Masteroppgave

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

Tarjei Steinshamn

Embedded utvikling og design av bølgeestimator

Masteroppgave i Kybernetikk og robotikk
Veileder: Thor Inge Fossen
April 2020

NTNU
Kunnskap for en bedre verden

Oppgavetekst

Oppgaven går ut på å designe og implementere programvare og algoritmer til en prototyp bølgesensor på mikrokontroller. Bølgesensoren er ment til å brukes på en flytende målebøye og skal kunne beregne bølgedata, bølgestatistikk, og bøyens posisjon og orientering i sanntid. Bølgesensoren vil bruke måledata fra en bevegelsessensor (IMU) som inkluderer akselerometer, gyroskop og magnetometer. Data som blir beregnet skal kunne sendes til en datalogger over seriell port.

Bevegelsessensoren Ellipse 2 Micro IMU fra SBG vil være tilgjengelig for bruk til oppgaven, og rådata fra en Fugro SEAWATCH bøye er tilgjengelig for testing av algoritmene for estimering.

Ved valg av mikrokontroller og design av programvare er det viktig at bølgesensoren har lavt strømforbruk, og det er viktig at programvaren er pålitelig, vedlikeholdbar, og er egnet til å kunne videreutvikles med mer funksjonalitet.

Sammendrag

Programvare og algoritmer for estimering av bølgedata og sanntids-bølgehøyde ble implementert på en STM32 ARM-Cortex-M mikrokontroller. Bølgesensoren er laget til bruk på flytende målebøyer, til estimering av bølger og til beregning av posisjon og orientering i sanntid. Ved bruk av DMA (Direkte minnetilgang) og multitasking med FreeRTOS kan bølgesensoren sende sanntidsdata med lav forsinkelse til datalogger over seriell port, samtidig som bølgestatistikk beregnes i bakgrunnen.

Bølgedata som beregnes er bølgespektrum, signifikant bølgehøyde h_{m0}, gjennomsnittlig bølgeperiode t_{m01}, tidsserien av bølgehøyden, og første- og andre fourierkoeffisienter for retningsspektrum. Bølgesensoren mottar akselerometer, gyroskop og magnetometerdata fra en SBG Ellipse 2 Micro IMU, som er koblet til UART på mikrokontroller.

STMicroelectronics eget gratis utviklingsverktøy STM32CubeIDE ble brukt til utvikling og debugging av software, samt til å generere kode for konfigurasjon og initialisering av mikrokontrolleren. STs lavnivå-drivere (Low-layer drivers) ble brukt for konfigurasjon og styring av UART og andre periferienheter på mikrokontrolleren.

Bølgesensoren bruker en AHRS til estimering av bøyens orientering i sanntid ved bruk av sensor-data fra akselerometer, gyroskop og magnetometer. Med bøyens orientering finner man den vertikale akselerasjonen som dobbelt-integreres for å finne bølgehøyde.

AHRS som velges til oppgaven er et Eksplisitt Komplementær Filter, kjent fra Mahony et al. (2008) [18] og bruker en implementasjon fra Hua et al. (2014) [12]. AHRSen testes på rådata fra en Fugro SEAWATCH bøye, til estimering av orientering (rull, stamp og gir-vinkler) og bølgedata/statistikk. AHRSen viser seg å være godt egnet til oppgaven.

De viktigste delene av kildekoden er med i vedlegget på slutten av rapporten. Observer for sanntidsestimering av bølgehøyde ble implementert. Observeren er implementert på tilstandsrom-form og bruker et kalman-filter til beregning av kalman-forsterkning og kovariansmatrisen i hvert tidssteg. Implementasjonen gir gode muligheter for å også estimere posisjon og hastighet horisontalt i sanntid. Valg av observer for sanntids estimering av posisjon og hastighet i sanntid, og akselerometerbias blir diskutert i rapporten.

Abstract

Software and algorithms for estimation of wavedata and real-time waveheight was implemented on a STM32 ARM-Cortex-M microcontroller. The wavesensor is made to be used on a floating weather buoy, for estimation of waves and for estimation of position and attitude in real-time. By using DMA (Direct Memory Access) and multitasking with FreeRTOS, the wavesensor can send real-time data, with low delay to the data-logger, connected with RS-232, and at the same time computes wavestatistics in the background.

The wavedata computed includes the omnidirectional wavespectrum, significant wave-height h_{m0}, average waveperiod t_{m01}, the timeseries of waveheight, and first- and second-order Fourier-coefficients for the directional wavespectrum. The wavesensor receives accelerometer-, attitude rate sensor- and magnetometer-data from a SBG Ellipse 2 Micro IMU, connected to the microcontroller with serial interface.

STMicroelectronics' free of charge integrated development environment STM32CubeIDE was used for development and for debugging the software, and to generate code for configuration and initialization of the microcontroller. The LL-drivers (Low-layer) from ST was used for programming the peripherals of the microcontroller.

With sensordata from the accelerometer, attitude rate sensor and magnetometer, the wavesensor uses an AHRS for estimation of the buoy's attitude in real-time. The buoy's attitude is used for computation of the vertical acceleration, which is integrated twice to find the waveheight.

The AHRS-observer chosen in this thesis is the Explicit Complementary Filter [18], using the implementation from Hua et al. (2014) [12]. The observer was tested on IMU-sensor data from a Fugro SEAWATCH buoy, for estimation of attitude and wavedata. The AHRS proves to be well suited for the task.

The most important parts of the source-code is provided in the appendix of the report. An observer for real-time estimation of waveheight was implemented. The implementation uses the state-space form of the observer, and computes the kalman-gain and covariance-matrix in each timestep. The implementation makes it easy to extend the observer to estimate horizontal position and velocity in addition to the vertical. Choice of observer for real-time estimation of position and velocity in all three dimensions, and estimation of accelerometer-bias is discussed in the report.

Forord

Denne masteroppgaven er skrevet ved Institutt for Teknisk Kybernetikk, NTNU. Jeg ønsker å takke hovedveileder, professor Thor Inge Fossen, som bidro med nyttige råd og innspill underveis, og som alltid var tilgjengelig for samtaler og spørsmål. Jeg vil også takke medveileder Robert Harald Rogne ved Fugro, for inspirerende innspill og diskusjoner. Til slutt vil jeg takke bedriften Fugro i Trondheim, tidligere Oceanor, som har vært samarbeidsbedrift for denne masteroppgaven, og bidro med testutstyr, rådata og lån av SBG-sensoren til oppgaven.

Tarjei Steinshamn

Trondheim, 13. april 2020

Innhold

Nomenklatur

AHRS Orientering- og retningsreferanse system (Attitude and Heading Reference System)

API Programmeringsgrensesnitt (Application Programming Interface)

DMA Direkte minnetilgang (Direct Memory Access)

DSP Digital Signalprosessering (Digital Signal Processing)

ECF Eksplisitt komplementær filter (Explicit Complementary Filter)

FFT Fast Fourier Transform

GNSS Globalt navigasjonssystem (Global Navigation Satellite System)

IDE Integrert utviklingsverktøy (Integrated Development Enviroment)

INS Treghetsnavigasjonssystem (Inertial Navigation System)

MCU Mikrokontroller (Micro Controller Unit)

RMS Kvadratisk gjennomsnitt (Root Mean Square)

RX Mottak (Reception)

ST STMicroelectronics

TTL Transfer-Transistor Logic

TX Sending (Transmission)

UART Universiell Asynkron Mottaker-Sender (Universal Asynchronous Receiver-Transmitter)

1. Innledning

1.1. Bakgrunn

1.1.1. Bølgeanalyse

Innen oseanografien beskriver sjøtilstanden hvordan havoverflaten beveger seg som følge av bølger. Sjøtilstanden karakteriseres av statistisk data, slik som signifikant bølgehøyde eller bølgeperiode. Disse statistikkene vil varierer med tid og sted. Ved måling og analyse av bølger er det vanlig å basere seg på antagelse om at bølgene kan beskrives ut ifra lineær bølgeteori [32], kalt Airy wave theory. Denne innebærer at bølgene består av summen av mange sinus-bølger med forskjellige frekvenser, amplituder og retninger.

