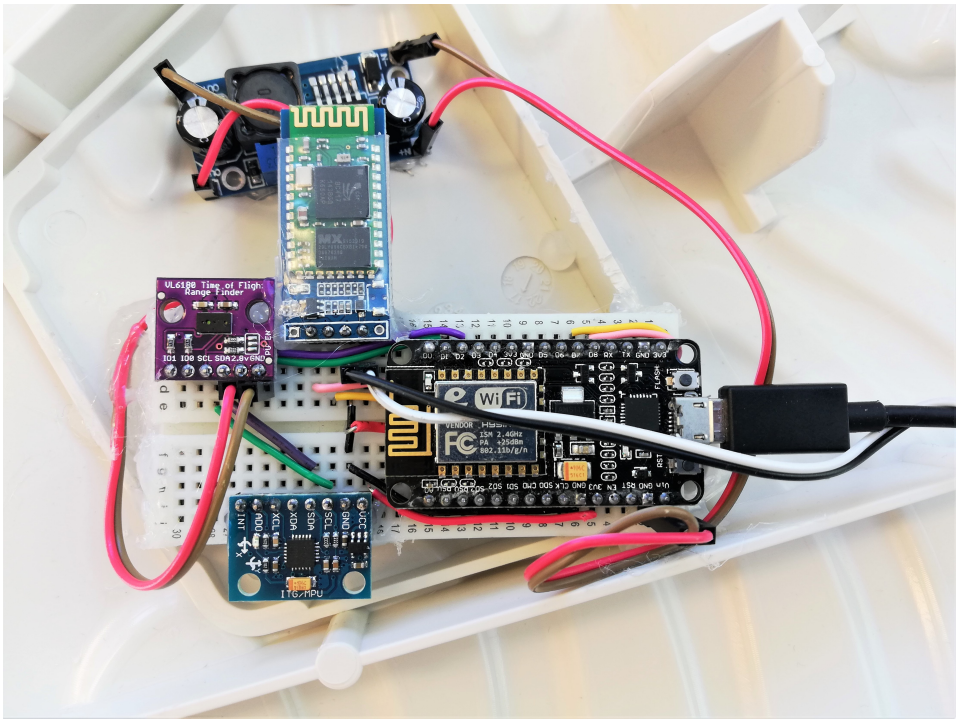
- adb pull /storage/emulated/0/LogFile.txt
- adb shell rm /storage/emulated/0/LogFile.txt



Figur 4.4: Oversikt av hvordan applikasjonens tråder opererer og snakker sammen mens smartklokken er i "logge-modus". En tråd leser sensormålinger og sender klokkeslett for nye målinger til transmit-tråden. Herfra blir klokkeslettene sendt via bluetooth til annedukken. En tredje tråd opererer i bakgrunnen og leser alle meldinger som kommer inn via bluetooth.

4.5 Little Anne Applikasjon

På innsiden av annedukken ble det plassert en mikrokontroller med sensorene beskrevet i kapittel 3. Denne enheten blir også referert som *"dybdemåleren"*. Dybdemåleren ble montert i dukken som vist i bilde 4.5, som er på undersiden av brystplaten til gjenopplivingsdukken. Ettersom måleren ble plassert der, var den i stand til å måle distansen platen blir trykket inn, samt akselerasjonen opplevd av brystplaten.



Figur 4.5: Dybdemåler med NodeMcu og sensorer montert på innsiden av Anne-dukken. Komponenten øverst i bildet er en spenningsregulator for å gi 2.8 volt til distansesensoren. Det hvite/svarte ledningsparet som kommer med usb'en er en separat 5v strømforsyning til bluetooth-modulen

Koden til dybdemåleren ble skrevet i c++, og ble kompilert i PlatformIO-utvidelsen til IDE'en Atom. PlatformIO er en krosskompilator for IOT-enheter som gjør det mulig å benytte seg av biblioteker både rettet mot c++ og for arduino. I tillegg tar programvaren seg av kompilering og opplasting av kode til mikrokontrolleren. Bruk av Platform IO var nyttig, ettersom det gjorde det enkelt å finne og benytte seg av biblioteker for sensorene brukt i oppgaven.

Både distanse-sensoren og akselerometeret kommuniserer med I^2C-protokollen. I^2C består av to ledere og er en *kommunikasjonsbuss*, som under normal drift gjør en *master* (MCU) i stand til å koble til 127 slaver (sensorer) på samme ledningspar. Akselerometeret MPU6050 og distansemåleren VL6180x ble derfor koblet til med samme ledningspar

fra mikrokontrolleren. Kommunikasjon med distansesensoren VL6180x foregår igjennom *sparkfun* sitt open-source bibliotek for Arduino (22). Kommunikasjon med akselerometeret benytter seg av Jeff Rowbergs populære open-source bibliotek *i2cdevlib* (16). Disse to bibliotekene gjør det enkelt å initialisere og kalibrere sensorene og deretter lese målinger fra dem. Sammen med Arduinos bibliotek for seriell tilkobling, utgjør disse basisen for de viktigste funksjonene brukt av mikrokontrolleren. Bluetooth-enheten HC-06 som er tilkoblet har et grensesnitt som gjør den veldig lett å bruke for mikrokontrollere. Enheten kobles til med en seriell port og initialiseres, og det kan deretter skrives og leses *chars* sendt over bluetooth. En char utgjør i dette tilfellet en ASCII-karakter som for eksempel bokstaven 'j' eller tallet '4'. En liste over hovedfunksjonene dybdemåleren består av forklares i tabell 4.2.

Tabell 4.2: De viktigste funksjonene som blir brukt av mikrontrolleren mens den måler og logger kompresjonsdybde over bluetooth.

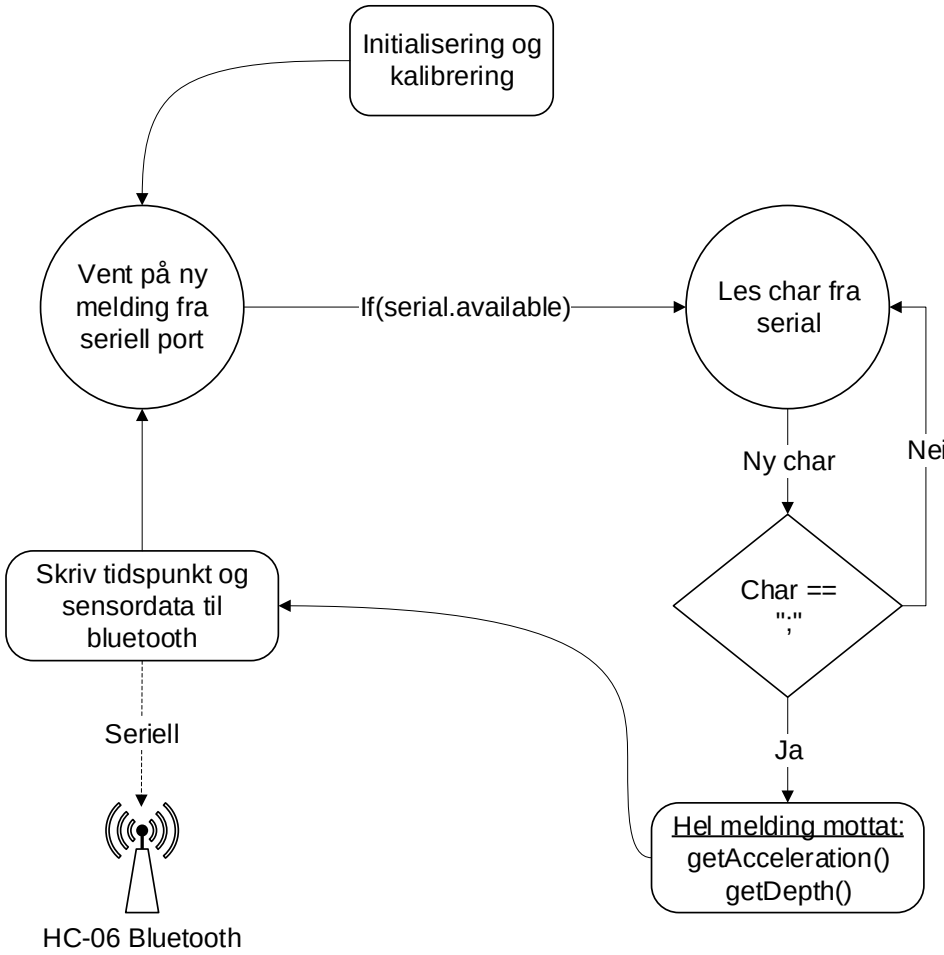
Funksjonsnavn	**Funksjon**
Serial.available()	Returnerer 1 om det finnes uleste chars på seriell-tilkoblingen
Serial.read()	Leser nyeste char fra seriell-tilkoblingen
Serial.print()	Muliggjør skriving av integers eller chars til seriell-tilkoblingen
getRealAcc()	Lagrer siste målinger fra akselerometer og gyro i 6 variabler. Benytter seg av en funkjson fra MPU6050-biblioteket
getDistance()	Returnerer målingen til distanse-sensoren. Funksjonen er fra VL6180-biblioteket

Ettersom NodeMCU er en enkel mikrokontroller uten operativsystem er det ikke mulighet til å ha tråder. Derfor går alt i en sekvensiell løkke, der den kort fortalt leser meldingen den får via bluetooth før den svarer med sensormålingene sine. Mer detaljert vil mikrokontrolleren starte med å gjøre følgende i det den skrues på:

- Initialisere seriell tilkobling til bluetoothmodulen HC-06 med riktig baudrate
- Initialisere akselerometeret og dybdemåleren
- Logge 200 distanse-målinger. Offsetet til distansemåleren blir da gjennomsnittet av disse målingene.

Deretter vil mikrokontrolleren gå inn i en evig løkke som blir fremstilt i figur 4.6. Denne løkken venter kontinuerlig til den får ny data på den serielle tilkoblingen. Ny data på den serielle tilkoblingen betyr at HC-06-modulen mottar et tidspunkt fra smartklokken som har blitt sendt med bluetooth. Karakteren 'a' markerer starten på et nytt tidspunkt. Mikrokontrolleren vil så lese alle verdiene frem til den mottar karakteren *';'*, som betyr at den har mottatt en hel melding fra smartklokken. Da blir de nyeste målingene fra MPU6050 og VL6180 hentet, før de blir skrevet som svar til smartklokken.

VL6180 opererete på 80 hz, men meldingene fra smartklokken ble mottatt og sendt tilbake med 100hz. Derfor ble bare distansemålingene levert tilbake med annen hver melding, som gjorde at VL6180 opererte på 50hz. Disse dybdemålingene ble interpolert i ettertid for å inneholde like mange målinger som de øvrige sensorene i oppgaven. Interpoleringen ble gjort ved å legge til gjennomsnittet av annen hver måling.



Figur 4.6: Oppbyggingen av prosjektet *LittleAnne* til mikrokontrolleren esp8266

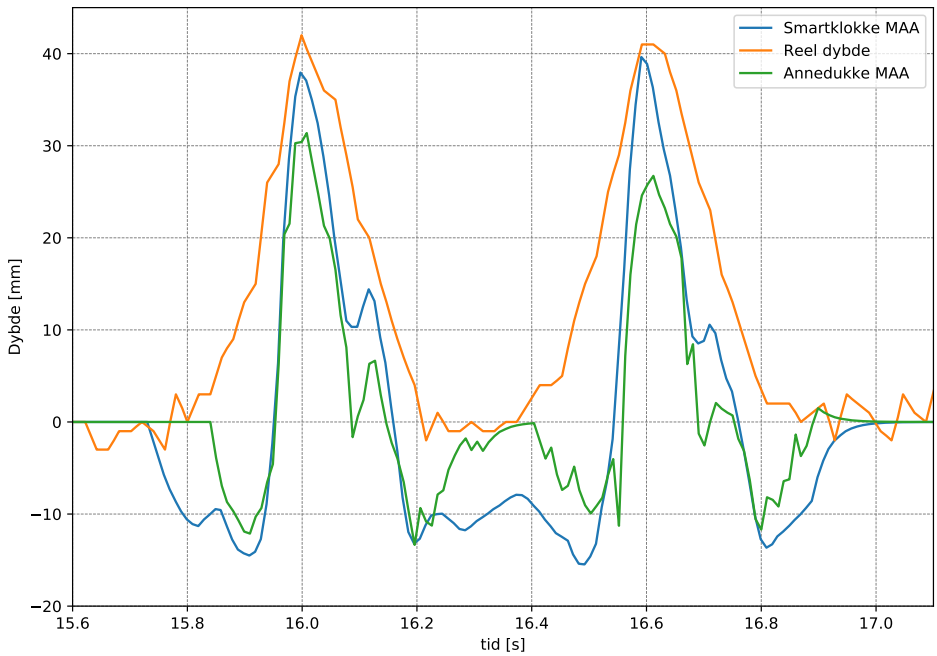
4.5.1 Bluetooth utfordringer

Løsningen med overføring av dybdemålinger ved hjelp av bluetooth bydde imidlertid på noen uforutsette problemer. Mens løsningen ble utviklet viste det seg at man ikke har noen garanti på at alle meldinger blir sendt/mottatt med samme mellomrom fra smartklokken, ikke ulikt utfordringen med at androids akselerometer-målinger ikke kommer med fast tidsintervall. Kommunikasjonsmodulen som brukes i Android-koden er en bakgrunnstråd som kan bli nedprioritert om operativsystemet har viktigere oppgaver, som da medfører at smartklokken plutselig sender ut en "bulk" med meldinger.

Det ble utført en liten test av 800 måle-linjer for å sjekke gjennomsnittlig forsinkelse til meldingene som ble sendt over bluetooth, som målte tiden fra en måling ble tatt på smartklokken til den mottok svar fra annedukken. Her ble det klart at bakgrunnstråden som leser meldinger gjør dette i bulk, hvor 1-15 meldinger blir mottatt samtidig. Den siste mottatte meldingen av en bulk hadde gjennomsnittlig $\sim$120ms forsinkelse. De resterende meldingene for bulken akkumulerte 10ms mer forsinkelse jo eldre de var. Den eldste meldingen i en bulk på 13 målinger fikk dermerd $\sim$120+13x10ms = 250ms forsinkelse. Det er imidlertid veldig vanskelig å vite hvor denne forsinkelsen kommer fra. Følgende situasjoner skaper forsinkelse:

- Tiden fra en måling blir gjort av smartklokkens sensor til android videresender verdien
- Forsinkelsen før inputstreamen blir skrevet til og androids bluetooth-modul videresender dette
- Kommunikasjon mellom MCU'en og dens bluetooth-modul
- Forsinkelsen fra meldingen blir mottat av smartklokken til android håndterer den

Det er vanskelig å vite nøyaktig hvor forsinkelsen oppstår, så derfor ble akselerometermålinger fra mikrokontrolleren sammenlignet med målingene fra smartklokken. Her ble det utført et par tester hvor smartklokken ble lagt fast på brystet til annedukken, før en kompresjon ble gjort. Ettersom annedukkens akselerometer og smartklokkens akselerometer er i samme stilling og blir påført samme kraft, burde de oppleve samme akselerasjon. Ved å sammenligne akselerasjonsmålingene så det ut til at målingene fra mikrokontrolleren normalt lå 30ms bak målingene til smartklokken. Denne forsinkelsen fremstod til å være lik for alle måleserier. For å kompensere for denne tidsforskyvningen, ble alle målinger fra mikrokontrolleren flyttet "3 målinger tidligere". Resultatet etter synkroniseringen i tid sees i figur 4.7. Her fremstår det som at akselerasjonsmålingene er tilnærmet perfekt synkronisert.



Figur 4.7: Sammenligningen av magnituden av akselerasjon målt i annedukken og i smartklokken, etter justering av tidsforskyvning.

Bruk av denne tidsforskyvningen på 3 målinger fungerte stabilt og ga ingen problemer i resten av oppgaven, men ettersom det er en del usikkerhet rundt hvor forsinkelsen oppstår er det ikke mulig å vite *helt nøyaktig* når en måling ble utført. Dette ble imidlertid sett bortifra i resten av arbeidet og de små tidsforskjellene som muligens kan oppstå for noen målinger ble neglisjert da det trolig ikke har noe som helst å si for resultatet.

4.6 Arbeidsmetode

Måledataen i logfilen fra smartklokken trenger prosessering før den kan brukes. For å analysere kompresjoner med *multivariate data analysis* og for å sjekke nøyaktigheten til integreringsteknikker var det nødvendig å separere målingene for *hver kompresjon*. Dette ble oppnådd ved at hver kompresjon ble opprettet som en eget objekt av en kompresjon-klasse. Først blir logfilen som inneholder all data fra dybdemåleren og smartklokken lastet opp av et skript i Python. Her blir all sensordataen fra logfilen lastet inn som egne variabler i en klasse. Resultatet vises i tabell 4.3. Disse variablene er lister av alle de forskjellige målingene som blir lagret på smartklokken. Lengden av alle disse listene er den samme, og tilsvarer antall linjer i logfilen. Dette gjør det mulig å grafisk fremstille all mulig informasjon om sensor-målingene ettersom en har tilgjengelig målinger (mag/x/y/z/gx...) og tilsvarende tidspunkt for målingene. Denne informasjonen blir lagret i en variabel kalt AccData.

Tabell 4.3: Medlemsvariabler av klassen *AccData*. Alle elementene er lister av lengde som tilsvarer antall målinger fra smartklokkens logfil. De under nederste strek er målingener fra dybdemåleren, mens de over er fra smartklokken.

Egenskaper	**Beskrivelse**
dateInputMilliSec	Inneholder alle tidspunkt i unix-tid
time	Inneholder alle tidspunkt som klokkeslett
time_sec	Inneholder millisekund siden første måling
x, y, z	3 Lister med alle akselerasjons-målinger i x, y, og z retning
gx, gy, gz	3 Lister med alle gyrometer-målinger i x, y, og z retning
mag	Inneholder magnituden av akselerasjon for hvert tids-steg
vel_HP	Liste av *fart*: *mag* integrert og høypass-filtrert
pos_HP	Liste av *dybde*: *vel_HP* integrert og høypass-filtrert
mm	Liste av alle dybdemålinger fra Annedukken
x_a, y_a, z_a	3 Lister som inneholder alle akselerasjons-målinger i x, y, og z retning fra Annedukken

Å lage disse kompresjonsobjektene ble utført ved å iterere igjennom *AccData* og isolere hver kompresjon. Hvor en kompresjon starter og stopper blir basert på posisjons-estimatene fra det interne akselerometeret i smartklokken. Hver kompresjon vet hvor lang den er, når den startet og inneholder all måledata for perioden den varte. Kompresjonen får også dybde-estimater samt referansedybden målt av sensoren i annedukken. Under testing av dybde-estimerings-teknikker ble det derfor iterert over alle kompresjons-objektene for å optimere for det som ga minst totalt avvik.