Målebøyer og bølgeradar er eksempler på instrumenter man kan bruke til å måle sjøtilstanden med. Det finnes flere leverandører av målebøyer og bølgesensorer på markedet. Eksempler er SEAWATCH-bøyer fra Fugro, WaveDroid fra Obscape, Waverider-bøyer og bølgesensorer fra Datawell BV, og bølgesensorer fra Nortek. Kunnskap om sjøtilstanden er relevant ved design av konstruksjoner ved kysten og på havet, slik som for eksempel oljeplattformer. Bølgeestimering er også relevant ved forskning på bølger eller ved værmelding til skipstrafikk. De fleste av disse bølgemålerne er basert på bevegelsen til en flytende målebøye, mens strømmålerne fra Nortek, bruker trykkmåling til å måle bølger.

To metoder er vanlige for å analysere bølger: nullkrysnings-metoden og spektral-analyse [6, 20]. Begge disse metodene går hovedsaklig ut på å analysere bølgehøyden, definert som den vertikale posisjonen til havoverflaten på en bestemt lokasjon. Nullkrysnings-metoden er en tidsserieanalyse av bølgehøyden for en bestemt periode. Fra denne tidsserien beregner man ulike bølgestatistikker slik som signifikant bølgehøyde og periode. Analysen baserer seg på at man identifiserer hver enkelt bølge ved å se på når bølgehøyden går fra negativ til positiv (krysser null-punktet oppover). Ved Spektral-analyse bruker man bølgespekteret som er energispekteret av bølgehøyde for en bestemt periode. Energispekteret er gitt av absoluttverdi til frekvensspekteret opphøyd i andre. Fra bølgespekteret kan man avlede ulike statistikker, inkludert signifikant bølgehøyde og periode. Først beregner man gjerne momentene av spekteret som er gitt ved $m_k = \int f^k S(f) df$, $k \in \mathbb{Z}$ der $S(f)$ er bølgespekteret og f er frekvensen, og så kan man beregne bølgestatistikkene fra momentene. For eksempel er signifikant bølgehøyde gitt ved $h_{m0} = 4.0\sqrt{m_0}$.

Bølgespekteret kan også beskrives ved $E(f, \theta)$. Parameteren θ angir bølgeretning, som gir en ekstra dimensjon i forhold til $S(f)$. $E(f, \theta)$ kalles retningsspekteret. Bølgespek-

teret $S(f)$ kan uttrykkes fra retningsspekteret $E(f,\theta)$ ved å integrere over θ: $S(f) = \int_0^{2\pi} E(f,\theta)d\theta$.

Både ved tidsserie- og spektralanalysen må man som sagt velge en bestemt tidsperiode av bølgehøyden til å beregne statistikken utifra. Perioden bør være tilstrekkelig lang slik at man begrenser tilfeldigheters påvirkning på statistikken. Men perioden må ikke være så lang at sjøtilstanden har forandret seg for mye. Nortek [20] anbefaler at perioden er lang nok til å inneholde 100 hendelser av det man ønsker å måle, slik at hvis den lengste bølgen man ser etter har periode på 10 sekunder, så blir perioden $10s \cdot 100 = 1000s$ som tilsvarer cirka 17 minutter. De beskriver at den lengste måle perioden som kan brukes er 3 timer.

Denne oppgaven er basert på flytende bøyer som utgangspunkt for å måle bølger. Hvis man antar at flytebøya følger havoverflaten og står i en fast posisjon horisontalt, så vil bølgehøyden tilsvare hiv (med motsatt fortegn), som defineres som bøyens vertikal-posisjon i NED-ramma (North-East-Down).[1] For å finne bølgehøyden trenger man altså da bare å finne vertikal-posisjonen til bøya. Flytende bøyer som også følger havoverflatens helning kalles rull-stamp-hiv bøyer [29]. Denne egenskapen bruker man når man vil beregne retningsspektrum. Det finnes også andre typer flytebøyer, der bøyens tilting ikke følger helningen på havoverflaten, slik at beregning av bølgeretningen blir forskjellig. I hvilken grad en rull-stamp-hiv bøye følger havoverflaten perfekt, vil avhenge av tregheten og størrelsen til bøya i forhold til periode og lengde på bølgene. En stor og tung målebøye vil ikke kunne måle like hurtige bølger som en bøye som er mindre og lettere. Fortøyningen vil også kunne påvirke i hvilken grad bøya følger havoverflaten.

Bølgeretningsspekteret

Retningsspekteret $E(f,\theta)$ kan også uttrykkes som $E(f,\theta) = S(f) \cdot D(f,\theta)$ der $S(f)$ er bølgespekteret. Siden

$$S(f) = \int_0^{2\pi} E(f,\theta)d\theta = S(f) \int_0^{2\pi} D(f,\theta)d\theta$$

får vi at $\int_0^{2\pi} D(f,\theta)d\theta = 1$.

$D(f,\theta)$ representerer spredningsspekteret og ved Fourier-ekspansjon kan $D(f,\theta)$ uttrykkes som

$$D(f,\theta) = \frac{1}{\pi}\Big[\frac{1}{2} + \sum_n \{a_n \cos n\theta + b_n \sin n\theta\}\Big]$$

[1] Dette forutsetter riktignok at origo til bøya ligger i havoverflaten. Hvis bøya for eksempel har en høy mast og man måler posisjon utifra toppen av masta så vil tilting gi en relativ endring av posisjonen. Men dette er lett å kompensere for hvis man kjenner til orienteringen: den relativ posisjonen til masta er gitt av en fast rom-vektor, og endringen i posisjon som man får ved å rotere denne vektoren må man trekke ifra.

Her er $a_n(f)$ og $b_n(f)$ funksjoner av frekvensen f. Analysen av retningsspekteret baserer seg på å estimere første- og andreordens fourierkoeffisienter a_1, b_1, a_2 og b_2. Bølgestatistikkene beregnes deretter utifra disse. Eksempler på slike parametre er gjennomsnittlig bølgeretning gitt av $\theta_{m1}(f) = \arctan 2\Big(b_1(f), a_1(f)\Big)$ og retningsspredningen gitt av $\sigma(f) = \sqrt{2(1 - r_1(f)}$ der $r_1 = \sqrt{a_1^2 + b_1^2}$.

Vi har estimater av a_1, b_1, a_2 og b_2, og vet at $\int_0^{2\pi} D(\theta)d\theta = 1$, men dette er ikke nok til å finne $D(\theta)$. Hvis man vil beregne et estimat for retningsspekteret $E(f, \theta)$ må man basere seg på antagelser. MEM-metoden (Maximum Entropy Method) som antar at entropien til spredningsspekteret $D(\theta)$ er maksimal, er en populær måte å finne et estimat for $D(\theta)$ og dermed $E(f, \theta) = S(f) \cdot D(f, \theta)$, se [16].

For å beregne a_1, b_1, a_2 og b_2 for en rull-stamp-hiv bøye bruker man bøyens helning i nord og øst retning, i tillegg til bøyens hiv (vertikal posisjon).[2] Beregningene kan gjøres i frekvensdomenet, der $N(f)$, $E(f)$ og $H(f)$ er frekvens-spekteret til henholdsvis nord-tilting, øst-tilting og p_z. Utregningene blir slik[15]:[3][4]

$$
\begin{aligned}
a_1 &= \frac{Q_{HN}}{\sqrt{C_{HN}(C_{NN} + C_{EE})}} \\
b_1 &= \frac{Q_{HE}}{\sqrt{C_{HH}(C_{NN} + C_{EE})}} \\
a_2 &= \frac{C_{NN} - C_{EE}}{C_{NN} + C_{EE}} \\
b_2 &= \frac{2C_{NE}}{C_{NN} + C_{EE}}
\end{aligned}
$$

der Q og C er definert slik at $A^*B = C_{AB} - iQ_{AB}$, der * angir den kompleks-konjugerte. Fra matematikken vet vi at $\mathcal{F}\{a \star b\} = \mathcal{F}\{a\}^* \cdot \mathcal{F}\{b\}$, der $\mathcal{F}$ angir Fourier-transform og $\star$ angir korrelasjon. Derfor vil for eksempel $-Q_{HN}$ tilsvare imaginær-verdien til korrelasjonen av p_z og nord-tilting i frekvensdomenet: $-Q_{HN} = \text{img}(H^*N) = \text{img}\{\mathcal{F}\{h \star n\}\}$.

1.2. Problembeskrivelse

Denne oppgaven går ut på å designe og implementere programvare og algoritmer til en prototyp bølgesensor på mikrokontroller. Bølgesensoren er ment til å brukes på en

[2] Bøyens helning i nord eller øst uttrykkes som sinus av helningsvinkelen

[3] For å beregne a_1, b_1 ser man på hvordan tiltingen korrelerer med hiv med 90° faseforskyvning. Det er fordi helningen på en bølgekomponent, representert ved en sinus-bølge, er en sinus-funksjon med 90° faseforskyvning. I frekvensdomenet tilsvarer denne korrelasjonen imaginær-verdien, altså kvadraturspekteret Q.

[4] a_1 og b_1 sier noe om gjennomsnittlig bølgeretning og bølgespredningen, men hvis man har bølgekomponenter med samme frekvens som går i motsatt retning, så vil ikke disse kunne beskrives av a_1 og b_1. Slike bølgekomponenter vil derimot beskrives av de andre-ordens fourier koeffisientene a_2 og b_2.

flytende målebøye. Målebøyer måler gjerne mange forskjellige ting og har flere forskjellige sensorer. Estimering av orientering, posisjon og hastighet til målebøyen kan være relevant for bevegelseskompenesering av andre sensorer. Et eksempel er vindmåling med lidar-sensor. Det kan da være relevant å estimere orientering og hastighet i alle tre retninger, for å bedre kunne estimere vind og turbulens. I [13] ble dette gjort på en Fugro SEAWATCH bøye med en bevegelsesobserver (MRU) fra Norsub. Både en bevegelsesobserver og en vanlig bølgesensor for bøyer, baserer seg på målinger fra en IMU-sensor for estimeringen, men det selges ikke noen sensorer som kombinerer disse to type estimatorene i én sensor. I denne oppgaven skal estimeringen av orientering, posisjon og hastighet, og estimering av bølgedata, kombineres i én sensor basert på samme IMU.

For målebøyer vil det normalt sett ikke være så viktig med høy grad av responstid, ved for eksempel bevegelsesestimering, fordi man bare gjør målinger, slik at det ikke er noen tilbakekobling i sanntid av målingene slik det er ved et typisk styringssystem. Det kan likevel være relevant å estimere i sanntid fordi ved å gjøre databehandling på målingene med en gang, slipper man å lagre og organisere store mengder med rådata. Det kan også være relevant å estimere bølger i sanntid for eksempel for varsling til surfere når det kommer store bølger. Dessuten kan en kombinert bevegelsesobserver og bølgesensor være relevant for andre applikasjoner enn målebøyer. For eksempel kan det tenkes at estimatene av bølgeparametrene kan brukes for å gi bedre bevegelsesestimering eller bedre regulering i systemer der høy responstid er relevant. For eksempel i [10] brukes bølgeamplituden og bølgefrekvensen som parametre til en adaptiv hiv-observer. Et annet eksempel er [3], der sjøtilstanden estimeres fra hiv, rull og stamp for bruk til dynamisk posisjonering (DP) for marine fartøyer. At oppgaven tar utgangspunkt i en målebøye kan være greit, fordi det er et enkelt utgangspunkt som man lett kan bygge videre på.

Ved valg av mikrokontroller og design av programvaren blir det vektlagt at bølgesensoren skal ha lavt strømforbruk og være pålitelig og vedlikeholdbar. Dette er fordi målebøyer har begrensede muligheter for energilagring og skal fungere over lang tid uten tilsyn. Tilsyn og eventuelle reparsjoner er gjerne kostbart fordi målebøyen er plassert ute på sjøen, og må dermed fraktes til land. Bølgesensoren skal kunne kommunisere med en datalogger over seriell port.

1.2.1. Tidligere arbeid

Standard bølgeanalyse utifra posisjons- og orienterings-data brukes i mange ulike målebøyer på markedet, og det finnes det mange forskjellige leverandører av slike, men det ble ikke funnet noen bølgesensor som kan estimere både bølgedata og bevegelsesdata i sanntid. Det ble heller ikke funnet noen åpent tilgjengelig implementasjon av en bølgesensor.

Det finnes en god del kilde-kode på nett for implementering av AHRS. Mye er basert på c-kildekoden som Madgwick S. gav ut med sin publisering [17]. Se for eksempel [34] for c-kode og [19] for python-kode.

Det finnes mange forskjellig kode-eksempler på internett som implementerer kommunikasjon over UART på mikrokontrollere. For MCU-familien STM32 fra STMicroelectronics har ST et utvalg med eksempler i verktøy-settet STM32Cube [22].

X-CUBE-MEMS1 [23] er en software-pakke fra ST med kildekode fra ulike applikasjoner, deriblant en som heter DataLogFusion, som implementerer en AHRS med to-veis kommunikasjon over UART, ved bruk av HAL(Hardware Abstraction Layer)-driverbiblioteket fra ST. DMA brukes for UART Rx, men ikke for UART Tx, og eksempelet har ikke støtte for OS/multitasking. AHRS-algoritmen er lukket-kildekode. Applikasjonen bruker en enkel kommunikasjonsprotokoll som også blir brukt i denne oppgaven, se A.2.

Github-siden [11] har eksempler som implementerer sending eller mottak over UART på STM32 ved bruk av LL(Low-Layer)-driverbiblioteket til ST og med bruk av FreeRTOS. I potensiell svakhet med implementeringen er at sending (TX) over UART ikke bruker mutex/semafor, slik at når en tråd venter på at UART TX skal bli ledig, så vil andre tråder med lavere prioritet bli blokkert.

1.3. Hovedbidrag

- Vurderer valg av AHRS og bevegelsesobserver for bruk til bølgeestimering fra flytende målebøye
- Demonstrerer implementasjon av kombinert AHRS/posisjons-observer og bølgesensor på mikrokontroller.
- Vurderer valg av MCU til oppgaven i forhold til prosesseringskraft, strømforbruk og minneforbruk.
- Demonstrerer hvordan man implementerer effektiv toveis-kommunikasjon over UART på STM32 mikrokontroller ved bruk av FreeRTOS/multitasking, DMA, LL-driver (register-nivå) og kodegeneratoren til STM32CubeIDE (gratis utviklingsverktøy).
- Effekten av akselerometerbias på estimering av bølger og orientering, undersøkes, og mulige måter å estimere eller kompensere for biasen på, diskuteres.

1.4. Organisering av rapporten

I kapittel 2 beskrives valg orienterings- og posisjonsestimator, samt metoder for bølgeestimering. Valgene som blir gjort blir begrunnet og diskutert.

I kapittel 3 beskrives valg og konfigurasjon av mikrokontroller og implementering av programvaren. Designvalg for programvare blir også diskutert.

I kapittel 4 under seksjon 4.1 blir bølgesensoren testet, og resultatene blir beskrevet.

I seksjon 4.2 blir orienterings- og bølgeestimatoren testet på rådata fra en Fugro SEAWATCH bøye. Justeringsfaktorer for orienteringsobserveren blir testet og valgt, og effekten av akselerometerbias for estimeringen blir undersøkt.

Kapittel 5 og 6 diskuterer resultatene og gir forslag til videre arbeid med bølgesensoren.