Senere i oppgaven blir Multivariat Data-analyse brukt. Multivariat data-analyse er en analysemetode man kan benytte for å oppdage mønstre i store datasett. I tilfellet i denne oppgaven med HLR, gir det mening å sammenligne alle kompresjonene i treningsettet. For å kunne sammenligne kompresjonene med hverandre, må hver kompresjon ha akkurat like mange variabler tilknyttet seg. Kompresjonene er av ulik lengde, derfor gir det ikke mening å ha med tids-seriene for målingene. Hvordan kan man sammenligne første måling i y-akselerasjon mellom 2 kompresjoner? Det gir ikke noe mening, og derfor må all mulig relevant data om en kompresjon ekstraheres til å bli representert av skalare variabler som kan kalkuleres for alle kompresjoner. Dette kan bli gjort ved å for eksempel ta i bruk *delta-variabler*. Delta variablene representerer forskjellen mellom høyste og laveste utslag for en gitt måling. Slike variabler er lette å sammenligne mellom kompresjoner. I analysen finnes det muligens mønstre hvor alle de kompresjonene med høyt utslag (delta) i akselerasjon er dype kompresjoner. Et annet alternativ kan være å summere summen mellom alle målingene for hver kompresjon eller finne gjennomsnittet. Å velge disse variablene for analyse er ingen fasit, men de som blir kalkulert her er det som undertegnede tenkte kunne ha relevans for kompresjonsdybden. I tabell 4.4 forklares variablene som ble kalkulert for hver kompresjon som senere blir brukt i *Principle Component Analysis*.

Tabell 4.4: Utdrag av variablene som ble laget for hver kompresjon. Disse variablene blir senere brukt til PCA

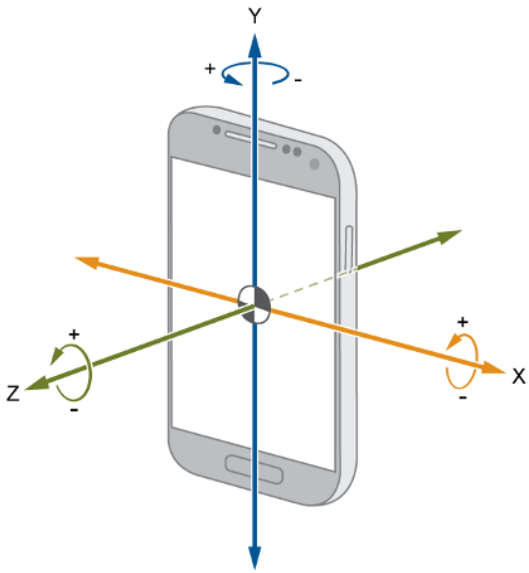
Egenskaper	**Beskrivelse**
DateInputMS	Navnet for hver kompresjon - starttidspunkt i Unix tid
mm_real	Dybdemålingen fra annedukken - referansemåling
duration	Varighet til kompresjon i ms
num_elements	Antall elementer i kompresjonen
mm_rise	Dybde-estimat basert på MAA stigende flanke
mm_fall	Dybde-estimat basert på MAA synkende flanke
mm_avg	Dybde-estimat basert på gjennomsnitt av stigende og synkende flanke
a_mag_delta	Absoluttverdien av utslaget til MAA (MAA_max - MAA_min)
a_x_delta	Absoluttverdien av utslaget til akselerasjon i x retning
a_y_delta	Absoluttverdien av utslaget til akselerasjon i y retning
a_z_delta	Absoluttverdien av utslaget til akselerasjon i z- retning
g_x_delta	Absoluttverdien av utslaget til gyrometer i x retning
g_y_delta	Absoluttverdien av utslaget til gyrometer i y retning
g_z_delta	Absoluttverdien av utslaget til gyrometer i z retning
g_x_int	Integralet av gyrometermålinger i x retning
g_y_int	Integralet av gyrometermålinger i y retning
g_z_int	Integralet av gyrometermålinger i z retning
mm_rise_SE	Avvik mellom mm_rise og referanse (mm_real)
mm_fall_SE	Avvik mellom mm_fall og referanse (mm_real)
mm_avg_SE	Avvik mellom mm_avg og referanse (mm_real)
clip	Antall ganger smartklokkens akselerometer har nådd maksverdi

KAPITTEL 5

Teoretisk Analyse

Den simpleste måten for estimering av posisjon med de sensorene som finnes i en smartklokke oppnås ved å dobbeltintegrere akselerasjon, som gir posisjon som resultat. Dette medfører allikevel med seg et par problemer for praktisk implementasjon. *Hvilken* akselerasjon skal integreres for å få et nøyaktig resultat? Smartklokken gir akselerasjon i 3 retninger, men det er åpenbart at det hadde vært lettere å bare forholde seg til ett enkelt akselerasjons-signal i en dimensjon. Hva slags integreringsteknikk og filtrering burde så benyttes for dette akselerasjons-signalet? Disse problemstillingene blir forsøkt svart i dette kapittelet.

5.1 Dybdeestimering



Figur 5.1: Kordinatsystemet relativt for sensorene på Android-enheter (23).

Akselerometeret i smartklokken gir akselerasjon i x, y og z-retning gitt i et koordinatsystem basert på sitt eget legeme som referansesystem, som vist i figur 5.1. Gravitasjonskraften vil alltid virke på legemet og smartklokken vil bli utsatt for den kraften uansett hvilken orientering klokken har. Ligger smartklokken i ro med skjermen opp, vil smartklokkens referansesystem være det samme som jordens, og smartklokken vil måle 9.81 m/s^2 i z-retning. Endres orienteringen til klokken, vil gravitasjonskraften bli fordelt over x, y, og z retning, men den totale akselerasjonen smartklokken utsettes for er fremdeles lik gravitasjonskraften. Dette må ta hensyns til i denne oppgaven ettersom smartklokken vil ha en vilkårlig orientering basert på hvordan den er plassert på et håndledd og om den er på høyre eller venstre hånd.

Før dybden kan estimeres, må det det bli gjort en forutsetning på forholdene HLR blir utført i. To muligheter for å redusere smartklokkens akselerasjon til en dimensjon er:

- Basere seg på at pasienten ligger ortogonalt med gravitasjonskraften, og dermed prøve å isolere kraften klokken måler diagonalt med gravitasjonskraften.
- Kalkulere *magnituden av akselerasjon* (forkortet MAA), som vil si at man ser på den totale akselerasjonen smartklokken utsettes for og ser bort i fra retning.

Smartklokkens bevegelse på håndleddet under en kompresjon er relativt forutsigbar, men avhenger likevel av teknikken kompresjonen utføres med samt personen som utfører kompresjonen. Utføres kompresjonen i en litt merkelig vinkel, kan håndleddet bevege seg en lengre strekning enn brystkassen som blir komprimert. Med andre ord er det ikke mulig å

garantere at smartklokken plassert på håndleddet til en person beveger seg i *samme retning* og derfor ikke like langt som brystkassen til pasienten. Basert på vinkelen kompresjonen utføres med i forhold til brystkassen, vil posisjonen klokken forflytter seg alltid være like stor eller større enn posisjonsendringen til brystkassen. I tilfellene hvor kompresjoner blir utført med en vinkel vil følgelig avstanden estimert basert på den totale akselerasjonen være større enn om kompresjonen hadde blitt utført ortogonalt på pasientens brystkasse. Dette problemet oppstår når posisjonsforandring blir estimert basert på *magnituden av akselerasjon*, ettersom det da bare blir sett på den totale posisjonen smartklokken beveger seg uavhengig av retning. Fordelen med å bruke magnituden av akselerasjon er i imidlertid at om HLR utføres presist og ortogonalt med brystkassen vil den fungere uansett i hvilken stilling pasienten ligger. Dette er i tilegg en av de matematisk enkleste metodene for posisjonsestimering med integrasjon og er derav enklest å implementere.

En annen løsning enn å bruke magnituden av akselerasjon er å isolere kraften som utøves i jordens z-retning. Dette kan bli estimert ved å regne ut rotasjonen og vinkelen for hvert tidssteg under en kompresjon. Er retningen til smartklokken tilgjengelig er det mulig å regne ut hvor stor andel av kraften som tilsvarer hver akse i jordens referansesystem. Dette kan implementeres med *fusion-filtre*, altså filtre som baserer seg på flere sensorer. Slike filtre benytter seg gjerne av akselerometeret, gyrometeret og et magnometer om tilgjengelig for å raskest og nøyaktigst mulig beskrive rotasjonen til en enhet. En antagelse som må gjøres før det blir forsøkt å isolere kraften langs gravitasjonsretningen er at personen alltid ligger helt flatt, altså ortogonal med gravitasjons-retningen. Om personen ligger i en slak oppoverbakke, vil den opplevde posisjonsendringen langs gravitasjonsaksen være større enn det som blir påført personen.

5.2 Integrasjon

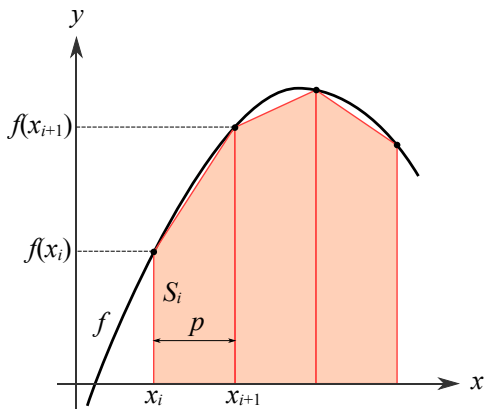
Akselerasjon integrert to ganger resulterer i posisjon. Dette blir fremstilt i ligning 5.2. I ligningen tilsvarer s=posisjon, v=fart, a=akselerasjon, t=tid og Δ endring.

$$v = \frac{\Delta s}{\Delta t}, \qquad a = \frac{\Delta v}{\Delta t} \tag{5.1}$$

$$a = \frac{\Delta^2 s}{\Delta^2 t^2} = \frac{d}{dt}\frac{d}{dt}s \qquad \rightarrow \qquad s = \int_{t_0}^{t}\int_{t_0}^{t} a \tag{5.2}$$

I ligningene over er integrasjon gitt for kontinuerlige signaler. Målingene fra sensorer er imidlertid diskrete og kan ikke integreres analytisk ved mindre det først utføres en form for *kurvetilpassing*. For diskrete signaler blir integrering omtrent det samme som å kumulativt addere alle målinger multiplisert med tidsteget mellom hver måling. I denne oppgaven benyttes trapesmetoden for integrasjon, fremstilt i figur 5.2 og gitt i ligning 5.3. Dette ble implementert i python.

$$\int f(x)dx \approx \sum_{i=1}^{n} \frac{x_{i-1} * x_i}{2} * \Delta x \tag{5.3}$$



Figur 5.2: Illustrasjon av trapez-metoden for integrasjon (24).

5.3 Magnitude av akselerasjon

Den enkleste metoden for å løse dimensjonsproblemet med akselerometermålinger i tre retninger er å kalkulere *magnitude av akselerasjon*, vist i formel 5.4. MAA er den totale akselerasjonen et legeme utsettes for, og ser bort ifra retning. For enkelhetens skyld blir også gravitasjonskraften trukket fra, slik at magnituden av akselerasjon blir lik null når smartklokken ligger helt i ro. Prinsippet her å neglisjere retningen akselerasjonen blir utført i og samtidig gi et en-dimensjonalt akselerasjons-signal som er enkelt å forholde seg til.

$$Akselerasjon_{magnitude} = \sqrt{Acc_x^2 + Acc_y^2 + Acc_z^2} - Gravitasjon \tag{5.4}$$

Merk at magnituden til akselerasjonen er naturlig disponert til å være positiv. Den største teoretiske negative akselerasjonen som MAA kan ha er -9,81. Dette oppnås når smartklokken er i fritt fall, og hvor smartklokkens akselerasjon i x-, y- og z-retning er null. Sett fra jordens referansesystem akselereres klokken med 1g langs tyngdekraften. Beveges smartklokken parallelt med jordens gravitasjonskraft under 1g blir altså MAA negativ. Er akselerasjonen større enn dette vil MAA bevege seg mot null igjen. MAA vil bli 0 når smartklokken ligger helt i ro, og når den beveger seg langs gravitasjonsretningen med 2g.

Gitt en situasjon hvor smartklokken er i fritt fall, har den en akselerasjon på 1g (9.81 m/s^2) fra jordens synspunkt mens klokkens MAA blir $0m/s^2$. Beveges smartklokken med $1g + 3 = 12.81m/s^2$ eller $1g - 3 = 6.81m/s^2$ parallelt med gravitasjonskraften vil MAA bli $3m/s^2$ i begge tilfellene. Brystkompresjoner med kraftig akselerasjon kan på grunn av dette bli tolket som svakere enn de egentlig er. Dette er åpenbart ikke ønskelig, men er ikke så lett å unngå med posisjonsestimering basert på MAA. Senere blir det utforsket om det går an å forbedre estimatene basert på MAA ved hjelp av informasjon fra gyroskoper i smartklokken. Dette blir studert med PCA-analyse i kapittel 6.6.

5.4 Akselerasjon: Sensor fusion

Et alternativ til magnituden av akselerasjon, er å isolere kraften i Z-retning ved å kalkulere orienteringen til smartklokken. Dette kan bli estimert med et *fusion filter* som baserer seg på målinger fra mer enn en sensor. Å estimere *heading* til et legeme basert på akselerometer og gyrometer er mye forsket på, men det er gjerne vanskelige å implementere (25). Begge sensorene er plaget av drift, og når dataen integreres to ganger er det vanskelig å vite om man bare måler støy eller om man faktisk finner retningen til enheten. Noe som vanskeliggjør dette problemet er at man ikke har noe referansepunkt. Et quadcopter som skal stå helt stille eller en båt som ligger i ro har gjerne akselerometrene sine plassert helt i vater. I de situasjonene kan et filter vite at z-retning skal være det samme som gravitasjonskraften, 9.81, og man kan da bruke dette som referanse for å regne ut de andre vinklene. Et magnetometer kan bidra til å øke nøyaktigheten på slike filtre. Mangetometeret måler retningen mot jordens geomagnetiske poler, men dette er sårbart mot plutselige elektromagnetiske lokale endringer, samt at det ikke er et raskt instrument. For at et sensor-fusion filter skal være til nytte for dette prosjektet må det i tilegg ha relativt rask respons og operere rundt 100hz for å få tilstrekkelig mengde målinger. Dette kan implementeres med et kalmanfilter eller et komplementærfilter. På grunn av at MAA er enklere å implementere, fikk det høyest prioritet i denne oppgaven.

Android har en innebygd ”sensor” hvor orienteringen til enheten har blitt kalkulert basert på gyroskopet og akselerasjon. Sensoren *TYPE_ROTATION_VECTOR* gir en rotasjonsmatrise som gir vektorkomponentene for MAA for x- y- og z-retning. Orienteringen i z-retning kunne blitt brukt for å isolere akselerasjonen smartklokken ble utsatt for langs gravitasjonsretningen. Dette ble testet på smartklokken, men responsen var for treg til å brukes i denne oppgaven. Ettersom undertegnede ikke fikk til å justere dette, måtte et komplementærfilter blitt implementert selv for å orientering, noe det ikke ble tid til (26).

5.5 Filtrering av signal

Akselerasjonsmålinger er sårbare for støy som medfører at magnituden av akselerasjon gir et nokså fluktuerende og støyete signal. Derfor er det hensiktsmessig å lavpassfiltrere dette signalet for å fjerne mesteparten av den høyfrekvente støyen. HLR utføres omtrent aldri over 4hz (240 slag i minuttet), og et ideelt analogt filter kunne vært brukt for å kutte ut alle frekvensene over dette. I denne oppgaven brukes det imidlertid bare et enkelt lineært diskret lavpassfilter, som viste seg til å gi tilfredstillende raske og stabile resultater basert på arbeidet i prosjektoppgaven (1). Formelen for dette lavpassfilteret er gitt i 5.5 hvor x er den filtrerte målingen, mens y er målingen fra sensoren.

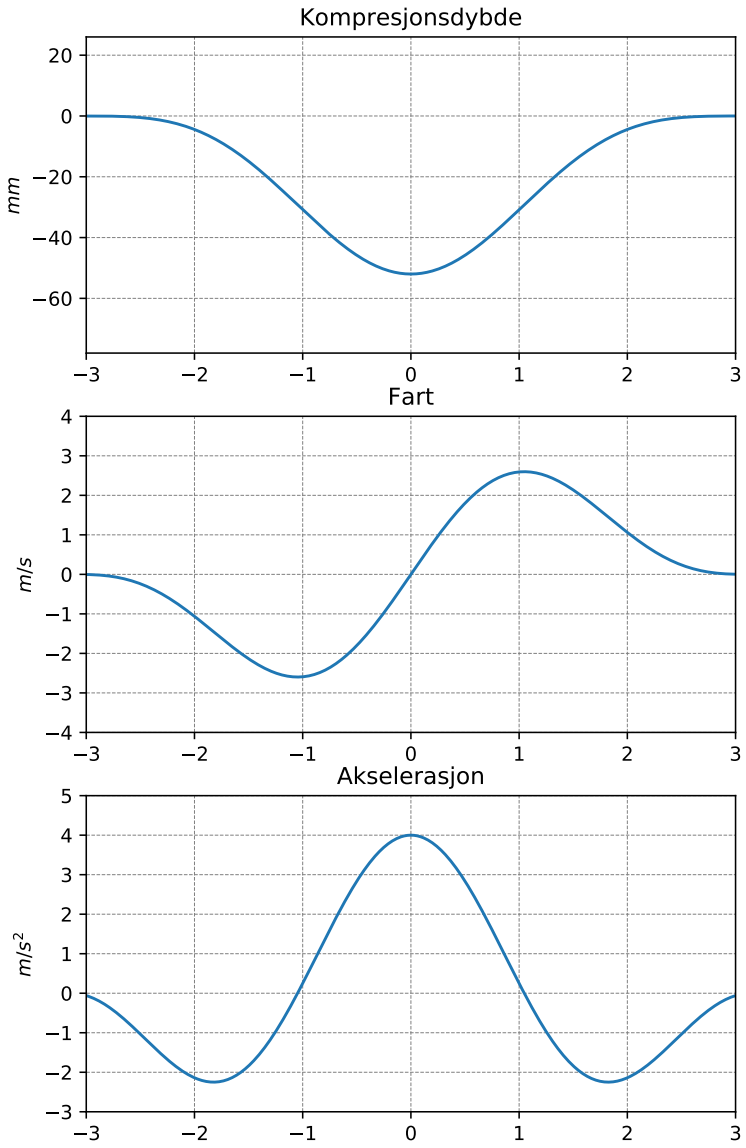
$$x = k * y + (k - 1) * x_{forrige} \tag{5.5}$$

Høypassfiltrering kreves senere i oppgaven i det magnituden av akselerasjon blir integrert. Sensorene i smartklokken er ikke perfekte, og de drifter litt over tid. Når denne driften blir integrert opp blir det resulterende signalet uhåndterlig. Dette løses ved å fjerne de mest saktegående endringene som driften består av. I formelen 5.6 er x det høypassfiltrerte signalet basert på målingene y.

$$x = k * (x_{forrige} + y_{nå} - y_{forrige}) \tag{5.6}$$

5.6 Isolert kompresjon

For å få en bedre forståelse av hva slags signal som jobbes med, blir dataen av en isolert kompresjon utforsket i dette delkapittelet. Hva er sammenhengen mellom magnitude av akselerasjon, og forflytning av annedukkens brystkasse?

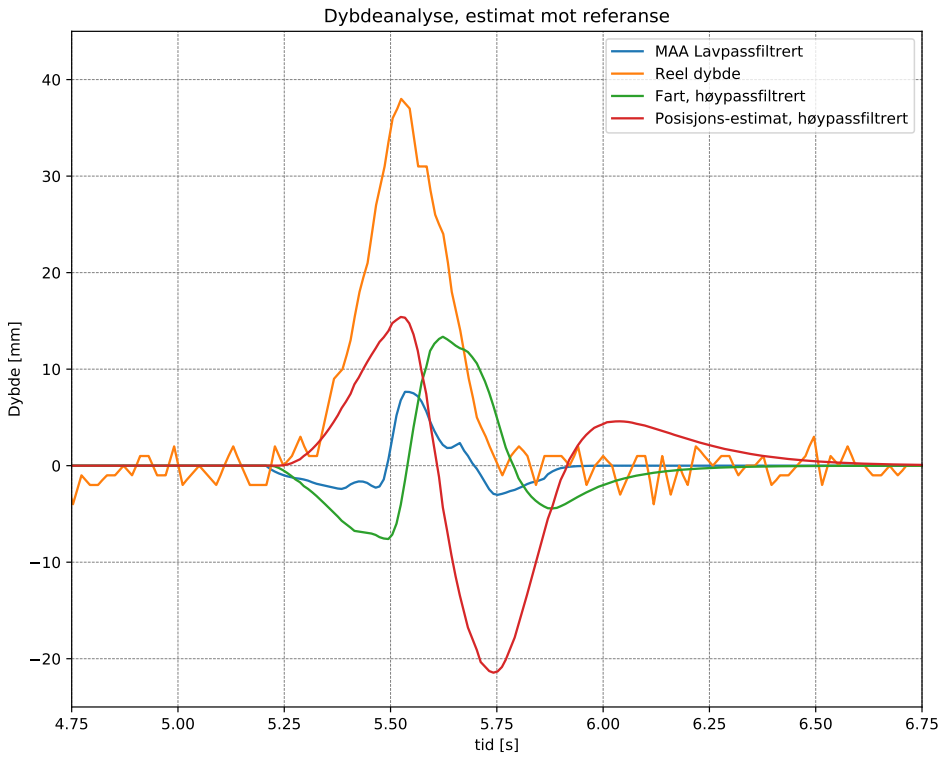


Figur 5.3: Teoretisk antagelse av posisjonsforflytningen til brystkassen under en hjertekompresjon, som også har blitt derivert to ganger.

5.6.1 Antagelse: Forskyvning av brystkasse under kompresjon

Øverst i figur 5.3 er det foretatt en analytisk antakelse av hvordan posisjonen for en brystkasse beveger seg under en normal brystkompresjon. Gitt at brystkassen starter i posisjon 0 under en kompresjon, vil brystkassen forflytte seg ca $50mm$ i det man nedpresser brystet før presset slutter og kompresjonen er ferdig. Her er antas at både brystkassen, håndleddene og derav smartklokken man har på seg har tilnærmet lik posisjonsforflytning under en kompresjon. Disse grafene antar dog en at kompresjonen av brystkassen bare foregår i en akse, altså diagonalt med gravitasjonskraften (*rett ned)*. Som diskutert i kapittel 5.3 vil magnituden av akselerasjon bare ha en akse, men dette signalet vil kunne være større om man har en vinkel under kompresjonen som ikke er diagonal med gravitasjonskraften. Nederst i figur 5.3 er posisjonforskyvningen av en kompresjon blitt derivert to ganger. Forhåpentligvis vil magnituden av akselerasjon fra smartklokken være lignende dette, ettersom det er en tilnærmet lik bevegelse som blir utført.

5.6.2 Praksis: Akselerasjon fra smartklokke under kompresjon



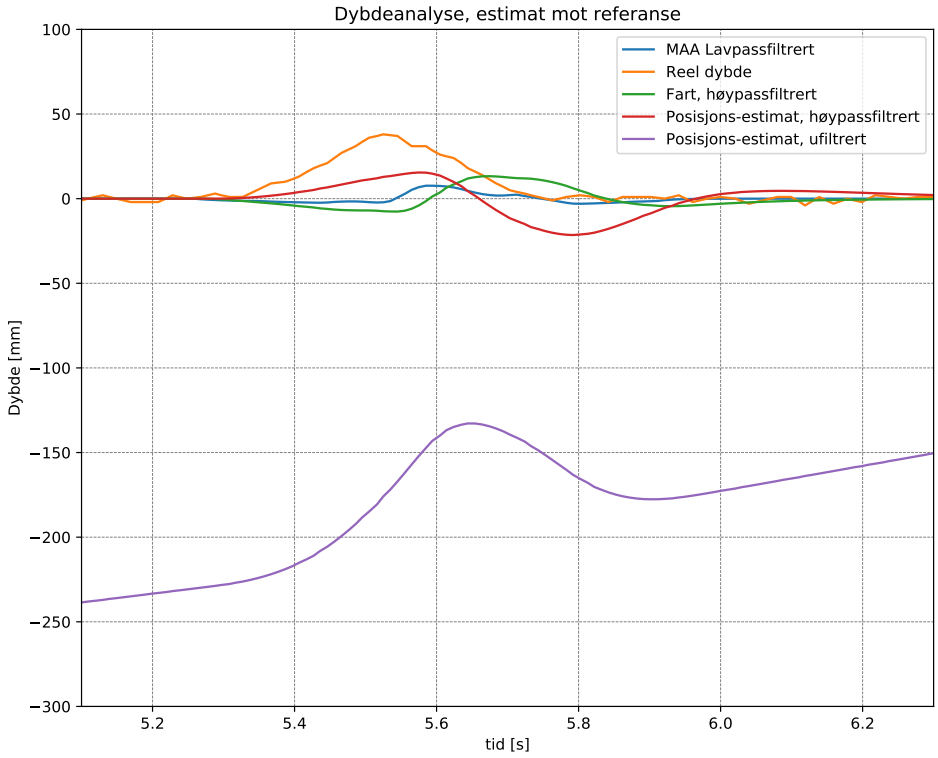
Figur 5.4: Magnitude av akselerasjon fra smartklokke integrert to ganger og skalert opp. I tilegg er dette sammenlignet med den reele dybden målt i annedukken

Figur 5.4 fremstiller akselerasjon, fart og dybde for en isolert kompresjon utført på annedukken. Magnituden av akselerasjon er tatt fra smartklokken, og har blitt integrert to ganger for å få et posisjons-estimat. Begge de integrerte signalene er her høypass-filtrert. Den "Reele dybden" er data hentet fra sensoren i annedukken. Avlesing av grafen viser at den reele dybden for kompresjonen var 38mm dyp og varte i ca i et halvt sekund.

Grafen til magnituden av akselerasjon er som antatt i kapittel 5.6.1 ganske likt den nederste grafen i figur 5.3. Det vil si at MAA som klokken blir utstatt for under en kompresjon stemmer overens med det som ble antatt teoretisk. Håpet var at ettersom akselerasjonen er relativt lik i teori og praksis, så ville det integrerte posisjons-estimatet fra MAA bli omtrent likt som den simulerte posisjonen øverst i figur 5.3. Dette viste seg ikke å stemme i praksis, da det estimerte posisjonssignalet basert på integrasjon oppfører seg litt annerledes. Posisjonsestimatet har tross alt blitt integrert og høypassfiltrert to ganger, så det var muligens ikke overaskende at det har annerledes respons enn teoretisk antatt.

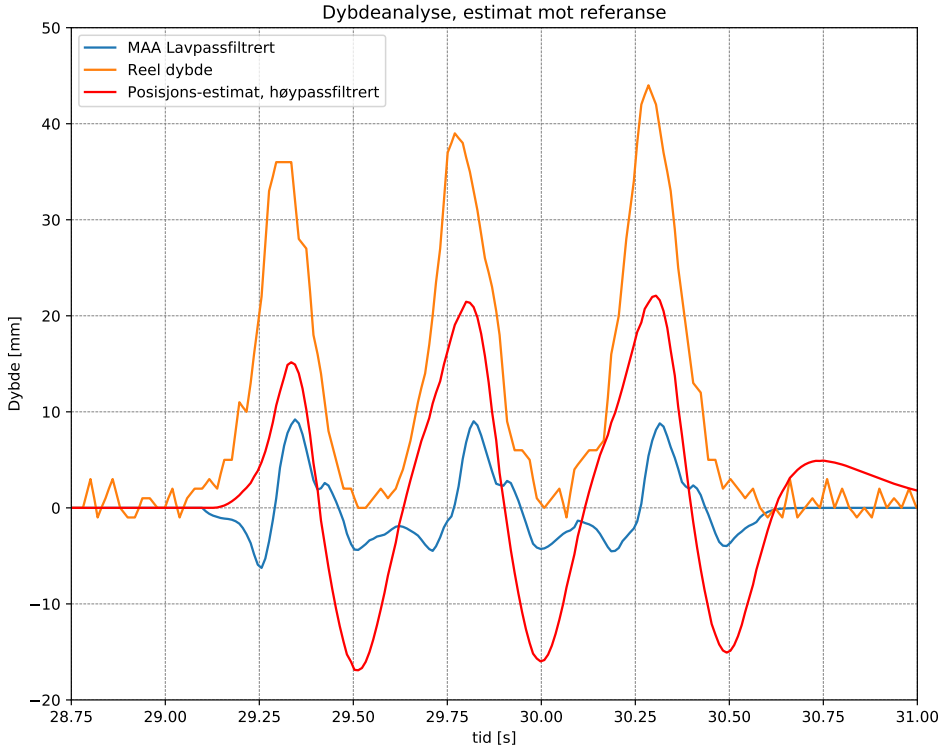
I figur 5.5 vises posisjonsestimatet basert på MAA fra smartklokken uten høypass-filtrering. Sett bortifra den åpenbare driften, er det tydelig at posisjons-estimatet ligner mer på den teoretiske antagelsen gjort i 5.3, bare at signalet er invertert. Posisjons-estimatet uten høypassfiltrering er imidlertid tilnærmet ubrukelig. I løpet av den 35 sekund lange målesekvensen som kompresjonen fra figur 5.5 er fra, drifter posisjonsestimatet til over 7000 mm. En grunn til driften kan også skyldes på grunn av at MAA for en kompresjon normalt har et større positivt enn negativt areal. Dette kommer som en følge av magnituden er naturlig disponert for å være positiv enn negativ som forklart i 5.3. Teoretisk skulle dette arealet blitt 0 ettersom håndleddet startet og ender i samme posisjon.

Likevel, så er posisjonsestimatet nyttig så lenge det blir filtrert riktig. Posisjonsestimatets bunnpunkt i figur 5.4 er på ca -22mm, og toppunktet er 15mm. Høydeforskjellen tilsvarer da 37mm, ikke veldig ulikt den reele dybden på 38mm. Selv om posisjonssignalet etter filtrering ikke gir like mye logisk mening er det fremdeles brukbart som et estimat for kompresjonsdybde, ettersom essensen av informasjonen i signalet forblir det samme. I resten av oppgaven blir derfor dybdeestimatene for en kompresjon basert på avstanden mellom topp og bunnpunktene for posisjons-grafen til en kompresjon.



Figur 5.5: Akselerasjon fra smartklokke integrert to ganger og skalert opp. I tilegg er dette sammenlignet med den reele dybden målt i annedukke, samt hvordan posisjons-estimatet blir uten høypassfiltrering

5.7 Stig eller falltid



Figur 5.6: Posisjonen estimert fra smartklokkens akselerometer sammenlignet med dybden målt i annedukken for tre kompresjoner.

Under en HLR-sekvens kommer kompresjonene kontinuerlig. Hvor en kompresjon ender og neste starter er vanskelig å determinere. Kompresjonen i midten av figur 5.6 har åpenbart et toppunkt på posisjonsgrafen, rundt $t = 29.8$. Men, det er to bunnpunkt tilhørende den kompresjonen. Blir det mest riktig å kalkulere posisjonsestimatet for en kompresjon basert på bunnpunktet fra den stigende eller den fallende flanken? Nøyaktigheten av begge deler blir utforsket i kapittel 6.2.

KAPITTEL 6

Resultater og Diskusjon

I dette kapittelet blir diverse dybde-estimerings-algoritmer laget og undersøkt. Alle algoritmene tar utgangspunkt i det samme treningsettet. Dette datasettet består av kompresjoner utført av flere personer og inneholder totalt over 2000 kompresjoner fordelt over flere HLR-sekvenser. Sekvensene har blitt utført ved forskjellige tidspunkter på døgnet, med varierende lysforhold og temperatur. Dette påvirker sensorene i både smartklokken og annedukken, samt at kompresjonene har blitt utført av flere mennesker med forskjellig teknikk for HLR. Det antas allikevel at målingene er representative for virkelighetstro situasjoner. I tilegg antas det at at hver kompresjon er stokastisk, altså at målingene og dybden estimert for forrige kompresjon ikke har noe å si for neste kompresjon. Dette er kanskje en sannhet med modifikasjoner, ettersom integralene av akselerasjon og gyroskopmålinger går over tid og vil til dels påvirke målingene for neste kompresjon. Statistikk for datasettet vises i tabell 6.1. Bluetooth-kommunikasjonen mellom mikrokontrolleren og smartklokken viste seg å være stabil og robust, da bare 17 av 98725 linjer var ufullstendige (0.02%).

Nettopp det faktum at målingene ikke blir utført i 100% kontrollerte omstendigheter og av forskjellige mennesker burde forhindre *over-fitting* av estimerings-algoritmer. Hadde all testing blitt utført av en robot i kontrollerte omgivelser burde det være mulig å oppnå mye mer nøyaktige dybde-estimater. Men, da hadde man muligens endt opp med en algoritme som bare var nøyaktig når den ble brukt av roboter. Målet med en HLR-assistent på smartklokke er at den skal kunne brukes av *alle*. Ettersom omstendighetene og testforholdene er varierte i disse eksperimentene vil trolig de foreslåtte modellene være mer realistiske for bruk i det virkelige liv. I tilegg har alle disse uoverensstemmelsene lite å si ved sammenligning av dybde-estimerings-teknikkene mot hverandre, ettersom alle algo-

ritmene tar utgangspunkt i det samme datasettet. For hver kompresjon blir variablene gitt i tabell 4.4 kalkulert basert på målingene fra annedukken og smartklokken. For presisering, alle estimatene av dybden er *utelukkende* basert på smartklokkens sensorer, som prøver å estimere "referansen/ekte dybden/mm_real" gitt av dybdemåleren i annedukken.

Tabell 6.1: Statistikk for treningsettet brukt for å bygge opp algoritmer for dybde-estimering

Egenskap	**Beskrivelse**
Størrelse	98725 linjer - totalt 1.1 million målinger
Ufullstendige målinger	17
Antall kompresjoner	2126
- Dybde gjennomsnitt	42mm
- Dybde median	40mm
- Rekkevidde	16mm-72mm
- outliers fjernet	22
Generelt	Datasettet er satt sammen av totalt 11 måleserier utført av 5 forskjellige personer
Frekvens	Snitt 0.42 sekund per kompresjon $\sim$142 BPM / 2.4hz

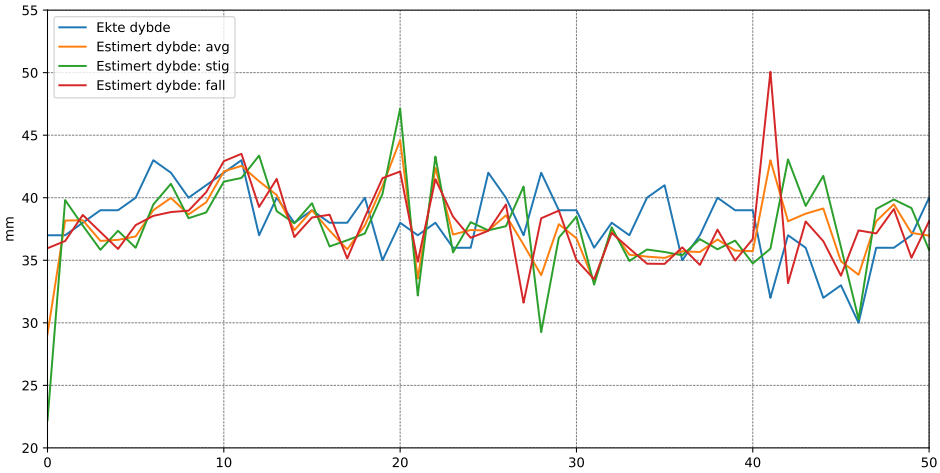
Før hver HLR-sekvens startes blir dybdesensoren kalibrert, men i noen tilfeller hadde måleren forskjellig nullpunkt i starten og slutten av en måleserie. Målingene endret seg på grunn av at VL6180-sensoren ble varm. Dette utgjør likevel ikke et problem, ettersom dybden for en gitt kompresjon er basert på det laveste punktet før kompresjonen og den høyeste målingen under kompresjonen. Dette neglisjerer drift i distansesensoren, samtidig som det motvirker effekten av *incomplete release*. Incomplete release betegner ufullstendige kompresjoner, som vil si situasjoner hvor brystkassen starter i en posisjon hvor den allerede er presset inn et lite stykke. Selv om brystkassen blir inntrykket 60mm kan det vær at brystkassen allerede var inntrykket 20mm idet kompresjonen startet. Selve kompresjonen blir da bare 40mm. En algoritme for dybde-estimering må da prøve å estimere den faktiske posisjonsforandringen på 40mm, og ikke den totale kompresjonsdybden på 60mm. Merk at det kreves en større kraft å utføre en kompresjon fra 20-60mm enn fra 0-40mm på grunn av fjæren i dukken $F = -kx$. Dette burde likevel ikke ha noe å si for posisjonsestimeringen ettersom dobbeltintegralet av akselerasjon i teorien alltid skal resultere i posisjonsendring.