2. Estimering av bølgedata, posisjon og orientering

For å beregne bevegelsen til en målebøye, bruker man en bevegelsessensor (IMU) , slik som akselerometer og gyro-sensor, som måler vinkelhastighet. Et treghetsnavigasjonssystem (INS) er et system som bruker bevegelsessensor til å beregne posisjon og orientering. Et treghetsnavigasjonssystem kan implementeres på flere forskjellige måter. For eksempel så bruker Datawell sine bølgesensorer en gravitasjonsbasert stabiliserings-plattform med et 1-akse akselerometer. Stabiliseringsplattformen er et mekanisk system som gjør at akselerometeret alltid peker vertikalt. For å finne bølgehøyden dobbeltintegrerer man den vertikale akselerasjonen. Stabiliseringsplattformen måler forøvrig også rull og stamp. Et slik system kan være meget nøyaktig, men er relativt dyrt og ikke vanlig blant andre typer bølgesensorer.

En utbredt metode for INS-systemer blant flytebøyer og andre farkoster er å bruke et strapdown-system. Et strapdown system består av en IMU og en dataprosessor. I et strapdown system er gyrosensoren en rategyro, IMUen blir montert fast til fartøyet og prosessoren beregner posisjon og orientering. For å beregne kompass-retningen bruker man gjerne et magnetometer. I motsetning til det mekaniske systemet fra Datawell er et strapdown-system mer avhengig av relativt hurtig dataprosessering. I denne oppgaven brukes et strapdown-system inkludert magnetometer.

Et INS-system kan forbedres ved å bruke et navigasjonssatellittsystem (GNSS). Ofte bruker man da posisjonsmålingene fra GNSS som referanse for å gi bedre og driftfrie posisjonsestimater. En av de tre bølgesensorene til Datawell for eksempel er basert på GNSS-teknologi. For en mer nøyaktig kompassretningen kan man dessuten bruke 2 GNSS-mottakere, der de to antennen står i avstand til hverandre.

Et system som bare beregner orienteringen kalles en AHRS. Det å estimere orientering med et strapdown-system er et ulineært estimeringsproblem. Prinsippet går ut på at man integrerer gyromålingen for å finne orienteringen. Over tid vil man integrere opp målestøy slik at orienteringen vil drifte, som betyr at man over tid får store avvik. Derfor bruker man gravitasjonsvektoren målt av akselerometeret og jordens magnetfelt målt av magnetometeret som referanse. Gravitasjonsvektoren er alltid vertikal, og er nødvendig for å beregne rull og stamp, mens magnetfeltet peker mot nord, og er nødvendig for å beregne gir (horisontal retning). Å bruke et Eksplisitt Komplementær Filter (ECF)[18] er en enkel og effektiv metode for å implementere en slik AHRS. Når man har estimert orienteringen med en AHRS, kan man rotere vektorer fra Body-ramma til Ned-ramma. Den vertikale akselerasjonen finner man da ved z-komponenten i NED-ramma, til summen av

akselerometermålingen og gravitasjonsvektoren. Man må legge til gravitasjonsvektoren, fordi akselerometeret måler den spesifikke kraften, som tilsvarer akselerasjonen pluss den spesifikke normalkraften, der normalkraften er motsatt av gravitasjonskraften. Når man har funnet den vertikale akselerasjonen, kan man enkelt finne hiv/bølgehøyden ved å dobbeltintegrere og høypassfiltrere. Høypassfiltrering må til fordi man over tid vil integrere opp målestøy, som gjør at hiv vil drifte. Dobbeltintegrering og høypassfiltrering kan gjøres i frekvensdomenet, eller tidsdomenet.

En begrensning med Mahony-observeren er at man antar at akselerometer-målingene er fri for bias, så hvis denne forventes å være stor, må den også estimeres. Akselerometerbias kan estimeres som en del av posisjons-estimeringen. Et trivielt eksempel er 1-akse akselerometeret på Datawell sin bølgesensor: estimert akselerometerbias blir lik middelverdi av målt akselerasjon, fordi middelverdien av den reelle akselerasjonen vertikalt vil konvergere mot 0. For et strapdown-system blir det mer komplisert fordi akselerometerene roterer, og fordi den horisontale posisjonen til målebøyer gjerne drifter en del. I motsetning til den vertikale bevegelsen, er denne driftingen lavfrekvente bevegelser der akselerasjonen ikke konvergerer like raskt mot 0. Vanligvis vil en posisjonsobserver for et strapdown-system være designet for fartøyer som beveger seg over lange avstander i betydelige hastigheter, sammenlignet med en målebøye, og da blir det enda vanskeligere å estimere akselerometerbiasen med en INS som bare bruker IMU, og derfor vil man typisk bruke et INS/GNSS system. Siden bevegelsene til en målebøye er mer begrenset vil det være bedre muligheter for å kunne estimere biasen uten GNSS.

En posisjonsobserver som estimerer akselerometerbias i et strapdown-system, vil være en ulineær observer. Når man designer en slik observer vil det ofte være hensiktsmessig å lage én enkelt observer for både posisjon og orientering. Det er fordi estimeringen av disse to er gjensidig avhengig: for eksempel så har akselerometerbias innvirkning på både posisjons- og orienteringsestimatene, og orienteringsestimatet inngår i posisjonsestimeringen. Et Extended Kalman filter vil kunne egne seg bra til et slikt estimeringsproblem. Det er dessuten en fordel med Extended Kalman filter i forhold til ECF-observeren at man beregner kovariansen til tilstandene i observeren, noe som potensielt gir bedre estimater og som også gir informasjon om usikkerheten til estimatene. En ulempe med å bruke Kalman filter er at det er mer beregningskrevende.

Bølgeestimatoren i denne oppgaven vil bli designet uten bruk av GNSS. Hvorvidt det er nødvendig å estimere biasen vil være avhengig av hvor nøyaktige og stabile akselerometere som brukes. Før en IMU tas i bruk bør den være kalibrert. For et ukalibrert akselerometer eller gyro, vil den riktige verdien typisk være en ulineær funksjon av målesignalet, som dessuten er temperaturavhengig. I tillegg er målingen gjerne feil pga skjevstilling av de tre aksene til sensoren. For å få nøyaktige målinger er det nødvendig at IMUen kalibreres. For eksempel er IMUen som brukes i denne oppgaven kalibrert fra leverandør. Kalibreringen er lagret i IMUen som bruker en mikroprosessor til å omregne det ukalibrerte målesignalet til en kalibrert verdi. Derfor er en slik IMU mer strømkrevende og mye dyrere enn en ukalibrert sensor. En slik sensor vil naturligvis ha mye mindre bias enn en ukalibrert. Men Kalibreringen kan ikke fjerne bias som varierer over tid. For SBG-sensoren er dette oppgitt som «In run bias instability» og «One year bias

stability».

Som forklart så vil akselerometerbias kunne medføre at en observeren blir vesentlig mer komplisert, og hvis nøyaktigheten til IMUen er god nok og man ikke har noe INS-system som referanse, er det kanskje ikke hensiktsmessig å legge vekt på denne biasen i design av bølgeestimator. I denne oppgaven vil vi derfor ta utgangspunkt i den enkle måten å estimere orientering og z-posisjon på. Orienteringsestimeringen vil derfor baseres på ECF-observeren.

2.1. Referansesystem, eulervinkler og konvensjoner

Referansesystemet som brukes i denne oppgaven for jordramma, kalles NED (engelsk: North-East-Down), der x-, y- og z-aksen tilsvarer henholdsvis nord, øst og ned-retning. NED-konvensjonen er en vanlig konvensjon å bruke for fartøyer. Dette referansesystemet brukes ved estimering, men når bølgehøyden representeres og omtales brukes typisk oppover som positiv retning, fordi dette virker mer naturlig. For body-ramma, som er referanseramma til bøyen, brukes også z-aksen, som retning nedover, mens x- og y-aksene går horisontalt. Ved representasjon av orientering, brukes ofte euler-vinkler, som kan flere forskjellige konvensjoner. Konvensjonen som blir brukt her, er den som er vanlig å bruke for fartøyer på sjøen og for fly. Med denne konvensjonen blir rotasjonsmatrisen for å rotere fra NED- til body-ramma ved bruk av euler-vinkler slik:

$$\boldsymbol{R}_b^n(\boldsymbol{\Theta}_{nb}) = \begin{bmatrix} \mathrm{c}\psi\mathrm{c}\theta & -\mathrm{s}\psi\mathrm{c}\phi + \mathrm{c}\psi\mathrm{s}\theta\mathrm{s}\phi & \mathrm{s}\psi\mathrm{s}\phi + \mathrm{c}\psi\mathrm{c}\phi\mathrm{s}\theta \\ \mathrm{s}\psi\mathrm{c}\theta & \mathrm{c}\psi\mathrm{c}\phi + \mathrm{s}\phi\mathrm{s}\theta\mathrm{s}\psi & -\mathrm{c}\psi\mathrm{s}\phi + \mathrm{s}\theta\mathrm{s}\psi\mathrm{c}\phi \\ -\mathrm{s}\theta & \mathrm{c}\theta\mathrm{s}\phi & \mathrm{c}\theta\mathrm{c}\phi \end{bmatrix}$$

der c er s er forkortelse for cosinus og sinus. Rull, stamp og gir-vinkel er representert ved henholdsvis ϕ, θ og ψ.

Høyrehåndsregelen gjelder for NED-ramma og brukes også for body-ramma og ved rotasjon.

Når termen «hiv» brukes i oppgaven, så menes vertikal posisjon i forhold til havoverflata, altså z-posisjon i NED-ramma. Hiv tilsvarer derfor bølgehøyden, bortsett fra at fortegnet er motsatt. Slik begrepene hiv og bølgehøyde brukes i denne oppgaven kan de tenkes på som like.

Ved beregninger brukes for det meste enhetskvaternioner for å representere rotasjon. Rotasjonsmatrisen for enhetskvaternioner er derfor mye brukt, og den ser slik ut:

$$\boldsymbol{R}_b^n(\boldsymbol{q}) = \begin{bmatrix} 1 - 2(\varepsilon_2^2 + \varepsilon_3^2) & 2(\varepsilon_1\varepsilon_2 - \varepsilon_3\eta) & 2(\varepsilon_1\varepsilon_3 + \varepsilon_2\eta) \\ 2(\varepsilon_1\varepsilon_2 + \varepsilon_3\eta) & 1 - 2(\varepsilon_1^2 + \varepsilon_3^2) & 2(\varepsilon_2\varepsilon_3 - \varepsilon_1\eta) \\ 2(\varepsilon_1\varepsilon_3 - \varepsilon_2\eta) & 2(\varepsilon_2\varepsilon_3 + \varepsilon_1\eta) & 1 - 2(\varepsilon_1^2 + \varepsilon_2^2) \end{bmatrix}$$

der enhetskvaternionen er representert ved $\boldsymbol{q}=\begin{bmatrix} \eta & \varepsilon_1 & \varepsilon_2 & \varepsilon_3 \end{bmatrix}^T$. Enhetskvaternioner er kvaternioner med norm 1, slik at $\|\boldsymbol{q}\| = 1$ der $\|\cdot\|$ betegner normen.

2.2. Eksplisitt komplementær filter (ECF)

2.2.1. Standard implementasjon

ECF-observeren kan i prinsippet brukes med mange forskjellige referansemålinger[18]. For vår estimator bruker vi et akselerometer og et magnetometer og algoritmen på kontinuerlig form blir da slik:

$$\begin{aligned}
\boldsymbol{\epsilon}^b_{\text{mes}} &= k_1 \boldsymbol{u}^b \times \hat{\boldsymbol{u}}^b + k_2 \boldsymbol{v}^b \times \hat{\boldsymbol{v}}^b \\
\dot{\hat{\boldsymbol{b}}}^b_{\text{gyro}} &= -k_3 \boldsymbol{u}^b \times \hat{\boldsymbol{u}}^b - k_4 \boldsymbol{v}^b \times \hat{\boldsymbol{v}}^b \\
\dot{\hat{\boldsymbol{q}}} &= \boldsymbol{T}_q(\hat{\boldsymbol{q}})\Big(\boldsymbol{\omega}^b_{\text{imu}} - \hat{\boldsymbol{b}}^b_{\text{gyro}} + \boldsymbol{\epsilon}^b_{\text{mes}}\Big)
\end{aligned}$$

der $k_i > 0$. $\boldsymbol{q}$ er enhetskvaternionen som representerer orienteringen. Målingene er normalisert slik at $\boldsymbol{u}^b = \boldsymbol{a}^b_{\text{imu}} / \left\|\boldsymbol{a}^b_{\text{imu}}\right\|$ og $\boldsymbol{v}^b = \boldsymbol{m}^b_{\text{mag}} / \left\|\boldsymbol{m}^b_{\text{mag}}\right\|$. $\hat{\boldsymbol{u}}^b$ betegner referansevektoren for akselerometermålingen i body-ramma, som er den normaliserte gravitasjonsvektoren med motsatt fortegn (normalkraften normalisert):

$$\hat{\boldsymbol{u}}^b = \boldsymbol{R}^b_n(\boldsymbol{q}) \begin{bmatrix} 0 & 0 & -1 \end{bmatrix}^T$$

$\hat{\boldsymbol{v}}^b$ betegner referansevektoren for magnetfeltet i body-ramma. Dette vil egentlig være det normaliserte magnetfeltet til jorda. Horisontalt peker magnetfeltet mot nord, men det har samtidig en z-komponent og siden vi ikke kjenner til vinkelen i forhold til horisonten bruker man ofte følgende triks:

$$\hat{\boldsymbol{v}}^b = \boldsymbol{R}^b_n(\boldsymbol{q}) \begin{bmatrix} \sqrt{(\boldsymbol{m}^n_{\text{mag},x})^2 + (\boldsymbol{m}^n_{\text{mag},y})^2} & 0 & \boldsymbol{m}^n_{\text{mag},z} \end{bmatrix}^T$$

Der $\boldsymbol{m}^n_{\text{mag}} = \boldsymbol{R}^b_n(\boldsymbol{q})^T \boldsymbol{m}^b_{\text{mag}}$. Vi har ellers at $\boldsymbol{T}_q(\boldsymbol{q})\boldsymbol{x} = \frac{1}{2}\boldsymbol{q} \otimes \begin{bmatrix} 0 & \boldsymbol{x} \end{bmatrix}^T$ der $\otimes$ betegner kvaternion-multiplikasjon. $\boldsymbol{R}^b_n(\boldsymbol{q})$ kan også beskrives slik [31]: $\boldsymbol{R}^b_n(\boldsymbol{q})\boldsymbol{x} = \boldsymbol{q} \otimes \boldsymbol{x} \otimes \boldsymbol{q}^*$, der * betegner den konjugerte .

Ved å søke på internett kan man finne c- eller python-kode der ECF-observeren er implementert på den måten som ble vist her [34, 19]. En annen versjon av ECF-observeren er beskrevet i en artikkel av Hua et al. [12].

2.2.2. Implementasjonen fra Hua et al. (2014)

Det interessante med denne observeren er at den skal gi bedre dekobling mellom rull, stamp og gir. Det vil bety at innsvingning av gir ved oppstart, eller ved forstyrrelser på magnetfeltet, vil i liten grad påvirke rull eller stamp estimatet. Siden man ikke trenger gir for å estimere hiv, så kan dette være nyttig for en flytebøye, fordi hiv-estimatet blir mer robust for forstyrrelser og konvergerer rasker ved oppstart. Observeren til Hua et al. vil derfor bli brukt i denne oppgaven, og det vil bli sammenlignet med den forrige implementasjonen. En annen forskjell ved denne observeren er at det blir brukt anti-windup på integraleffekten til observeren.