6.1 Scoring

Hva definerer en god estimator? For å kunne si noe om hvor bra et dybde-estimat er trengs en slags score-funksjon som uttrykker *hvor bra estimatoren egentlig er*. Som ble diskutert i testen av dybdemåleren i kapittel 3.8, er det flere måter å kalkulere *presisjon* på. Det hjelper lite om dybdeestimatene er gjennomsnittlig helt korrekt for en lang HLR-sekvens, men for hver isolerte kompresjon er avviket +- 30mm. Ønsket er å holde avviket for hver kompresjon så lavt som mulig. Med avviket for en kompresjon menes den estimerte dybden minus den reele dybden. For alle estimatorene blir presisjonen kalkulert og fremstilt i form av RMSE og MAD. I tilegg kalkuleres nøyaktigheten som MSD, som kan justeres bort med et offset. MAD tilsvarer det gjennomsnittlige avviket i absoluttverdi og betegner da spredningen av estimatene. RMSE forteller det samme, bare vektlegger og straffer større avvik mye mer enn små avvik. Hva som er viktigst er uvisst, men det er åpenbart at om begge blir lavere estimatene bedre. I de kommende delkapitlene blir flere teknikker for dybde-estimering utforsket. Ytelsen til disse estimatorene testes til slutt på et nytt og ukjent testsett, som presenteres sist i kapittel 6.

6.2 Dobbeltintegrasjon: Magnitude av Akselerasjon

Den første og enkleste måten for dybdeestimering baserer seg på dobbelintegrasjon av akselerometermålinger fra smartklokken. Som ble diskutert i kapittel 5.7 var det uvisst om det var best å estimere kompresjonsdybde basert på den stigende eller den fallende flanken til posisjonsgrafen. Figur 6.1 viser sammenligning av 3 forskjellige posisjonestimater og den reele dybden for de første 50 kompresjonene i treningsettet.



Figur 6.1: Sammenligning av estimert kompresjonsdybde og den reele komprimerte dybden.

I figur 6.1 kommer det fram at både dybde-estimatet basert på den stigende og den fallende flanken av posisjonsgrafen har en del store plutselige utslag fra den reele dybden. Et tredje alternativ *(avg)* ble da introdusert, nemlig gjennomsnittet av de to målingene. Dette estimatet er mindre utsatt for de store utslagene, og det antas derfor at det kan gi mer stabile og korrekte dybde-estimater.

Tabell 6.2: MAD og RMSE for 3 forskjellige estimeringsteknikker for dybdemåling.

	MSD [mm]	**MAD [mm]**	**RMSE**
Stig	-3.6	5.4	6.8
Fall	-3.7	5.3	6.6
Avg	-3.7	4.9	6.1

Tabell 6.2 fremstiller avviket og nøyaktigheten for disse tre elementære estimeringsteknikkene, basert på alle kompresjonene i treningsettet. Som ble spådd tidligere, er det avg-estimatet som gir minst avvik mellom målt og estimert dybde og kan derfor betraktes som det mest korrekte estimatet. Gjennomsnittlig, er dybdeestimatene 3.7mm for grunne for avg-estimatet. Dette kunne følgelig blitt justert ved å legge til et offset på 3.7mm for alle dybde-estimater. I dette forsøket ble posisjonsestimatene filtrert med vilkårlig valgte verdier som så ut til å gi tilstrekkelig bra resultater. I det neste avsnittet blir det etterforsket hva som skjer ved valg av disse filtreringsverdiene, og det blir lagt til offset for å optimalisere nøyaktigheten og presisjonen til estimatene. Videre vil all optimalisering prøve å minimere avviket til dybden basert på avg-estimatet ettersom dette var det mest korrekte dybde-estimatet i sin originale tilstand.

6.3 Optimalisering av parametre

Som diskutert i 5.2 fører integrasjon med seg drift. Magnituden av akselerasjon lavpassfiltreres først, før det så integreres to gangner. Å velge verdiene for disse filtrene vil ha påvirkning på oppløsningen, hastigheten og støyen i det resulterende dybde-estimatet. Variablene som justerer dette er gitt i tabell 6.3. For dybdeestimering med dobbeltintegrasjon vil det beste settet av k-verdier, være de verdiene som gjør at den estimerte dybden er likest den reele dybden målt av annedukken.

Tabell 6.3: Variabler brukt under lavpass og høypassfiltrering fra magnitude av akselerasjon til et dybdeestimat. Minimum og maksimumsverdiene for variablene under søks-algoritmen er også gitt, samt deres optimerte verdi

Variabel	min	maks	Optimal	Formål
LP_k	0.60	0.80	0.63	Lavpassfiltrering av MAA
vel_k_HP	0.85	0.95	0.92	Høypassfiltrering for å fjerne driften i farts-signalet
pos_k_HP	0.85	0.95	0.94	Høypassfiltrering for å fjerne driften i dybde-estimeringen
Offset	-10	10	-6	Konstant offset [mm] lagt til alle posisjons-estimater
Faktor	0.2	2.0	1.1	Konstant faktor mulitplisert med alle posisjons-estimater

Disse verdiene er vanskelige å velge, ettersom forskjellige kombinasjoner av dem kan gi bra løsninger. For å finne det optimale settet av variabler ble det først forsøkt å utføre et gradient-søk hvor man gir en algoritme et sett med startverdier, som så endrer verdiene for variablene i den retningen som maksimerer en gitt *scorefunksjon*. Utfordringen her er at dette problemet ikke er *konvekst*. Det vil si at det finnes utrolig mange *lokale løsninger*.

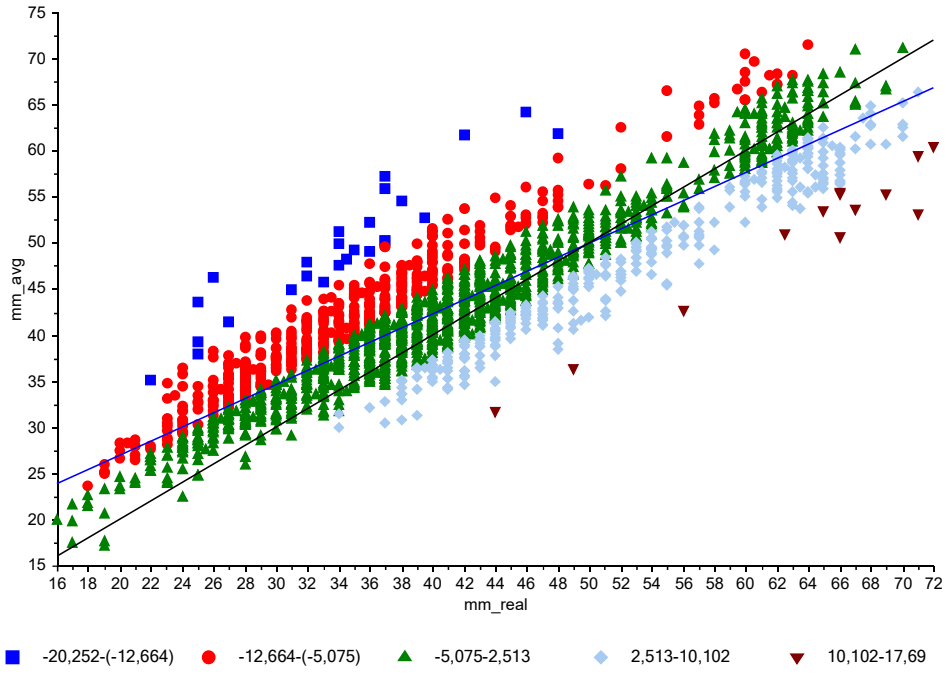
Gitt en situasjon en av variablene i 6.3 holdes konstant, vil det følgelig finnes en kombinasjon av de fire andre verdiene som gir best mulig løsning for akkurat det tilfellet. Dette er da en lokal løsning. Med gradient-søk kan det vær at en slik variabel blir satt fast og man optimaliserer for den faste verdien. Avhengig av hvilke startverdier gradient-søk-algoritmen har, vil den gå i en retningen som vil finne nærmeste *beste løsning*, altså nærmeste lokale løsning. Det er da ingen garanti for at man finner den globalt beste løsningen. Derfor ble det laget en funksjon basert på *brute-force*-prinsippet, som prøver å finne optimal løsning av filtrerings-verdier simpelthen med å sjekke et stort antall mulige kombinasjoner. I tabell 6.3 vises de verdiene som kom etter optimalisering av dybde-estimatene basert på treningsettet. Resultatet før og etter optimalisering vises i 6.4, hvor det kommer godt fram at både nøyaktighet og presisjon øker som følge av disse teknikkene.

I tilegg til optimalisering av filtrerings-verdier, ble det forsøkt å forbedre dybde-estimatene ytterligere ved å innføre både et *offset* og en *multipliserende faktor*. Ideen for dette var at en eventuell forskyvning i dybde-estimatene eller for stor/liten spredning skulle utlignes. Dette ble testet på samme måte som for optimalisering av filtreringsverdier, nemlig med en

Tabell 6.4: Sammenligning av presisjon for dybdeestimater *(mm_avg)* før og etter optimalisert signalbehandling

	MSD [mm]	MAD [mm]	RMSE
Avg Original	-3.7	4.9	6.1
Avg Optimalisert	-1.9	4.0	5.1
Offset + faktor	-0.2	3.6	4.7

funksjon som itererte igjennom x antall mulige kombinasjoner og målte RMSE og MAD av resultatet. En sammenligning mellom den estimerte dybden og den reele dybden for estimatene av *offset + faktor* vises i figur 6.2. Videre benyttes disse optimaliserte filterverdiene og offset+faktor for dybdeestimatene *mm_avg*.



Figur 6.2: Sammenligning av estimert kompresjonsdybde (mm_avg) og den reele komprimerte dybden (mm_real) for hver mm. Den reele dybden er langs x-aksen og de tilhørende estimatene for kompresjoner for den gitte dybden vises på y-aksen. Den optimale linjen for korrekte estiamter er gitt i svart, og regresjonslinjen for estimatene er fremstilt av den blå linjen. Fargene angir grupper som fremstiller avviket mellom estimert og reel dybde

I figur 6.2 er det mulig å se spredningen av dybden til kompresjonene, som går fra 16 til 72mm. Her kommer det fram av regresjonslinjen at målingene er litt forskjøvet. De grunne kompresjonene har en tendens til å bli estimert for litt dypere enn de er. Det kommer også fram at det er en del spredning i estimatene.

6.4 Multivariate Data Analysis

Multivariat Data Analysis går ut på oppdage mønstre i store datasett med flere variabler. I denne oppgaven går dette ut på å analysere kompresjonene basert på de variablene som er tilgjengelig for hver kompresjon. Hittil har variablene mm_rise/avg/fall blitt kalkulert for hver kompresjon. Fra akselerometeret går det også an å trekke ut delta-variabler for akselerasjonsutslagene i x, y og z retning.

I tilegg til akselerometeret, så inneholder smartklokken i denne oppgaven et gyrometer og et mangometer. Gyrometeret måler som kjent rotasjon, noe som muligens kan være interessant for å øke presisjonen til dybde-estimatene som foreløpig bare har vært basert på akselerometeret. Derfor ble det i tilegg logget data fra gyrometeret i smartklokken. Spørsmålet var egentlig *hva* gyrometeret egentlig kan fortelle oss om en kompresjon, og hvordan vi kan ekstrahere denne informasjonen til å bli fremstilt av enkle variabler. Rotasjonen til gyrometeret er gitt som rad/s, og integralet av gyrometer-målinger vil tilsvare den totale rotasjonen under en kompresjon. Det er naturlig å anta at en kompresjon ender og starter i omtrent nøyaktig samme stilling. I løpet av en kompresjon burde med andre ord endringen i orientering være 0, og den integrerte gyromålingen for en kompresjon burde også være lik 0. Blir den totale rotasjonen noe annet enn 0 er det vanskelig å vite om det skyldes drift, eller fordi håndleddet nå har en annen rotasjon. Noe som muligens er mer innholdsrikt er utslaget i gyro-målingene i løpet av en kompresjon. Det sier noe om kompresjonen ble utført rolig og lineært, eller kraftig og med rotasjon av håndleddet.

Både integralet og delta-målinger for gyroskopet tas med videre i den multivariate data-analysen, for å utforske hva slags informasjon som finnes i disse variablene. I kapittel 6.5 blir Principal Component Analysis utført for å undersøke dette.

6.5 PCA - Teori

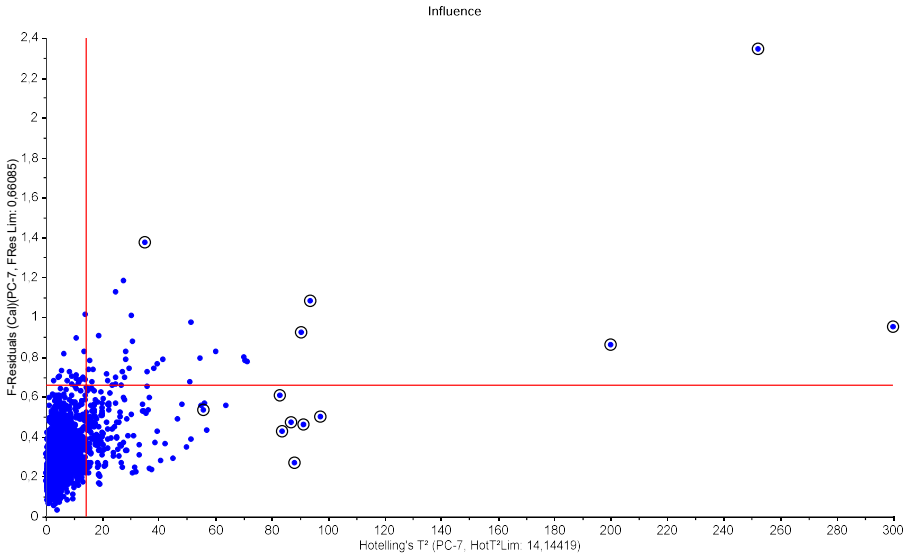
Når tileggsinformasjon om kompresjonene blir introdusert, får man følgelig mer informasjon tilgjengelig. Dette burde åpenbart være en fordel for å kunne lage en presis og robust modell, men spørsmålet er *hva* denne informasjonen forteller oss. Si at hver kompresjon består av et dybde-estimat, en dybde-referanse og 10 tileggsvariabler. Hvordan kan man vite hvilke av disse tileggsvariablene som inneholder interessant informasjon? Har maksutslag i gyro-måling i z-aksen noe som helst å si for nøyaktigheten av dybde-estimatet for en kompresjon? Har rytmen HLR ble utført med, som påvirker tiden en kompresjon tok, noe å si? Om disse variablene gir noe meningsfull informasjon er vanskelig å forutsi da det ikke nødvendigvis er noen åpenbare sammenhenger for det blotte øyet. For å få innsikt i dette benyttes *Principal Component Analysis*, som er en ypperlig måte for å redusere dimensjonen på et dataset men samtidig beholde all nyttig informasjon. PCA er en statistisk metode hvor et dataset blir transformert til et alternativt koordinatsystem som fremhever mønstre i datasettet. Formålet med PCA er altså å visuelt fremstille sammenhenger og korrelasjon mellom variablene i datasettet. Dette gjøres ved å flytte dataen til et koordinatsystem med færre dimensjoner, men som samtidig beholder all essensen og karakteristikk fra datasettet. PCA kan med andre ord brukes for å se om det finnes noe nyttig informasjon i tileggsmålingene som blir lagt til hver kompresjon (27) (28).

PCA blir utført ved å foreta en dimensjons-reduksjon av egenskapsrommet, hvor dataen blir fremstilt i det nye koordinatsystemet basert på datasettets *Prinicpal Components*. De prinsipielle komponentene er egenvektorene til kovariansmatrisen for datasettet. Egenverdiene til hver egenvektor forteller hvor mye varians det er i dataen for retningen til egenvektoren. Det vil si at første prinsipielle komponent (PC1) uttrykker *retningen av mest varians for datasettet*, altså retningen til den største spredning og er det samme som den egenvektoren med størst egenverdi. PC2 blir da retningen av ant mest variasjon i datasettet og så videre. I denne oppgaven blir programvaren *Unscrambler* brukt for kalkulering av PCA (29) (30).