Observeren til Hua et al. på kontinuerlig form blir som følger:

$$
\begin{aligned}
\boldsymbol{\epsilon}^b_{\text{mes}} &= k_1 \boldsymbol{u}^b \times \hat{\boldsymbol{u}}^b + k_2 \hat{\boldsymbol{u}}^b (\hat{\boldsymbol{u}}^b)^T \boldsymbol{v}^b \times \hat{\boldsymbol{v}}^b \\
\dot{\hat{\boldsymbol{b}}}^b_{\text{gyro}} &= -k_b \Big(\hat{\boldsymbol{b}}^b_{\text{gyro}} - \text{sat}_\Delta(\hat{\boldsymbol{b}}^b_{\text{gyro}}) \Big) - k_3 \boldsymbol{u}^b \times \hat{\boldsymbol{u}}^b - k_4 \boldsymbol{v}^b \times \hat{\boldsymbol{v}}^b \\
\dot{\hat{\boldsymbol{q}}} &= \boldsymbol{T}_q(\hat{\boldsymbol{q}}) \Big(\boldsymbol{\omega}^b_{\text{imu}} - \hat{\boldsymbol{b}}^b_{\text{gyro}} + \boldsymbol{\epsilon}^b_{\text{mes}} \Big)
\end{aligned}
$$

der $\boldsymbol{v}^b = \Big(\boldsymbol{\Pi}_{\boldsymbol{u}^b} \boldsymbol{m}^b_{\text{mag}} \Big) / \Big\| \boldsymbol{\Pi}_{\boldsymbol{u}^b} \boldsymbol{m}^b_{\text{mag}} \Big\|$ og $\hat{\boldsymbol{v}}^b = \boldsymbol{R}^b_n(\boldsymbol{q}) \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$. Her er $\boldsymbol{\Pi}_{\boldsymbol{x}} \coloneqq |\boldsymbol{x}|^2 \boldsymbol{I}_3 - \boldsymbol{x}\boldsymbol{x}^T, \forall \boldsymbol{x} \in \mathbb{R}^3$, som gir den ortogonale projeksjonen på planet ortogonalt til $\boldsymbol{x}$. Vi har også at $k_b, \Delta > 0$, og $sat_\Delta(\cdot)$ er en enkel metningsfunksjon definert ved $sat_\Delta(\boldsymbol{x}) \coloneqq \boldsymbol{x} \min(1, \Delta / |\boldsymbol{x}|)$. Ellers er observeren lik den forrige.

2.3. Posisjonsestimering i sanntid

2.3.1. Estimering av posisjon og akselerometerbias

I boken Fossen (2011)[8], finner man en enkel posisjons- og hastighetsobserver, som inkluderer estimering av akselerometerbias. Den observeren er designet for estimering av posisjon og hastighet der man bruker en posisjonsreferansemåling, som typisk vil være en posisjonsmåling med GNSS. I denne oppgaven tar man utgangspunkt i at man bare har en IMU, og dermed ingen posisjonsreferanse. Man kan derfor ikke estimere absoluttposisjonen, men bare relativ posisjon. Posisjonsobserveren fra Fossen (2011) kan da brukes ved at man setter posisjonsmålingen som lik 0. Ligningen for observeren blir data:

$$
\begin{aligned}
\dot{\hat{\boldsymbol{p}}} &= \hat{\boldsymbol{v}} - \boldsymbol{K}_1 \hat{\boldsymbol{p}} \\
\dot{\hat{\boldsymbol{v}}} &= \boldsymbol{R}^n_b (\boldsymbol{f}^b_{\text{imu}} - \hat{\boldsymbol{b}}) + \boldsymbol{g} - \boldsymbol{K}_2 \hat{\boldsymbol{p}} \\
\dot{\hat{\boldsymbol{b}}} &= -\boldsymbol{K}_3 \boldsymbol{R}^n_b (\boldsymbol{\Theta})^T \cdot \hat{\boldsymbol{p}}
\end{aligned}
$$

På tilstandsrom-form blir dette:

$$\begin{bmatrix} \dot{\hat{\boldsymbol{p}}} \\ \dot{\hat{\boldsymbol{v}}} \\ \dot{\hat{\boldsymbol{b}}} \end{bmatrix} = \begin{bmatrix} -\boldsymbol{K}_1 & \boldsymbol{I} & \boldsymbol{0} \\ -\boldsymbol{K}_2 & \boldsymbol{0} & -\boldsymbol{R}_b^n(\boldsymbol{\Theta}) \\ -\boldsymbol{K}_3\boldsymbol{R}_b^n(\boldsymbol{\Theta})^T & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \cdot \begin{bmatrix} \hat{\boldsymbol{p}} \\ \hat{\boldsymbol{v}} \\ \hat{\boldsymbol{b}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{R}_b^n(\boldsymbol{\Theta})\boldsymbol{f}_{\text{imu}}^b + \boldsymbol{g} \\ \boldsymbol{0} \end{bmatrix} \tag{2.1}$$

Man setter målingene $\boldsymbol{z} = \boldsymbol{p} = \boldsymbol{0}$. Her betegner $\hat{\boldsymbol{p}}$ posisjonsestimatet, $\hat{\boldsymbol{v}}$ hastighetsestimatet, $\hat{\boldsymbol{b}}$ biasen til akselerometeret og $\boldsymbol{f}_{\text{imu}}^b$ er akselerometermålingen. Denne observeren er ulineær fordi man estimerer akselerometer-biasen i referanseramma til sensoren (body-ramma), slik at denne må roteres til jordramma. $\boldsymbol{K}_1$, $\boldsymbol{K}_2$ og $\boldsymbol{K}_3$ er justeringsfaktorer for observeren. Disse gir tilbakekobling fra posisjonsmålingen til alle de andre tilstandene, bare at for tilbakekobling til akselerometerbiasen så brukes rotering fra jordramma til sensorramma. Justeringskonstantene kan regnes ut ved å bruke et Extended Kalman Filter (EFK), ved spesifisering av målestøy matrisen $\boldsymbol{R}$ og prosessstøy-matrisen Q. I tillegg til bias-estimeringen, og at posisjonen måles til $\boldsymbol{0}$, fungerer observeren ganske enkelt ved at den dobbelt-integrerer den vertikale akselleresjonen.

2.3.2. Estimering av bølgehøyde i sanntid

I programvaren til bølgesensoren i denne oppgaven, brukes en observer som bare estimerer posisjon og hastighet vertikalt. Den vertikale posisjonen tilsvarer som sagt bølgehøyden. Denne observeren er en forenklet utgave av observeren i ligning 2.1. Det brukes også bare én bias-tilstand. Ligningene for observeren er som følger:

$$\begin{aligned} \dot{\hat{p}}_z &= \hat{v}_z - k_1\hat{p}_z \\ \dot{\hat{v}}_z &= (\boldsymbol{f}_{\text{imu}}^n)_z + g_z - \hat{b} - k_2\hat{p}_z \\ \dot{\hat{b}} &= -k_3\hat{p}_z \end{aligned}$$

På tilstandsrom-form blir dette:

$$\begin{bmatrix} \dot{\hat{p}}_z \\ \dot{\hat{v}}_z \\ \dot{\hat{b}} \end{bmatrix} = \begin{bmatrix} -k_1 & 1 & 0 \\ -k_2 & 0 & -1 \\ -k_3 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \hat{p}_z \\ \hat{v}_z \\ \hat{b}_z \end{bmatrix} + \begin{bmatrix} 0 \\ (\boldsymbol{f}_{\text{imu}}^n)_z + g_z \\ 0 \end{bmatrix} \tag{2.2}$$

Man setter målingen til $\boldsymbol{z} = p_z = 0$. Justeringsfaktorene k_1, k_2 og k_3 ble beregnet ved bruk av kalman-filter, ved å justere målestøyen r, og prosess-støyen q. Implementasjonen finnes i vedlegg A.4. Implementasjonen er på en generell form og kalman-forsterkningen og kovariansmatrisen blir beregnet i hvert tidssteg. Siden observeren som brukes er lineær, så kunne man brukt konstante justeringsfaktorer, men ved å beregne kalman-forsterkningen i hvert tidssteg, så gjør det at det blir forholdsvis enkelt å bytte ut denne observeren med observeren i ligning 2.1.

2.4. Beregning av bølgedata og bølgestatistikk

Kilde-koden for beregning av bølgedata og bølgestatistikk for bølgesensoren, finnes i vedlegg A.6, og som python-kode i vedlegg A.8.

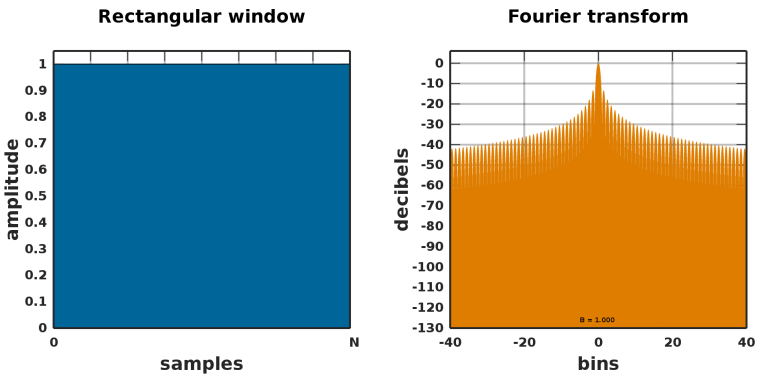
Bølgedata som beregnes i denne oppgaven er bølgehøyde i tidsdomenet, bølgespekteret, signifikant bølgehøyde og gjennomsnittlig bølgeperiode, h_{m0} og t_{m01} basert på bølgespekteret, og første- og andre fourierkoeffisienter for retningsspekteret. Bølgespekteret er mye brukt i analysen av bølger, og brukes til å regne ut flere bølgeparametre h_{m0} og t_{m01}. Retningsspekteret er likt som bølgespekteret, men viser også hvordan bølgene er fordelt over ulike bølgeretninger. To parametre som ofte beregnes fra retningsspekteret er bølgeretningen og bølgespredningen.

Tidsserien av bølgehøyden er relevant hvis man vil analysere hver enkelt bølge. Det kan for eksempel være for å studere ulineære egenskaper ved bølgene. Det er også vanlig å beregne statistiske bølgeparametre fra tidsserien ved bruk av nullkrysningsanalyse. I denne oppgaven beregnes også tidsserien for bølgehøyden i sanntid, og da kan det være en mulighet å basere seg på denne for den periodiske bølgeanalysen. En ulempe med dette er at sanntidsberegningen ikke har informasjon om fremtidige målinger. Hvis man derimot beregner bølgehøyden for hver periode, kan man bruke ikke-kausal filtrering, som potensielt gir mer nøyaktig estimater og som typisk gjør at man slipper faseforskyvning.

Bølgehøyden i tidsdomenet for hver periode beregnes på lignende måte som for sanntidsberegningen. Med orienteringen $\hat{\boldsymbol{q}}$ blir akselerasjonsmålingen i NED lik $\boldsymbol{f}^n_{\text{imu}} = \boldsymbol{R}^b_n(\hat{\boldsymbol{q}})^T \boldsymbol{a}^b_{\text{imu}}$. Den vertikale akselerasjonsmålingen er da z-komponenten: $(\boldsymbol{f}^n_{\text{imu}})_z$. Akselerometeret måler spesifikk kraft slik at, den faktiske akselerasjonen vertikalt blir $a^n_z = (\boldsymbol{f}^n_{\text{imu}})_z + g$, fordi man må trekke fra den spesifikke normalkraften som følge av gravitasjonen. Man finner bøyens hiv-posisjon fra den vertikale akselerasjonen a^n_z ved å dobbeltintegrere og høypassfiltrere. Høypassfiltrering må til fordi man over tid vil integrere opp målestøy, som gjør at p_z vil drifte. Disse operasjonene kan gjøres i frekvensdomenet. En annen fordel med å beregne hiv i frekvensdomenet er at man uansett beregner fourier-transformen til bruk i spektralanalysen. Da er det ganske fort gjort å beregne hiv siden integrering og filtrering er enklere i frekvensdomenet. Dobbeltintegrering tilsvarer multiplikasjon med $1/(2\pi i f)^2$ der f er frekvensen[33]. Etter at man har dobbeltintegrert og høypassfiltrert kan man bruke invers fourier-transform for å finne tidsserien til hiv.

Når man beregner $a^n_z = (\boldsymbol{f}^n_{\text{imu}})_z + g$ er det bedre om man heller trekker fra snittet slik at gjennomsnitt til a^n_z over intervallet blir 0. Da fjerner man eventuell bias og lignende som føre til støy på grunn av spektral lekkasje. Man kan forstå hvorfor ved å se på frekvensspekteret til en rektangulær vindu-funksjon. Vindu-funksjonen illustrerer snittet som vi vil fjerne. se figur 2.1.

Når bølgespekteret er beregnet, finner man signifikant bølgehøyde ved formelen $h_{m0} = 4.0\sqrt{m_0}$, og gjennomsnittlig bølgeperiode ved formelen $t_{m01} = \frac{m_0}{m_1}$, der $m_k = \int f^k S(f) df$, $k \in \{0, 1\}$ gir første- og andremoment av bølgespekteret (se [7]). h_{m0} og t_{m01} er to statistiske bølgeparametre som blir mye brukt. Tradisjonelt ble signifikant bølgehøyde defineret



Figur 2.1.: Fourier-transform av rektangulært vindu

som gjennomsnittlig bølge for den største tredelen av bølgene. Typisk vil dette tilsvare h_{m0}, som gir en enklere måte å beregne denne verdien på.

2.4.1. Valg av analyseperiode og frekvensoppløsning

Perioden for bølgeanalysen bør velges lang nok til at man får lite spektrallekasje, og samtidig kort nok til at sjøtilstanden endrer seg lite, iløpet av perioden. Ifølge Tucker (2001) [29] så er cirka 17 minutter standard, men det er gjerne ønskelig at perioden er lengre enn dette. I denne oppgaven ble 17 minutter (2048 samples på 2 Hz) brukt for bølgesensoren og ved testing på rådata, men det blir også gjort tester for minneforbruk ved 34 minutter. Med lengde på 17 minutter er det ifølge Tucker (2001) ikke nødvendig å bruke vindus-funksjon før beregning av Fourier-transform, men det er ikke uvanlig å bruke det likevel.

Frekvensoppløsningen bør velges slik at bølgespekteret blir detaljert nok, uten at man får for mye samplingsstøy. Ifølge Tucker (2001) [29] er en oppløsning på 0.005 vanlig å bruke. I denne oppgaven tar man Fourier-transformen på tidsserier av 2048 samples på 2 Hz, som gir en oppløsning på $2/2048 = 0.001$Hz. Denne oppløsningen er altså 5 ganger høyere enn anbefalt, slik at man får betydelig med støy på bølgespekteret. Dette har ingenting å si for h_{m0} og t_{m01}, siden disse beregnes av momentene til spekteret. Men det har noe å si for eksempel ved beregning av fourier-koeffisientene for retningsspekteret. For å få riktig oppløsning på frekvensspektrene, bør kan man glatte disse, eller man kan beregne frekvensspekteret for kortere perioder, og så ta gjennomsnitt over periodene.

2.4.2. Diskrete effekter

Sensor-målingene fra IMUen er naturligvis diskrete verdier, med en konstant samplingstid. Så når man regner om til frekvensdomenet brukes diskret fourier-transform. Samplingstid for IMUen bør være høy nok til man får med de høyeste frekvensene for det man skal måle. Rådataene som blir testet på i denne oppgaven har samplingsrate på

6 Hz, mens i programvaren til bølgesensoren brukes 20 Hz for ekstra god margin. Når man skal beregne bølgespekteret virker det lite hensiktsmessig å bruke så høy frekvens, fordi de fleste bølgene er vesentlig langsommere, og fordi en typisk bøye er såpass tung at den demper de hurtigste bølgene. Derfor bør a_z^n nedsamples før beregning av Fourier-transformen, slik at man unngår lagring og beregning av overflødige mengder med data. Da er det greit å huske på Nyquist–Shannon samplingsteorem som sier at man minst må ha dobbelt så høy samplingsfrekvens som den hurtigste bølgefrekvensen man ønsker å måle.Når man nedsampler må man gjøre dette på en måte som unngår aliasing av signalet. I programvaren til bølgesensoren blir dataene nedsamplet fra 20 Hz til 2 Hz før man beregner Fourier-transform. Dette blir gjort ved å partisjonere signalet i deler med 10 etterfølgende samples og ta snittet av hver del. Dette er en enkel måte å nedsample på, men til senere bør det vurderes om det finnes bedre måter å gjøre dette på. Det ble ikke gjort noe nedsampling ved testingen på 6 Hz rådata.