6.6 PCA - Resultater

Før PCA kalkuleres må det tas et par hensyn i forhold til datasettet som er tilgjengelig. I tilfellet her er alle variablene/målingene tilgjengelige for hver kompresjon beskrevet numerisk, som er en klar fordel. I tilegg er det slik at høye verdier blir prioritert ovenfor lave når PCA kalkuleres. MM-estimatene har en numerisk verdi fra 0-75, mens gyro-målingene gjerne ligger rundt 0-3. Ettersom spennet i verdier er mye større for mm-estimatene blir også variansen større. PCA kan mistolke dette til å tro at variabelen med dybde-estimatene er ”viktigere”, simpelthen bare fordi verdiene er mye større. Derfor blir alle variablene *standardisert*, slik at alle variablene i utgangspunktet får lik vekting i modellen. I tilegg gjennomfører Unscrambler kryss-validering av modellen, som den bruker SVD (Singular Value Decomposition) for å lage.

6.6.1 PCA - Influence og Explained Variance



Figur 6.3: Influence-plot av PC1 og PC2 med outliers markert. Her består datasettet av 2116 samples.

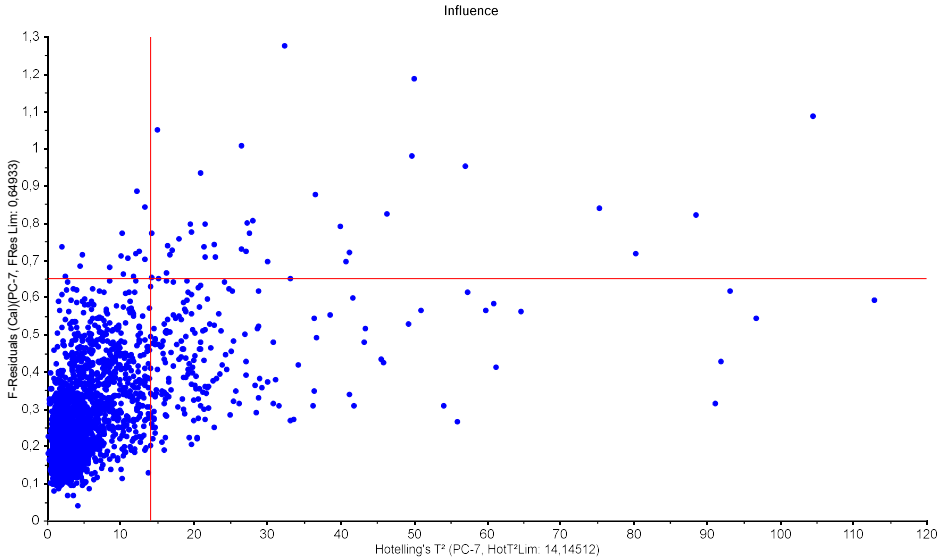
Kalkulering av PCA med Unscrambler gir en rekke forskjellige plot. I figur 6.3 vises *influence*-plottet for PCA. Influence-plottet fremstiller hvor bra hver kompresjon blir representert av modellen, da den viser *residualet* mellom referanse og modellen. Her er det fordel at så mange av samplene (kompresjons-objektene) ligger så nærme origo som mulig, da dette betyr residualet til samplet er lite. På y-aksen av figuren vises *residual variance*, som egentlig viser til hvor mye gjenstående varians som ikke er forklart av de 7 prinsipielle komponentene analysen består av. Minimalt residual er altså et mål på hvor bra samplene er forklart av modellen. Samples med høyt residual utgjør ikke nødvendigvis noe problem, det forteller oss bare at PCA ikke forstår hvorfor den samplen er som den er. Dette plottet blir analysert først for å kunne isolere og fjerne uønskede samples før de andre plottene blir studert.

På x-aksen av figuren fremstilles *leverage*. Denne aksen gir uttrykk for hvor bra samplene er beskrevet av modellen og deres påvirkning. En sample som er langt ute til høyre passer gjerne modellen bra, men den har stor påvirkning på modellen. Ytterpunkter her har gjerne litt annen karakteristikk enn resten av de øvrige samplene, men er bra forklart av modellen. For mange slike ytterpunkter kan føre til uforutsigbar oppførsel og gjøre modellen dårligere for de øvrige samplene. De mest skadelige samplene er de med høy leverage og høyt residual. De er dårlig forklart av modellen, samtidig som de påvirker modellen mye. Slike samples som påvirker modellen på en negativ og unødvendig måte, kalles *outliers*. Om disse outlierene viser seg til å være feilmålinger eller har en annen unormal oppførsel burde de fjernes, for å tilpasse modellen bedre til de samplene som faktisk er representative for det en prøver å måle.

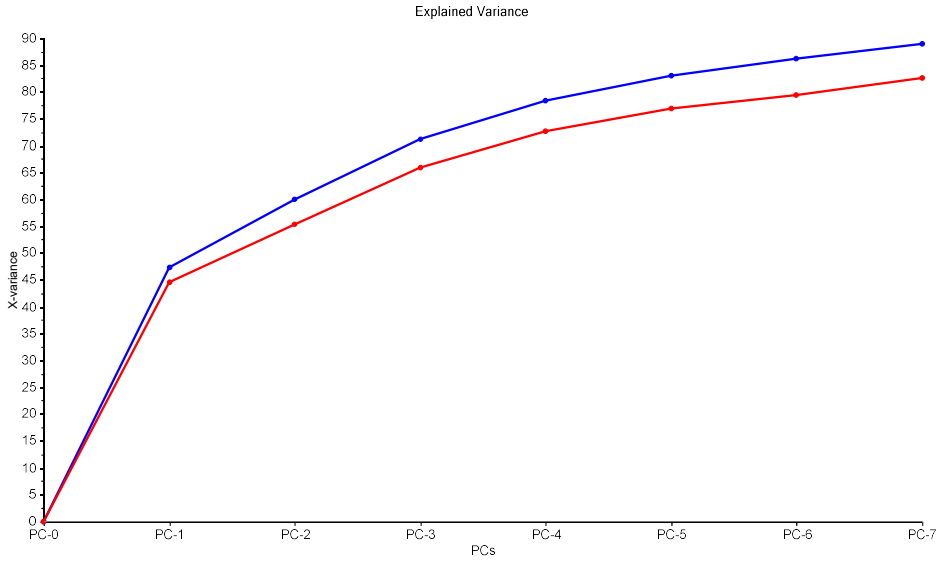
Noe som kom fram av influence-plottet i figur 6.3 var at noen av samplene var *outliere*, altså samples som ikke kunne bli særlig bra forklart av de prinsipielle komponentene. Det var spesielt en gruppe med kompresjoner som skilte seg ut, ved at dybde-estimatene typisk hadde et avvik mot mm_real på ca 40mm uten noen særlig grunn. Dette viste seg å faktisk være feilmålinger, for første og siste kompresjon i en måleserie får urealistisk dårlige målinger som følge av høypassfiltreringen av MAA. Her var det likevel bare dybde-estimat-variablene som skilte seg ut, for "delta"-variablene var som de skulle. Med andre ord, det kom fram av influence-plottet at PCA ikke hadde noen ide om hvorfor akkurat de samplene ble som de ble. I forhold til resten av samplene i datasettet, så ga ikke de samplene mening. Dette er også synlig i dybdeestimatene i figur 6.1, hvor første kompresjon har urealistisk dårlig estimat. Disse kompresjonene var uansett meningen at skulle kuttes, da disse estimatene blir ødelagt av innsvingningen til filtreringen. Om outlierene hadde vært forårsaket helt naturlig, kunne de ikke bare bli fjernet helt uten videre. Om det er spesielle kompresjoner som gjør at de samplene blir som de blir og det er forventet at det kommer nye slike samples i fremtiden, må nesten modellen kunne forstå hva som skjer.

Å fjerne disse outlierene er enkelt å innføre på smartklokken. Det løses ved å simpelthen ignorere dybdeestimatet fra den første kompresjonen i en HLR-sekvens. Siden disse outlierene har en klar begrunnelse og vi vet hvordan de oppstår (og hvordan å fjerne dem i sanntid) er det derfor trygt å fjerne disse outlierene fra modellen. Merk at disse samplene ble fjernet fra datasettet i python etter at de ble oppdaget, slik at disse kompresjonene heller ikke inngår i resultatene for estimeringen basert på bare integrasjon. Totalt ble 22 samples fjernet som følge av dette. Figur 6.4 viser hvordan influence-plottet ble etter dette subsettet av målinger ble fjernet. Ringene som er markert i 6.3 er et utdrag av de som ble fjernet. Fjerning av disse samplene ga en merkbar forskjell for både scores og Loadings for PCA-modellen. Et par "ekstreme" punkter på scores-plottet ble fjernet og det gjenstod en mye mer uniform samling av punkter. Dette gjorde også at variablene i loadings-plottet endret seg litt, men det ga ingen drastiske forskjeller. Resten av plottene i dette kapittelet benytter seg altså av det originale treningsettet hvor disse outlierne har blitt fjernet. Selv om disse ekstreme ytterpunktene ble fjernet er det fremdeles ganske mange samples som ikke blir perfekt forklart av PCA-modellen. Det forelå ikke noen åpenbar sammenheng mellom disse ytterpunktene, og grunnen til at de er her kan skyldes målestøy eller simpelthen at det er mye uforutsigbarhet med å blande så mange sensormålinger.

I Figur 6.5 fremstilles *explained variance*-plottet for å gi en indikasjon på *hvor* mye av variansen til datasettet de forskjellige prinsipielle komponentene forklarer. Første PC forklarer 47% av variansen, og totalt forklarer de første 4 PC'ene 80% av variansen i datasettet. Det er fordelaktig at så mye varians som mulig kan forklares av et lavt antall prinsipial-komponenter, og det fremstår her som om de prinsipielle komponentene faktisk får frem strukturen i datasettet og ikke bare forklarer støy.



Figur 6.4: Influence-plot av PC1 og PC2 der et sett av outliere har blitt fjernet.



Figur 6.5: Explained-variance-plot av PC1 og PC2. Den blå linjen er for kalibrering og den røde er for validering.

6.6.2 PCA - Loading og Scores

Et av de mest interessante plottene fra PCA er *loadings*-plottet. Her fremstilles vektene og påvirkningen til variablene for de prinsipielle komponentene. Loadings-plot er nyttige for å fremstille:

- Korrelerte variabler
- Negativt korrelerte variabler
- Variabler med lav påvirkning (rundt origo)

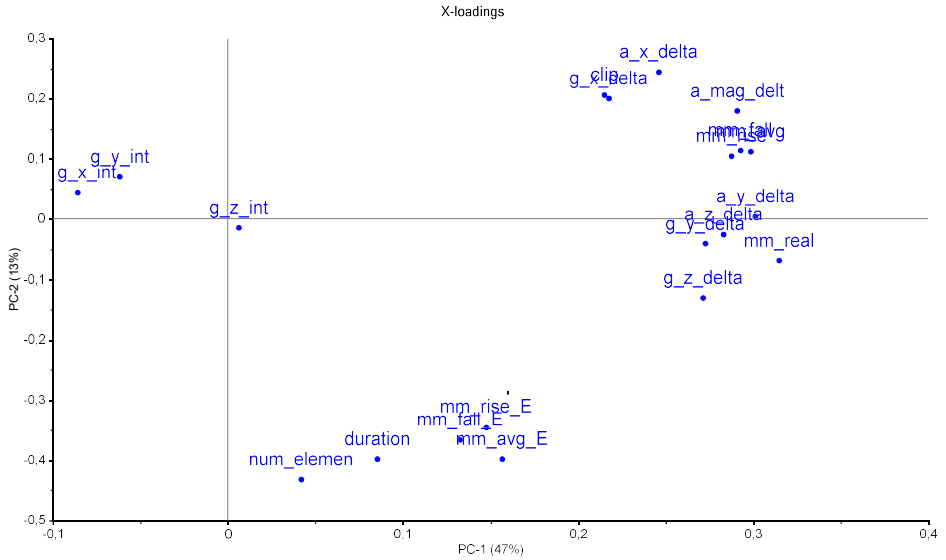
Variabler som ligger nærme hverandre er gjerne sterkt korrelerte, og variabler som ligger på motsatt ende av en prinsipiell akse er muligens negativt korrelerte. De variablene som er lengst unna origo er de som forklarer mesteparten av variasjonen i datasettet og inneholder derfor mest nyttig informasjon.

Figur 6.6 fremstiller vektene som påvirker de to første prinsipielle komponentene, med PC1 på x-aksen og PC2 på y-aksen. mm_real er den variabelen som er lengst ytterst til høyre, og ligger omtrent midt på PC1. Det vil si at PC1 i stor grad fremstiller kompresjonsdybden til objektene. Dette gir mening i form av at kompresjonsdybde i stor grad er det som skiller kompresjonene fra hverandre. Her er det noen tydelige grupperinger og korrelasjoner som er synlige:

- mm_fall, mm_rise og mm_avg ligger i en klynge oppe til høyre. Disse ligger veldig tett, som betyr at de er sterkt korrelerte. Dette gir logisk mening ettersom disse tre variablene egentlig bare er tre forskjellige måter å fremstille samme dybde-estimat på.
- Det samme gjelder for avviks-variablene mm_fall_E, mm_rise_E og mm_avg_E, som bare er avviket mellom den reele dybden og dybdeestimatene
- num_elements og duration er korrelerte. At en lengre kompresjon gir flere målinger er ikke overraskende.
- Alle ”delta”-variablene er langt ute på x-aksen. Det betyr at dypere kompresjoner ofte medfører større maksutslag i akselerasjon og rotasjon

De fleste av disse sammenhengene er åpenbare og bekrefter en del antagelser som ble gjort på forhånd. Men, det som er litt interessant er at grupperingen av avviks-variablene og duration ligger nærme hverandre. Det betyr muligens at avviket mellom målt og estimert dybde har en sammenheng med hvor lang kompresjonen var. Dette vises også i form av at PC2 fremstiller tiden brukt. A_x_delta er en måling for maksimalt utslag i x-akse for en kompresjon. Generelt sett vil en *raskere* kompresjon følgelig gi større utslag i akselerasjon. Det gir også mening at A_x_delta og duration som er på motsatt side av PC2 er negativt korrelerte. Høy duration medfører at kompresjonen går sakte, mens en lav duration tilsvarer en rask kompresjon som gir høyt utslag i akselerasjon. Tregere kompresjoner kan dermed ha en tendens til å lavere avvik mellom målt og estimert dybde.

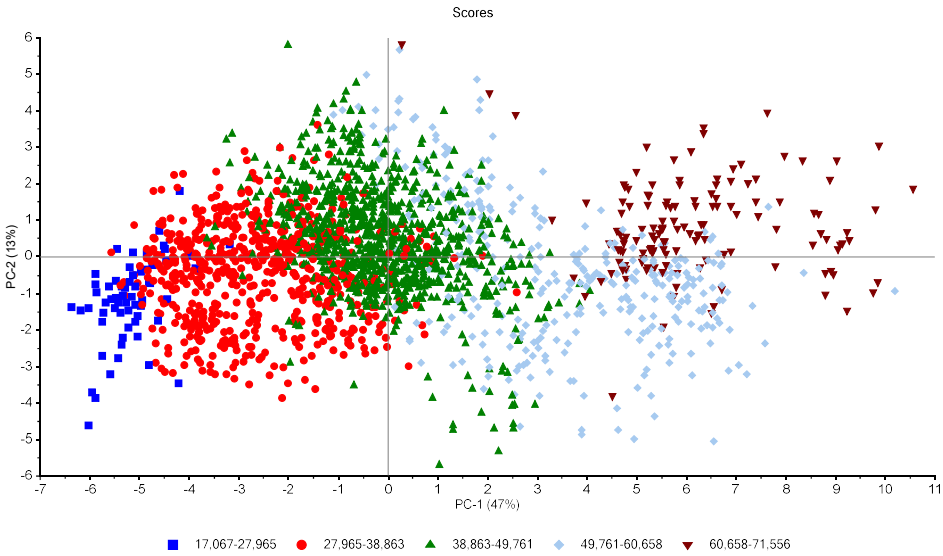
Noe annet nyttig som kommer frem her er at delta for gyro og akselerasjon i y og z retning er de variablene som ligger nærmest mm_real. Det er interessant at disse variablene er mer korrelerte med den ekte dybden enn de faktiske posisjonsestimatene (mm_avg). I tilegg forteller dette oss at det kan ligge nyttig data i gyroskopmålingene. Større utslag i rotasjon og akselerasjon gir altså en indikasjon på dypere kompresjoner. For øvrig ligger integrasjonsmålingene av gyroskopet rundt 0. Dette var forsåvidt som forventet i kapittel 6.4.



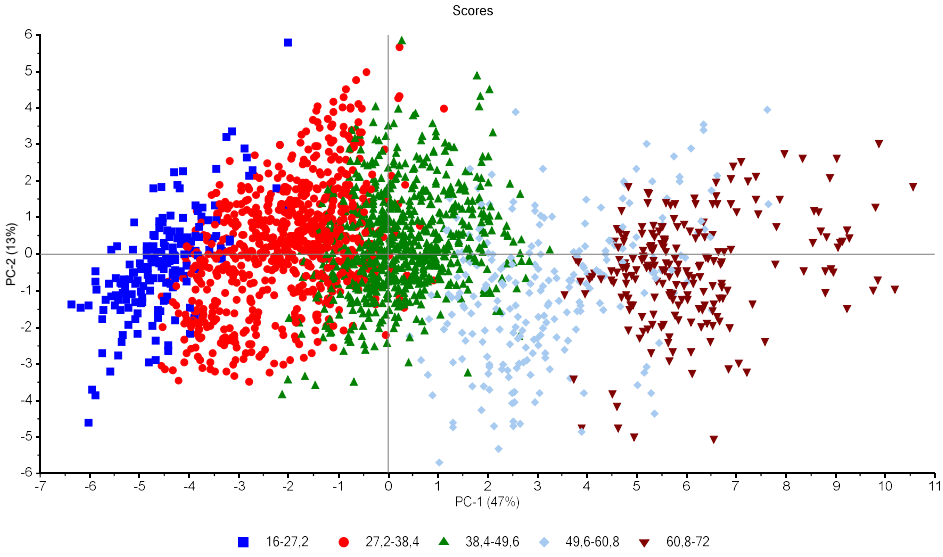
Figur 6.6: Loadings-plot av PC1 og PC2

Scores-plottet viser hvor objektene fra datasettet befinner seg i det nye koordinatsystemet basert på datasettets prinsipielle komponenter. Figur 6.7 viser scoreplot for datasettet for PC1 og PC2. Scoreplottene er egentlig helt like, forskjellen er bare at de er fargelagt etter økningen i henholdsvis mm_avg og mm_real. Sammenlignes subfigurene er det mulig å trekke paralleller til det som kom fram av figur 6.2. Det er tydelig at både mm_avg og mm_real øker langs PC1. Større reele dybder resulterer altså i større estimerte målinger, som er et bra tegn. Likevel så er det en slags forskyvning i dataen hvor begge øker langs PC1, men mm_real øker i en retning litt negativt på PC2 og mm_avg øker med positiv PC2. Noe annet som er litt merkelig, er at samplene ser ut til å samle seg i to grupper i scores-plottet, en stor gruppe til venstre og en litt mindre gruppe til høyre. Det fremkom ikke noen åpenbar grunn til denne grupperingen, men en forklaring kan være at datasettet er litt underrepresentert i en viss kompresjonsdybde. Dette kan observeres i 6.2, der det er litt færre observasjoner mellom 54-60mm enn de øvrige kompresjonsdybdene.

Det virker til å være mye informasjon i datasettet som kan utnyttes for å forbedre dybdeestimatene til kompresjoner. Et alternativ er å prøve seg frem med de variablene man har ved å multiplisere og legge til bias basert på f.eks varighet, gyro_delta etc. En annen metode er å lage en estimator som baserer seg på den informasjonen som PCA ekstraherer.



(a) Fargegruppering basert på mm_avg



(b) Fargegruppering basert på mm_real

Figur 6.7: Score-plot av PC1 og PC2. Fargeleggingen i grupper fremstiller dybden for mm_avg og mm_real

6.7 Prediktiv modell

Regresjons-analyse er en mye brukt metode for å lage prediktive modeller, og er en form for maskinlæring. Målet er å lage en modell for å prediktere en respons basert på et sett med målinger. I dette tilfellet er oppdraget å prediktere den reele kompresjonsdybden basert på *målingene* som er tilgjengelig. Regresjonsanalyse skiller mellom to variabler, de uavhengige, og de avhengige.

- De *uavhengige variablene X* er målingene som er tilgjengelig for hver *sample*. I dette tilfellet, så er en sample en kompresjon, og de uavhengige variablene er samlingen av informasjon vi har om hver måling. Dette er alle variablene som blir kalkulert basert på informasjon fra smartklokkens sensorer. Dette inkluderer deltamålingene for gyro og akselerometer, samt posisjonsestimatene kalkulert
- De *avhengige variablene Y* er responsen vi prøver å prediktere basert på de uavhengige variablene. I dette tilfellet blir den eneste avhengige variabelen kompresjonsdybden, ettersom det er den vi prøver å prediktere basert på målingene for kompresjonen.

Regresjonsmodeller prøver altså å prediktere avhengige variable basert på de uavhengige variablene. Ettersom det er flere uavhengige variabler, blir dette definert som multivariat regresjons-analyse. For multivariat regresjon finnes det tre mye brukte metoder:

- MLR - *multiple linear regression*, som er en lineær metode
- PCR - *principle component regression*
- *PLSR/PLS - partial least square regression* eller *projection to latent structures* som er to forskjellige navn for samme metode.

PCR og PLSR skiller seg fra MLR ved at de bruker *latente variable*. Dette vil si at de bruker den underliggende og gjerne skjulte informasjonen i et dataset. *Principle component regression* er tilnærment en regresjonsmodell-utgave av PCA. Modellen benytter seg av de prinsipielle komponenetene som prediktorer i en linær regresjons-modell. En grunnleggende antagelse med PCR er at retningen hvor prediktorene viser mest varians er direkte assosiert med responsen. Med andre ord vil PCR vil prøve å maksimere variansen, men det er ikke nødvendigvis dette er det som gir best estimat for responsen. I motsetning til PCR vil PLSR finne en regresjonsmodell ved å transformere både de uavhengige og de avhengige variablene til et nytt rom. I dette nye rommet vil PLSR ha dannet nye latente variable som er mer relevante for prediksjon av referansen Y. Det ble antatt at PLSR ville gi best resultater av regresjons-metodene, og det blir derfor nærmere undersøkt. De uavhengige variablene for prediktive analyser er gitt i tabell 6.5, og den ene avhengige variabelen er referansedybden *mm_real* (29) (31).

Tabell 6.5: Uavhengige variabler for prediktiv modell, sortert etter hvor målingene/variablene kommer fra

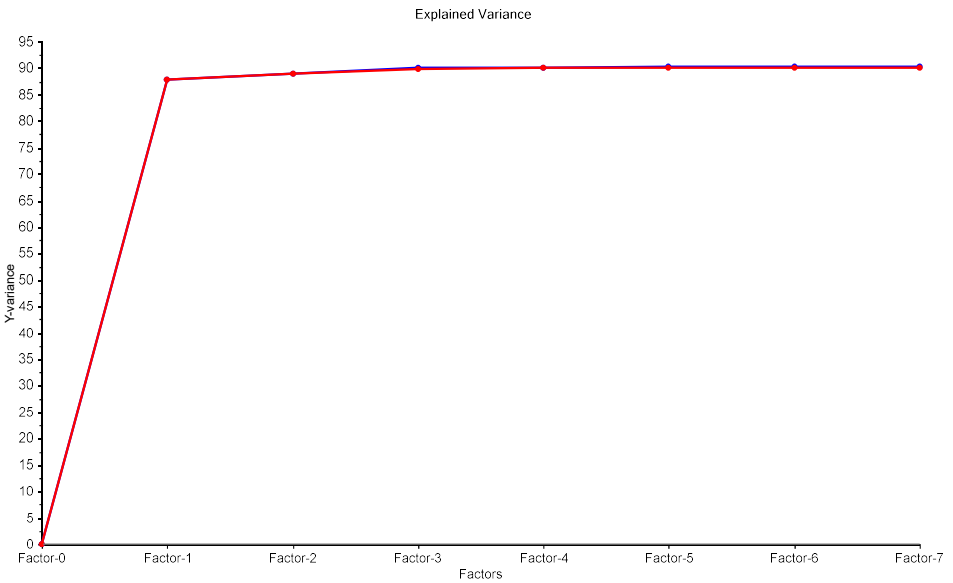
Generelt	**Akselerometer**	**Gyrometer**
Num_elements	a_x_delta	g_x_delta
Duration	a_y_delta	g_y_delta
	a_z_delta	g_z_delta
	a_mag_delta	g_x_int
	mm_fall	g_y_int
	mm_rise	g_z_int
	mm_avg	
	clip	

6.8 PLSR

PLSR ble kalkulert med Unscrambler på omtrent samme måte som for PCA. De uavhengige variablene fra tabell 6.5 er prediktorene X, og mm_real er responsen Y. Også her blir variablene i datasettet standardisert. Kryss-validering blir utført, og KERNEL PLS blir brukt for å lage modellen.

En av fordelene med å bruke PLSR er at på samme måte som med PCA, får man en grafisk oversikt av modellen ettersom den er basert på latente variable. En faktor å ta hensyn til med PLSR er hvor mange av de prinsipielle komponentene modellen skal baseres på, og dette kan analyseres ved å se på modellens plot for "explained variance".

6.8.1 PLSR - Explained Variance

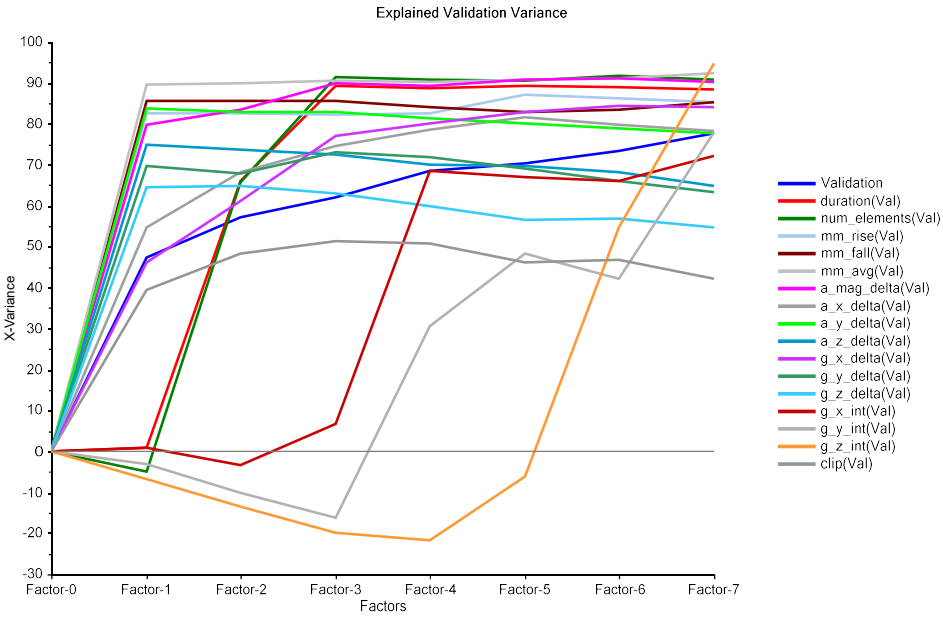


Figur 6.8: Explained Variance-plot for Y-referanse for PLSR-modellen

Som er synlig i figur 6.8, forklarer den første faktoren av PLSR 90% av variansen mellom datasettet og referansen. Å inkludere mer enn den ene faktoren gir ikke nevneverdig høyere forklart varians, men spørsmålet er om de neste faktorene inneholder nyttig informasjon. Flere faktorer forklarer mindre og mindre av variansen, og det er vanskelig å si om de bare forklarer støy. Likevel, blir RMSE for valideringen lavest ved å ha med 4 faktorer. Disse verdiene vises i tabell 6.6 og sammenlignes med et par av estimerings-algoritmene som ble beskrevet tidligere i dette kapitellet.

Tabell 6.6: Sammenligning av presisjon for dybdeestimater basert på sitt eget treningsett

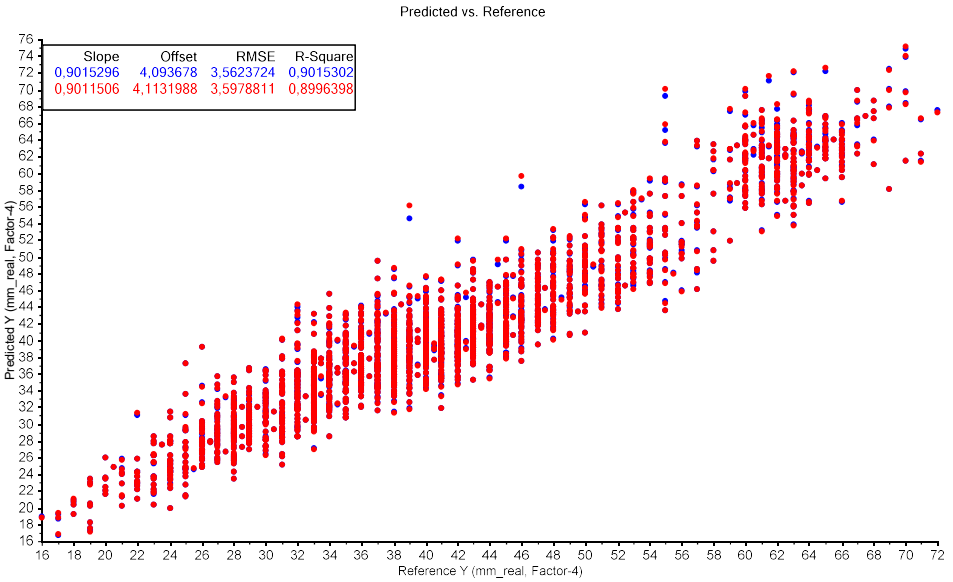
	MSD [mm]	MAD [mm]	RMSE
Avg Original	-3.7	4.9	6.1
Offset + faktor	-0.2	3.6	4.7
PLSR faktor 1	0.0	3.1	4.0
PLSR faktor 4	0.0	2.8	3.6



Figur 6.9: Explained Variance-plot for X-variablene for PLSR-modellen

I figur 6.9 fremstilles den forklarte variansen for de individuelle x-variablene til de forskjellige faktorene til PLSR-modellen. Faktor 1 forklarer mye av variansen langs det som tilsvarte PC1 for PCA-modellen. Denne faktoren forklarer mesteparten av variansen for dybde-estimatene med mm_avg øverst, og dette forklarer også en del av variansen til delta-variablene. For at over 90% av variansen til duration/num_elements skal være forklart, må 3 faktorer inkluderes. Faktor 5 og utover forklarer variansen til gyro-integralene på bekostning av de andre variablene. En modell basert på første faktor er muligens mer forutsigbar og pålitelig enn å inkludere 4, men dette er vanskelig forutse. Det er i det minste ikke noe poeng å inkludere mer enn de første 4 faktorene, da faktor 5 og oppover fører til en økning av RMSE for validering-settet. Flere faktorer enn 4 innfører sannsynligvis bare støy til modellen.

6.8.2 PLSR - Prediksjon vs Referanse



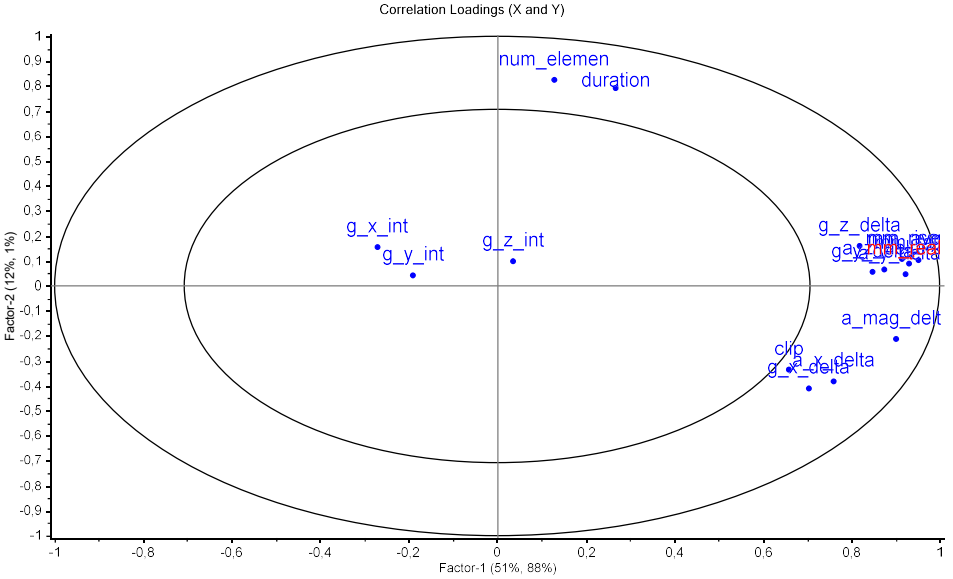
Figur 6.10: Predikterte dybdeestimater vs referanse for PLSR-modellen med 4 faktorer. De blå tallene er fra kalibrering, og de røde er for validering-resultater.

Plottet i figur 6.10 viser RMSE for de predikterte verdiene og mm_real. Det fremstår her som om modellen passer bra til treningsettet. Utenom et par unormalt "høye" målinger er det ikke noen spesielle outliere som skiller seg ut. Noe som kommer frem fra disse resultatene er at den enkle estimeringen med dobbeltintegrasjon i kapittel 6.2 har 31% høyere RMSE (Offset + faktor vs PLSR faktor 4). Det er en vesentlig forskjell, og nesten halvparten av RMSE for den originale algoritmen. Likevel, disse resultatene her er basert på samme datasett som modellen ble laget på. Estimatorene her har blitt optimalisert for sitt eget treningsett, og det hjelper ikke så mye om modellene bare fungerer for treningsettet. For å teste hvordan disse prediktive modellene faktisk presterer er det på tide å teste dem med helt ny og ukjent data.

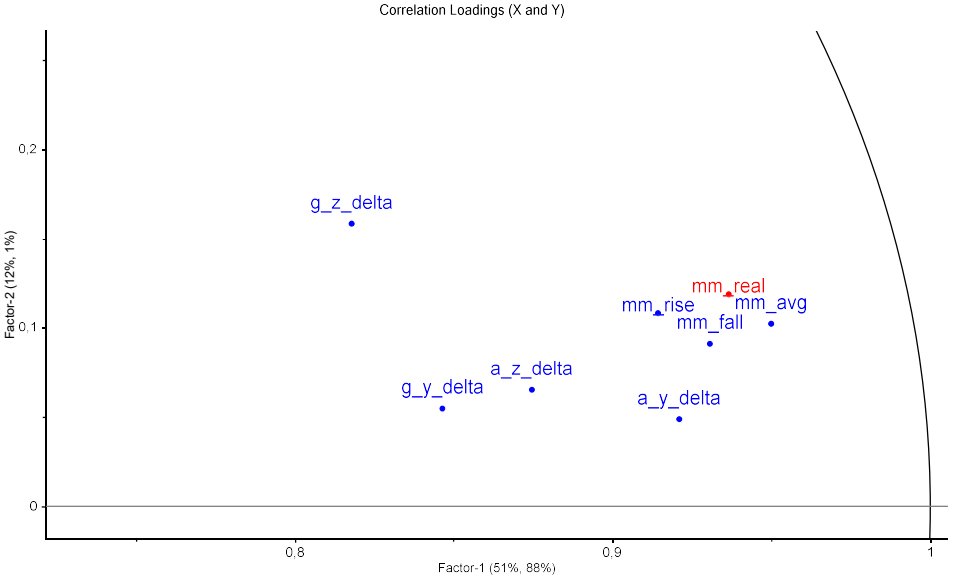
6.8.3 PLSR - Korrelasjon

I figur 6.11 er det et plot som minner om loadings-plottet fra PCA-analysen av det samme datasettet, bare at y-aksen er invertert. For PLSR er dette plottet meget interessant, da det fremstiller hvilke variabler som ”veier tyngst” i prediksjonen av referansen Y. Langt ute til høyre, ligger mm_real sammen med en klynge variabler.

Figur 6.12 er samme plott som 6.11, bare her er det zoomet inn på området rundt mm_real. Ikke spesielt overraskende, er de tre dybde-estimatene basert på MAA vesentlig for prediksjonen av mm_real. Disse estimatene er tross alt i ballparken til mm_real fra før av. Det som *er* interessant er samlingen av ”delta-målingene” i samme område. Dette betyr at disse variablene er viktige for modellen for å prediktere mm_real best mulig. Det kan dermed se ut som det ligger verdifull informasjon i gyrometermålingene.



Figur 6.11: Correleation Loadings-plot for Faktor 1 og 2



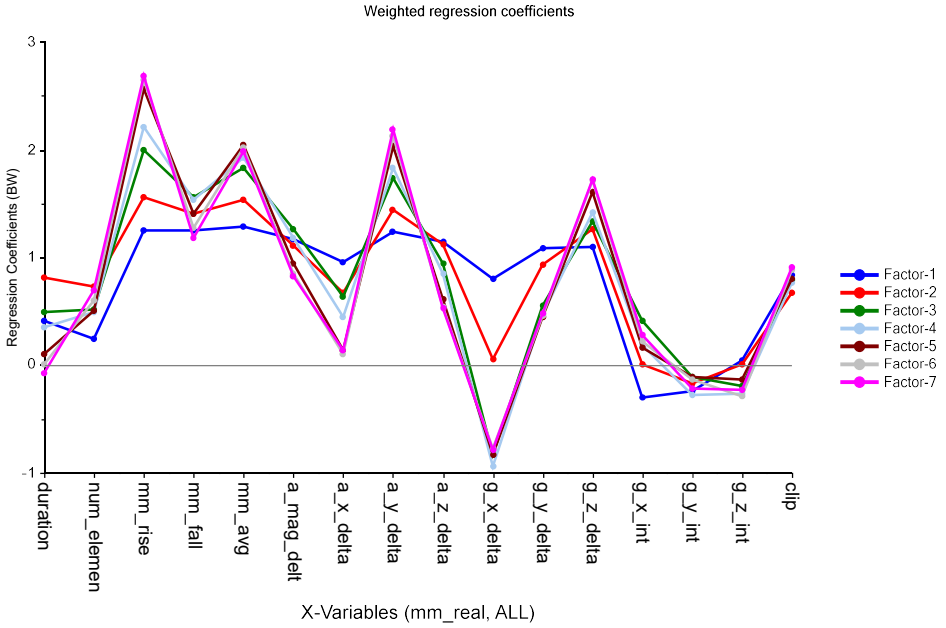
Figur 6.12: Correleation Loadings-plot for Faktor 1 og 2, nærbilde

6.9 Prediksjon med PLSR

Når prediktering med PLSR skal utføres finnes det to muligheter, hvor den første muligheten er *kort prediksjon*. Denne metoden benytter *regresjons-koeffisientene* fra PLSR-modellen til å kalkulere $Y_{predicted}$.

$$Y_{predicted} = b_0 + b_1 * x_1 + b_2 * x_2 + ... + b_k * x_k \tag{6.1}$$

I tilfellet for denne oppgaven har hver kompresjon $k = 16$ uavhengige målinger *x*. Det betyr at å prediktere et dybdeestimat *Y* for en kompresjon med en eksisterende PLSR-modell bare består av å multiplisere 16 målinger med deres regresjons-koeffisient og addere med et offset. Dette krever med andre ord veldig lite regnekraft. En sammenligning av regresjonskoeffisientene vises i figur 6.13. Denne figuren viser vektleggingen mellom de forskjellige variablene PLSR-modellen er bygget opp av. Her kommer det klart frem at vektlegging av integralmålingene fra gyrometeret er mye lavere enn for de øvrige uavhengige variablene. Det kommer også fram at inkludering av flere faktorer i PLSR-modellen at vektleggingen får større "dynamikk", hvor spennet i vekting blir større.



Figur 6.13: Sammenligning av regresjonskoeffisientene for hver faktor til PLSR-modellen

Tabell 6.7: Regresjonskoeffisenter for PLSR-modellen

	Vektet		Uvektet	
Variabel	Faktor1	Faktor4	Faktor1	Faktor4
B_0	-1,750	-5,418	-1,750	-5,418
num_elements	0,246	0,486	0,042	0,083
mm_rise	1,247	2,204	0,128	0,226
mm_fall	1,244	1,532	0,126	0,155
mm_avg	1,283	1,921	0,135	0,202
a_mag_delta	1,171	1,178	0,286	0,288
a_x_delta	0,951	0,447	0,294	0,138
a_y_delta	1,238	1,833	0,258	0,381
a_z_delta	1,141	0,849	0,224	0,167
g_x_delta	0,804	-0,946	1,208	-1,421
g_y_delta	1,085	0,470	2,751	1,190
g_z_delta	1,097	1,412	2,140	2,755
g_x_int	-0,305	0,180	-47,76	28,16
g_y_int	-0,239	-0,276	-90,74	-104,64
g_z_int	0,041	-0,263	11,25	-71,46
clip	0,836	0,758	0,429	0,389

Regresjonskoeffisientene for faktor 1 og 4, vektet og uvektet, vises i tabell 6.7. De vektede koefisientene er i forhold til hverandre, mens de uvektede er de som brukes under *short prediction* for å prediktere mm_real for en ny sample. En ting å bemerke seg med variablene brukt i PLSR-modellen, er vektingen til integralene av gyromålingene. Relativt til de andre variablene har disse målingene veldig lite å si, men disse integralene har nomalt målinger i orden 10^{-3} eller mindre. Det vil si at de *uvektede* regresjonskoeffisientene for disse variablene er skyhøye. Om hånden faktisk roteres mye under en kompresjon og gir et unormalt stort utslag på disse gyro-integralene vil det slå veldig ut på modellen. For å sikre mot dette burde enten disse estimatene tas vekk, ellers burde det vises at de har noen nytte. Slik det fremstår nå innfører ikke disse variablene noe annet enn støy, samt at de utgjør en risiko om det plutselig blir store rotasjonsendringer mellom hver kompresjon. I tillegg brukes det regnekraft på å kalkulere disse integralene som unngås om de ikke blir tatt med.

Den andre måten å prediktere dybde basert på PLSR-modellen er *full prediksjon*. Her blir de nye målingene *x* projisert inn i den eksisterende PLSR-modellen. Dette medfører at det er mulig å utnytte PLSR sitt fulle potensiale med outlier-detection. Ved å projisere kompresjonen over på PLSR-modellens X-rom blir samtidig nye T-scores kalkulert. I likhet med outlier-detection for PCA blir da X-residualet og leverage kalkulert for hver måling som forteller noe om kvaliteten på estimatet. Brukes full prediksjon kalkuleres det altså en prediksjon Y for dybden og en score for hvor bra estimatet passer med PLSR-modellen. Selve estimatet skal i teorien være identisk for begge metodene.

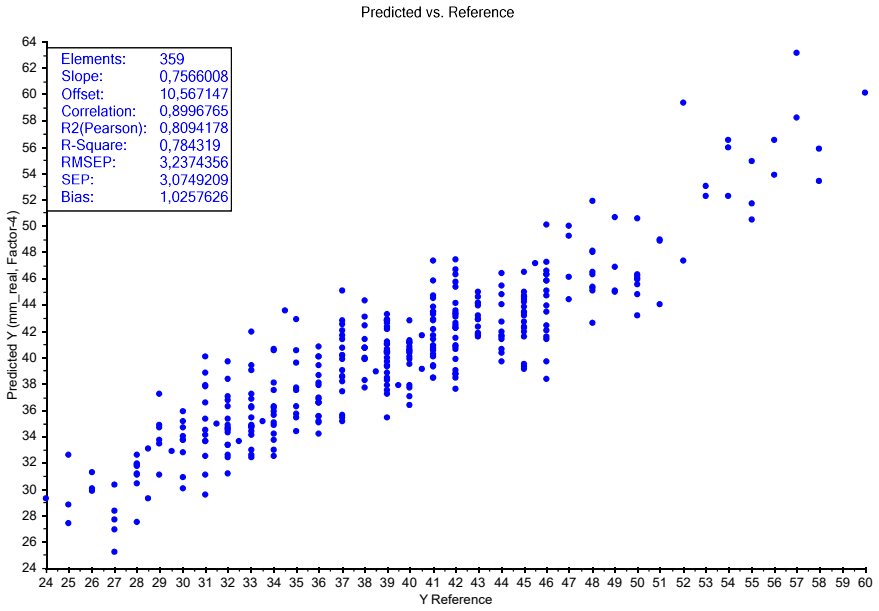
$$X = TP^T + E \tag{6.2}$$

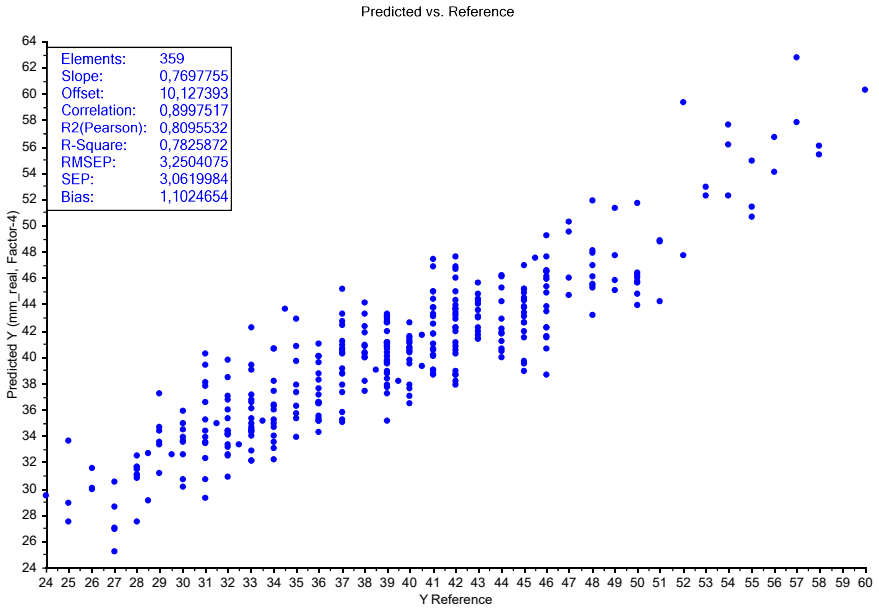
$$Y = TQ^T + F \tag{6.3}$$

For å kalkulere full prediksjon må først matrisen T kalkuleres, med f.eks NIPALS eller PLS1. Dette krever følgelig mer regnekraft enn å utføre kort prediksjon. Nyttigheten av å også vite kvaliteten til målingene kontra endringen i ytelse må selvsagt tas høyde for. Full prediksjon i form av NIPALS eller andre modeller ble ikke testet på smartklokken i denne oppgaven, men det eksisterer biblioteker for dette i Java.

I flere av delkapitlene har det blitt diskutert rundt viktigheten til integralene fra gyroskopet. Ettersom de virket til å bare inneholde støy ble det laget en ny modell for PLSR for å se hvordan predikteringen ble uten gyro-integralene. Alle de andre variablene for kompresjonene som er foreslått ser ut til å påvirke modellen i en positiv grad. Denne nye modellen er helt lik med den forrige utenom at gyr_x_int, gyr_y_int og gyr_z_int nå er fjernet. Basert på plottene for PLSR i unscrambler ser den nye modellen omtrent identisk ut med den forrige. Det førte til helt minimale forskjeller i loadingsplottet. Differansen for prediksjonen på testsettet mellom denne og den originale PLSR-modellen fremstilles i delkapittel 6.10. En sammenligning av de prediktere målingene fra PLSR-modellene med og uten gyromålinger kan sees i figur 6.14. Her fremstilles også spredningen av testsettet. Forskjellen med og uten gyro-målingene er minimale, og forklaringen til differansen kan godt bare skyldes støy som integral-målingene har ført med seg. At disse variablene ikke gir noe nyttig informasjon for PLSR-modellen ble også antatt basert på loadingsplottet av PCA. Det kan vær at det kommer fram fordeler med disse variablene med et større treningsett, men det foreligger ingen åpenbar grunn til å inkludere disse i PLSR-modellen.

(a) PLSR-Prediksjon med gyro_int





(b) PLSR-Prediksjon uten gyro_int

Figur 6.14: Predikterte målinger sammenlignet med mm_real for to PLSR-modeller (4 Faktorer inkludert). Testsettet er det samme for begge.

6.10 Ytelse på et testsett

I dette delkapittelet blir alle de foreslåtte dybdeestimerings-algoritmene testet med et helt nytt datasett. Modellene ble laget og optimalisert på treningsettet gitt i 6.1, men for å teste ytelsen i reele scenarioer blir algoritmene matet med ukjent testdata. Statistikk for testsettet som nå utprøves på algoritmene vises i 6.8. Dette testsettet er mindre enn treningsettet som ble brukt for å lage dybde-estimeringsalgoritmene, men HLR-sekvenene det er sammensatt av er utført på samme måte som for treningsettet. Denne testen er ment for å gi et innblikk i presisjonen til dybde-estimeringen for virkelighetstro situasjoner.

Tabell 6.8: Statistikk for testsett brukt for å sammenligne algoritmene for dybde-estimering

Egenskap	**Beskrivelse**
Størrelse	16367 linjer
Antall kompresjoner	359
- Dybde gjennomsnitt	41mm
- Dybde median	41mm
- Rekkevidde	27mm-63mm
- Outliers fjernet	4
Generelt	Datasettet er satt sammen av totalt 2 måleserier utført av to forskjellige personer
Frekvens	Snitt 0.44 sekund per kompresjon $\sim$136 BPM / 2.3hz

I tabell 6.9 blir nøyaktigheten og presisjonen til alle foreslåtte estimeringsteknikker på testsettet fremstilt. Alle algoritmene gjør en god jobb å estimere kompresjonsdybden til testsettet, men det er åpenbart at estimeringen basert på PLSR har lavere avvik enn den som bare er basert på dobbeltintegrasjon. Begge disse algoritmene har lavere MAD og RMSE enn det estimerings-modellene oppnådde for sitt eget treningsett. Dette er muligens litt overraskende, men det kan ha en naturlig forklaring i at testsettet har et lavere spenn i verdier enn treningsettet. Treningsettet har tettest populasjon av kompresjoner rundt 30-40mm, og det virker som det er i dette sjiktet under 60mm estimeringen fungerer best. I testsettet var det få "unødvendig" dype kompresjoner, og ingen dypere enn 63mm. Som diskutert tidligere kan de dypeste kompresjonene føre med seg unormalt store avvik på grunn av clipping. Siden det ikke er så mange dype kompresjoner, fører nok også dette til at RMSE og MAD er ekstra lavt for testsettet. Det er i denne kategorien normale HLR-sekvenser vil utfoldes og det er forsåvidt viktigst med høyest presisjon i dette området. Det må i tilegg tas i betraktning feilraten til dybdesensoren diskutert i 3.5. Da dybdesensoren ble testet, hadde den et gjennomsnittlig avvik på $\pm$ 1mm. Med det tatt i betraktning, er resultatene fra dybde-estimatene fra testsettet spesielt imponerende. Estimater med særlig mye lavere avvik enn dette under disse testforholdene virker ikke særlig oppnåelig med tanke på at presisjonen til referansenmålingen alene utgjør nesten halvparten av dette avviket.

Tabell 6.9: Sammenligning av presisjon for dybdeestimater på testsett

	MSD [mm]	**MAD [mm]**	**RMSE**
Avg Original	-5.9	6.0	7.0
Avg Offset + faktor	-2.6	3.4	4.4
PLSR faktor 1	-1.1	2.9	3.6
PLSR faktor 4	-1.0	2.6	3.23
PLSR faktor 4 uten g_int	-1.1	2.6	3.25

KAPITTEL 7

Diskusjon og videre arbeid

Resultatene for dybdeoptimering fra testsettet var meget positive og overbevisende for formålet med å ha en HLR-assistent-basert smartklokke. Til syvende og sist, så må spørsmålet *hvor bra er bra nok* besvares. Dybde-estimeringsteknikken basert på PLSR har et gjennomsnittlig avvik på under 3mm, som ifølge Lærdal Medical var en milepæl for å kunne implementere en robust dybdemåler. Skal dette brukes som en assistent til livredding er brukeren lite interessert i om kompresjonen var 33 eller 49mm. Brukeren er interessert i om kompresjonen var for grunn, akkurat passe eller for dyp. Et avvik på under 3mm burde være mer enn bra nok for å skille disse grensene fra hverandre. Om det uansett er ønskelig å dele opp i tre kategorier (for grunn, passe og for dyp) kunne det vært en ide å brukt en *Support Vector Machine* for de tre gitte klassene. Istedenfor å estimere mm_real som blir gjort med PLSR-prediksjonen, vil en SVM prøve å plassere en kompresjon i en av de tre klassene som blir gitt.

7.1 Implementasjon på smartklokke

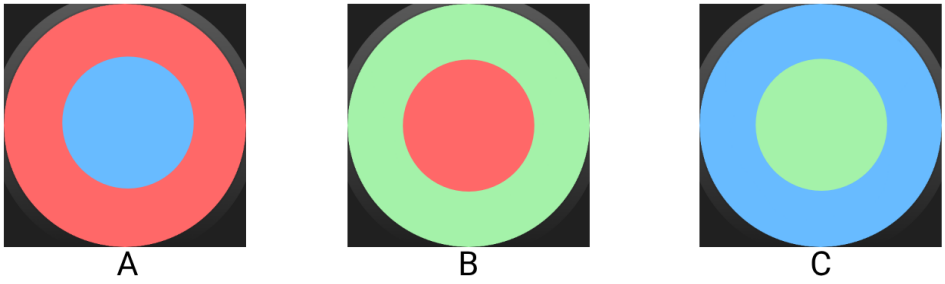
I kapittel 6 ble det foreslått flere metoder for å estimere kompresjonsdybde basert på smartklokkens interne sensorer. Disse metodene ble analysert på en stasjonær PC, og ble ikke implementert på smartklokken. For at PLSR med kort prediksjon skal kunne operere i sanntid på en smartklokke må den klare følgende:

- Logge sensordata med 100hz
- Lage et kompresjonsobjekt for hver nye kompresjon.
- Kalkulere alle verdiene x gitt i tabell 6.5. Dette medfører dobbelt-integrasjon og filtrering av MAA. Dette blir gjort med relativt enkle algoritmer.
- Multiplisere alle disse verdiene x med deres regresjons-koeffisienter, og addere dette sammen

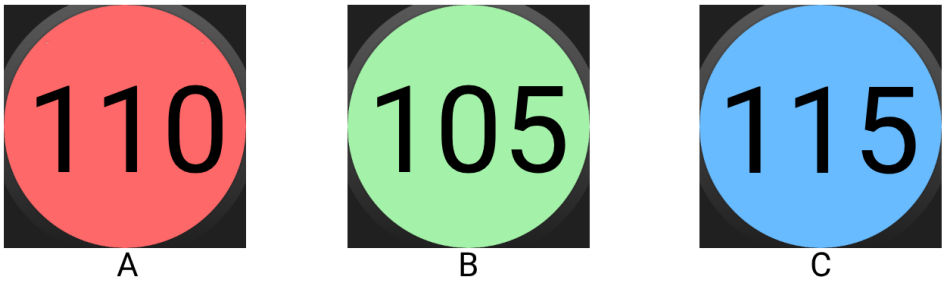
En kompresjon er på rundt et halvt sekund, som tilsvarer 50 linjer med data. Her er det så en rekke ting som må gjøres for å skape de 16 variablene som PLSR-modellen bruker. Brukes PLSR uten gyro_int er det forsåvidt bare 13 variabler som må kalkuleres. Den mest ”tidkrevende” oppgaven er sannsynligvis operasjonen ved å lage estimatet mm_avg. Dette estimatet krever at MAA kalkuleres, lavpassfiltreres, integreres, høypassfiltreres, integreres og så høypassfiltreres en siste gang. Felles for alle disse operasjonene er at de er lineære og har kjøretid $O(n)$. Har kompresjonen dobbelt så mange elementer, så tar det bare dobbelt så lang tid å kalkulere dybdeestimatene. Det var ikke tid til å gjennomføre test av kjøretid på smartklokken, men å kalkulere alle disse operasjonene med ikke-optimalisert pythonkode tok nanosekunder på en stasjonær pc. Det kan muligens ikke sammenlignes helt med en mobilprosessor, men det fremstår ingen grunn til at PLSR med kort prediksjon skulle være noe problem ytelsesmessig for de relativt sofistikerte mobilprosessorene som finnes i en smartklokke anno 2018.

For å ha en veileder på smartklokke, må brukeren få en eller annen form på tilbakemelding på hvor bra HLR blir utført. Det viktigste verktøyet en smartklokke har til dette, er skjermen. I tilegg kommer smartklokker som oftest med både en liten høyttaler og en vibrasjonsmotor som kan brukes for å gi tilbakemelding. For å fremstille både kompresjons dybde og rytme, kan det vises to sirkler på smartklokken. En ytre for rytme, og en indre for dybde. Fargene på sirklene kunne vært rød/grønn/blå for ”for grunn”/”passe”/”for dyp” og tilsvarende for rytmen. Eksempel sees i figur 7.1.

I forslaget ovenfor er det kanskje ikke så intuitivt å huske hvilken sirkel som betyr hva. En annen modell kan vær å bare ha bakgrunnsfargen bestemt av kompresjonsdybden, og vise rytmen med et tall. Dette blir fremstilt i figur 7.2 Brukeren må da vite hva som er optimal kompresjonshastighet på forhånd. Uansett hvordan dette designes så vil det kreve litt opplæring før bruk. Alternative implementasjoner kan være å gi smartklokken en stemme som sier ”raskere, tregere, dypere, grunnere”.



Figur 7.1: Forslag til GUI for HLR-veileder. A har for lav rytme og for dype kompresjoner. B har passe rytme men for grunne kompresjoner. C har for rask rytme men passelig dype kompresjoner. Hadde hele sirkelen vært grønn hadde både kompresjonsdybde og rytme vært bra.



Figur 7.2: Forslag til GUI for HLR-veileder. A har for grunne kompresjoner, B har passelig og C har for dype. Alle holder bra rytme (mellom 100 og 120).

7.2 Diskusjon

Som kjent burde kompresjonene forklares med et sett skalare variabler for å fungere med maskinlærings-metoder som PLSR. De variablene som ble brukt for å lage PLSR-modellen i denne testen er det undertegnede tenkte kunne ha relevans for kompresjonsdybden. Det kan godt hende at det finnes alternative måter å representere en kompresjon på. De originale dybde-estimatene som er med for hver kompresjon er basert på MAA i denne oppgaven. Som også ble diskutert i kapittel 5.4, er å isolere akselerasjonen i gravitasjonsretningen en mulighet som det ikke ble tid til å teste skikkelig i denne oppgaven. Dette benytter gyromålingene direkte ved kalkulering av kompresjonsdybde. En estimator basert på orienteringen av smartklokken kan hende gir bedre målinger hvor den skadede ligger ortogonalt med gravitasjonskraften.

Om en estimator basert på sensor-fusion hadde gitt bedre resultater hvor den skadede lå flatt, og MAA bedre på vinklede pasienter hadde det vært mulig å benytte begge modellene. Om ytelse viser seg til å ikke være et problem på smartklokken, kunne HLR-assistenten utført full prediksjon med flere modeller samtidig. Den modellen som gir lavest residual for prediksjonene er sannsynligvis den modellen som passer best for situasjonen den befinner seg i. Det samme gjelder for høyre og venstre hånd. Ved bruk av flere modeller kan man spesialtilpasse hver enkelt modell litt bedre enn om man har en stor generell modell. Spørsmålet som går igjen er hvor nøyaktig disse estimatene i grunn trenger å være, men å bruke flere modeller kan være en mulig måte å kontre uforutsette hindre som kan oppstå om man samler et mye større treningsett enn det som er brukt i denne oppgaven.

Det oppstod som kjent et problem med rekkevidden til akselerometeret i smartklokken brukt i denne oppgaven, som virker til å være en nokså standard rekkevidde for både Android og iOS-enheter. Selv om dette påvirker målingene har det trolig ikke all verden å si. Det er to mulige måter hvor clipping kan medføre et problem. Den første muligheten oppstår ved at hjertekompresjoner utføres alt for raskt, men da vil brukeren få tilbakemelding om *for høyt tempo* som forhåpentligvis senker kompresjonshastigheten. Den andre muligheten er at man utfører veldig dype kompresjoner. Om en 78mm kompresjon blir estimert til å være 73, så har heller ikke dette så mye å si, *begge er i kategorien for dype*. Da vil brukeren få tilbakemelding om at kompresjonene er for dype som igjen flytter målingene innenfor operativ rekkevidde til smartklokkens sensorer. Merk at akselerometre med en rekkevidde med mindre enn $\pm$ 2g vil være helt ubrukelig. En rekkevidde på $\pm$ 1g ville åpenbart ikke fungert ettersom det aldri hadde vært mulig å måle mer enn gravitasjonskraften i gravitasjonsretningen. Det er lite tenkelig at det finnes smartklokker med dette på markedet, men disse hadde i så fall ikke vært i stand til å drive en HLR-assistent.

I prosjektoppgaven (1) ble det fundert på hvor virkelighetstro det er å lage en modell for dybdeestimering basert på en gjenopplivingsdukke og ikke med data på mennesker. Sannheten er at det *ikke er mulig* å lage en modell på ekte mennesker. I ekte HLR-situasjoner har man ingen mulighet til å måle den nøyaktige dybden en brystkasse blir komprimert. Det kunne vært mulig med en nøye gjennomtenkt kamera-rigg, men dette er ikke akkurat en prioritet å rigge opp når en persons liv er på spill. I tilegg, så burde det

ikke ha noe som helst å si hva hjerte-kompresjonene blir utført på. Dybde-estimeringsalgoritmene i denne oppgaven baserer seg på dobbeltintegrasjon av akselerasjon. Resultatet av det skal teoretisk bli posisjon, uavhengig av retning til akselerasjonen eller legemet dette blir utført på. Det som har blitt undersøkt og lagd i denne oppgaven er dybde-estimering som har spesialisert seg på et visst arbeidsområde og forventer et visst mønster på akselerasjonsdataen. Alle menneskekropper er forskjellige, og ingen utfører HLR likt. En kompresjonsdybde-estimator vil da alltid være en estimator, ikke en fasit. Matematisk skal det være omtrent det samme å utføre HLR på et menneske som på en gjenopplivingsdukke eller i løse luften, og det antas derfor også at estimatorene foreslått her også vil fungere bra i virkelige situasjoner.

7.3 Videre arbeid

Skulle dette prosjektet blitt tatt videre til faktisk implementasjon, anbefales det så klart med grundigere testing med et større og mer variert test- og trenings-data. Det er viktig å huske på å ikke bare bruke perfekte HLR-sekvenser for å minimere mulighetene for overfitting. Det er heller ikke ønskelig at treningsdataen er for tungt vektet med grunne og dype kompresjoner da det ikke er i dette sjiktet hvor estimeringen trenger å være mest nøyaktig. I disse ytre delene av spekteret er ikke nøyaktigheten så ekstremt viktig, så lenge brukeren blir geleidet i riktig retning. Som diskutert tidligere, er det også mulig å implementere mer enn en estimator. Det kunne for eksmpel vært en estimator optimalisert for sjiktet 10-80mm, og en optimalisert for sjiktet 35-55mm. En grov estimator som luker ut *for grunn/for dyp*, og en mer nøyaktig som skiller grensene nøyere. Det er ikke sikkert det har noe for seg, men det kan hende det er en idé verdt å utforske.

Det fremstår ikke noen åpenbare grunner til å ikke bruke en slik smartklokke-basert HLR-assistent til opplæring på gjenopplivingsdukker som ikke har innebygde målere. Noen av Laerdal Medical sine nye dukker kommer allerede med en dybdesensor og bluetooth innebygget. En idé for fremtidig arbeid kunne vært å implementere sømløs oppkobling mellom en smartklokke-applikasjon og Laerdals originale innmat i dukken. Om disse to enhetene kommuniserer, vil man under opplæring av HLR med smartklokke samtidig kunne samle informasjon fra gjenopplivingsdukken om hvor bra smartklokkens estimatorer er.

Som diskutert tidlig i kapittel 6 er alle HLR-sekvensene til treningsettet og testsettet utført med en viss tilfeldighet. Det ble tatt hensyn for at referansemåleren inni Anne-dukken skulle fungere optimalt, ellers har omstendighetene for HLR-sekvensene vært varierte. Kompresjonene i treningsdataen ble utført av flere personer, og smartklokken har både vært plassert på venstre og høyre håndledd. Kompresjonene har blitt utført både sakte og raskt, grunne og dype. I disse naturlige omstendighetene klarte likevel den beste estimatoren å prediktere kompresjonsdybde med et gjennomsnittlig avvik på $\pm$ 2.6mm på et nytt og ukjent testsett. Resultatene i denne oppgaven antas å være jordnære og representative for faktisk ytelse av en smartklokke-basert HLR-assistent. Det fremstår uten tvil for undertegnede at presisjonen og nøyaktigheten til dybdeestimering på en smartklokke er mer enn bra nok for videreføring av idén.

Litteratur

[1] K. Henrichsen, "Hlr-assistent, en mulighetsanalyse for bruk av smartklokke til å gjenkjenne kvalitet og rytme under hjerte og lunge-redning," 2017.

[2] N. folkehelseinstitutt, "Norsk hjertestansregister," https://www.kvalitetsregistre.no/registers/486/resultater/949, hentet 29. mai 2018.

[3] Statista, "Smartwatch sales statistics," https://www.statista.com/statistics/538237/global-smartwatch-unit-sales/, hentet 29. mai 2018.

[4] S. Rajan, M. Wissenberg, and others", "Association of bystander cardiopulmonary resuscitation and survival," *Circulation*, vol. 134, no. 25, 2016. [Online]. Available: https://doi.org/10.1161/circulationaha.116.024400

[5] R. C. UK, "Resuscitation guidelines," https://www.resus.org.uk/resuscitation-guidelines/adult-basic-life-support-and-automated-external-defibrillation/, hentet 30. mai 2018.

[6] A. Hove, "Kvalifisering av smartklokke til bruk i hlr," 2017.

[7] A. Grünerbl, G. Pirkl *et al.*, "Smart-watch life saver: smart-watch interactive-feedback system for improving bystander cpr," *ISWC 15*, pp. 19–26, 2015.

[8] Y. Song, Y. Chee *et al.*, "Smartwatches as chest compression feedback devices: A feasibility study," *Resuscitation*, vol. 103, pp. 20–23, 2016. [Online]. Available: https://doi.org/10.1016/j.resuscitation.2016.03.014

[9] "Lg g watch r," http://www.lg.com/us/smart-watches/lg-W110-lg-watch-r, hentet 27. mai 2018.

[10] “Little anne product image,” https://www.redcross.org/store/previously-used-laerdal-little-anne-adult-manikin/160903.html, hentet 27. mai 2018.

[11] ProBots, “Probots product images,” https://probots.co.in/index.php?main_page=product_info&products_id=888, hentet 27. mai 2018.

[12] Wikipedia, “Esp8266,” https://en.wikipedia.org/wiki/ESP8266, hentet 10. februar 2018.

[13] STMicroelectronics, “Vl6180x time of flight sensor datablad,” http://www.st.com/en/imaging-and-photonics-solutions/vl6180x.html, hentet 15. februar 2018.

[14] M. Currey, “Mpu6050 product image,” http://www.martyncurrey.com/hc-05-fc-114-and-hc-06-fc-114-part-2-basic-at-commands/, hentet 27. mai 2018.

[15] G. HC, “Hc-06 bluetooth module datablad,” http://www.sgbotic.com/products/datasheets/wireless/hc06_datasheet.pdf, hentet mai 2018.

[16] J. Rowberg, “Open-source device libraries for microcontrollers - mpu6050,” https://www.i2cdevlib.com/, hentet 15. februar 2018.

[17] Android, https://developer.android.com/guide/topics/sensors/sensors_motion, hentet 11 Februar 2018.

[18] B. Inouye, “Accuracy vs precision figure,” https://manoa.hawaii.edu/exploringourfluidearth/physical/world-ocean/map-distortion/practices-science-precision-vs-accuracy, hentet 19. mars 2018.

[19] Umar, “Watch icon,” http://iconshow.me/watch, hentet 8. mai 2018.

[20] LifeguardPro, “Little anne,” https://www.lifeguardpro.com/en/formation/131-maniqui-rcp-little-anne.html, hentet 15. desember 2017.

[21] OpenCV, “Introduction to principal component analysis (pca),” https://docs.opencv.org/trunk/d1/dee/tutorial_introduction_to_pca.html, hentet 8. mai 2018.

[22] Sparkfun, “Sparkfun tof range finder sensor-vl6180,” https://github.com/sparkfun/ToF_Range_Finder_Sensor-VL6180, hentet 15. februar 2018.

[23] MathWorks, “Android coordinate system,” https://www.mathworks.com/help/supportpkg/android/ref/magnetometer.html, hentet 3. mai 2018.

[24] Wikipedia, “Trapezoidal rule,” https://commons.wikimedia.org/w/index.php?curid=8541370, hentet 2. mai 2018.

[25] P. Lawitzki, “Android sensor fusion,” https://www.codeproject.com/Articles/729759/Android-Sensor-Fusion-Tutorial, hentet 19. februar 2018.

[26] Google, “Android motion sensors,” https://developer.android.com/guide/topics/sensors/sensors_motion, hentet 12. mars 2018.

[27] Wikipedia, “Principal component analysis,” https://en.wikipedia.org/wiki/Principal_component_analysis, hentet 6. mai 2018.

[28] H. Martens and M. Martens, *Multivariate Analysis of Quality*. WILEY, 2001.

[29] B. Swarbrick, *Multivariate Data Analysis For Dummies*. WILEY, 2012.

[30] Camo Software, “Unscrambler.”

[31] K. S. Ng, “A Simple Explanation of Partial Least Squares,” http://users.cecs.anu.edu.au/~kee/pls.pdf, hentet 3. mai 2018.

Vedlegg

A Prosjektoppgave

Prosjektoppgaven om samme tema som ble skrevet av undertegnede høsten 2017. Oppgaven er plassert i:

`\HLR-Assistent.pdf`

B Smartklokke applikasjon

LittleAnne, kildekode til android-applikasjonen som ble laget til smartklokken. Ble sist kompilert med Android Studio 2.3.3. Mainfilen til applikasjonen ligger i:

`\LittleAnne\app\src\main\java\henrichsen\littleanne\MainActivity.java`

C Dybdemåler i annedukken

Kildekoden til mikrokontrollen som ble implementert i annedukken. Ble sist kompilert med platform IO v2.1.3 og lastet opp på en esp8266 mikrokontroller. Mainfilen til applikasjonen ligger i:

`\LittleAnneESP\src\main.cpp`

D Logfiler

Her ligger trening og testdataen som ble brukt i resultatdelen av oppgaven. Rådataen fra sensorene for alle kompresjonene ligger i:

```
\logfiler\trainingData.txt
```

TrainingData.txt tilsvarer filen som blir kalt *LogFile.txt* i kapittel 4. Etter at denne filen ble behandlet i python, ble *CprList.txt* laget som inneholder informasjon for hver kompresjon. Dette er filen som ble benyttet til PCA og for å lage PLSR-modellen. Filen bruker ; som skilletegn og skal være enkel å importere og analysere i for eksempel unscrambler.

```
\logfiler\cprListTrain.txt
```

En lignende fil med likt oppsett ble laget for test-settet, som senere ble forsøkt estimert basert på PLSR-modellen laget av treningsettet.

```
\logfiler\cprListTest.txt
```

E Pythonkode

Her er filene som ble brukt for klargjøre den rå sensordataen til analyse i unscrambler. For å kjøre "accelerationAnalysis.py" trengs de python-biliotekene som blir inkludert øverst i filen. Denne koden ble sist kjørt med Spyder og python 3.6 i et Anaconda-miljø. Er alle biblioteker på plass skal kjøring av "accelerationAnalysis.py" kalkulere cprList.txt basert på dataen i exercise.txt

```
\python\accelerationAnalysis.py
```

De to andre filene i python-mappen er hjelpefiler til "accelerationAnalysis.py". "accPlot.py" ble brukt for å plotte grafer av datsettet og "filterFunctions.py" inneholder hjelpefunksjoner.

F Unscrambler

Unscrambler-prosjektet med innlastet trening- og test-data ligger her:

```
\CPR.unsb
```