3. Implementering

3.1. Viktige egenskaper

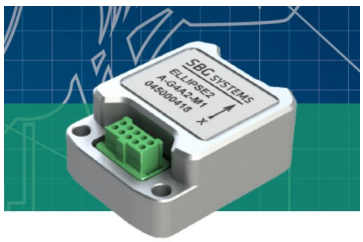
Følgende egenskaper ble vektlagt ved valg av maskinvare og utvikling av programvaren:

- Bølgesensoren skal være pålitelig, noe som er viktig fordi den skal kunne fungere på en målebøye langt til havs, som er lite tilgjengelig for tilsyn og kostbar å reparere.
- Det er naturligvis viktig at programvaren er vedlikeholdbar og lett kan utvides med ny funksjonalitet. Spesielt fordi programvaren er en prototyp, som forhåpentligvis har potensiale til å være nyttig for flere forskjellige applikasjoner.
- Bølgesensoren skal ha lavt strømforbruk, fordi en målebøye typisk står på havet over lang tid, der det er kostbart med tilsyn.
- Responstid vil normalt sett ikke være kritisk når bølgesensoren brukes på en vanlig målebøye, men god responstid er likevel positivt, fordi dette kan åpne opp for flere potensielle bruksområder. Samtidig gir god responstid bedre mulighet for å visualisering av estimeringen ved oppkobling mot PC, slik det ble gjort under testing.

3.2. Valg av maskinvare

3.2.1. IMU

IMUen som vil brukes til bølgesensoren er en Ellipse 2 Micro IMU fra SBG, som er lånt fra Fugro. Denne sensoren koster over 10 000 kroner, og utgjør derfor det aller meste av hardware-kostnader for bølgesensoren i denne oppgaven. Det som først og fremst gjør at denne sensoren er relativt dyr, er at den er ferdig kalibrert. IMUen kommer med manual og drivere i c-kode til å kommunisere med IMUen. Ellipse 2 Micro har to serie-porter, i tillegg til en CAN-port som bare kan sende data. Serie-portene kan bruke enten RS422 eller RS232 som fysisk protokoll. I denne oppgaven vil vi kommunisere over RS232 som av disse tre er den enkleste og vanligste protokollen å bruke. Strømforbruk er oppgitt til 400mW. For mer informasjon om denne IMUen vises det til [28].



Figur 3.1.: Ellipse 2 Micro

	Accelerometers	Gyroscopes	Magnetometers
Range	± 16 g	± 450 °/s	± 50 Gauss
Gain stability	1000 ppm	500 ppm	< 0.5 %
Non-linearity	1500 ppm	50 ppm	< 0.1 % FS
Bias stability	± 5 mg	± 0.2 °/s	± 1 mGauss
Random walk/ Noise density	57 µg/√Hz	0.15 °/√hr	3 mGauss
Bias in-run instability*	14 µg	7 °/h	1.5 mGauss
VRE	50 µg/g² RMS	1 °/h/g² RMS	-
Alignment error	< 0.05 °	< 0.05 °	< 0.1 °
Bandwidth	390 Hz	133 Hz	22 Hz

** Allan Variance, @ 25 °C*

Figur 3.2.: Ellipse 2 Micro egenskaper

3.2.2. Mikrokontroller

Mikrokontrolleren (MCU) som ble valgt er fra serien STM32L4+. Denne er en del av 32-bits mikrokontroller-familien STM32 fra STMicroelectronics(ST), som er basert på prosessorkjernen ARM Cortex-M. Mer spesifikt bruker STM32L4+ kjernen Cortex-M4F. En viktig grunn til å velge en slik mikrokontroller fremfor en svakere type, er at denne applikasjonen krever relativt mye RAM. Under bølgeanalysen bruker man flere lange dataserier: hvis man for eksempel har 2048 sekunder intervall på 2 Hz og hver verdi lagres i 4-bytes flyttall, blir det $2048 \cdot 2 \cdot 4 = 16$kB for hver dataserie. Da blir det fort flere hundre kilobytes med data man skal regne på. STM32L4+ har 640 kB med RAM. En annen grunn er at 32-bits MCUer har fått så lavt strømforbruk og er blitt så billige at det er lite å tjene på å bruke 8 eller 16-bit.

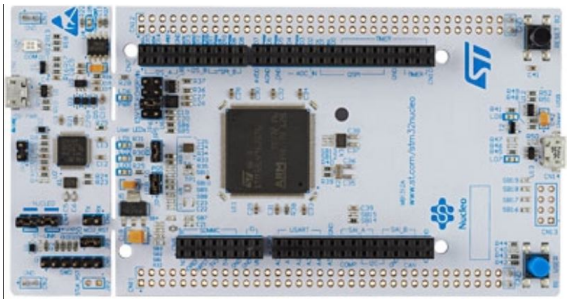
Det er også en fordel at Cortex-M4 har DSP-instruksjoner. Det betyr at digital signal behandling går hurtigere, og man har et eget funksjonsbibliotek (API), som gjør at for eksempel FFT (Fast Fourier Transform), matrisemultiplikasjon og annet gjøres enkelt, med et funksjonskall. Siden det er mye bruk av digital signal behandling både ved beregning av posisjon og orientering, og i bølgeanalysen (f.eks. fourier-transform), så gir dette raskere beregninger og det gjør programmeringsjobben enklere. Valg av en 32-bits

mikrokontroller slik som Cortex-M4F betyr også at man har lite begrensing med tanke på prosessorytelse, i tilfelle bølgesensoren skal utvides med flere, og mer avanserte og tyngre beregninger.

Det finnes mange andre 32-bits MCUer man kan velge. PIC32 er en alternativ arkitektur og mikrokontroller-familie fra Microchip. Cortex-M arkitekturen fra ARM er den mest populære, og MCUer fra mange forskjellige brikke-produsenter er basert på denne arkitekturen. ST er blant de store brikkeprodusentene, og NXP og Texas Instruments er andre alternativer. Ved valg av en stor brikkeprodusent som ST har man tilgang til et gratis og kraftfullt utviklingsverktøy og eksempelkode man kan bruke. ST har dessuten et bra utvalg av evalueringskort man kan kjøpe.

3.2.3. Evalueringskort

Evalueringskortet som blir brukt i denne oppgaven er et NUCLEO-L4R5ZI fra ST. MCU-en heter STM32L4R5ZI. Kortet kan kjøpes på nett til cirka 300 kroner. Dette er et STM32 Nucleo-kort, som er den enkleste typen evalueringskort fra ST. I likhet med Arduino kan Nucleo-kortene bruke utvidelseskort. Alternativt har man mulighet til å kjøpe et Discovery-kort hvis man blant annet vil ha mer RAM. Et STM32L4R9I-DISCO kan bestilles for cirka 900 kr på nett og har blant annet 16-Mbit ekstern RAM og 512-Mbit eksternt flash-minne, og en liten LED-skjerm. Et annet alternativ er NUCLEO-H7A3ZI-Q til cirka 300 kroner. Denne bruker en kraftigere MCU fra ST basert på Cortex-M7 med 1.4MB intern RAM.

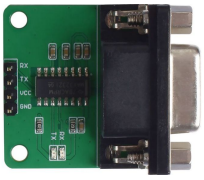


Figur 3.3.: NUCLEO-L4R5ZI

3.2.4. Annet

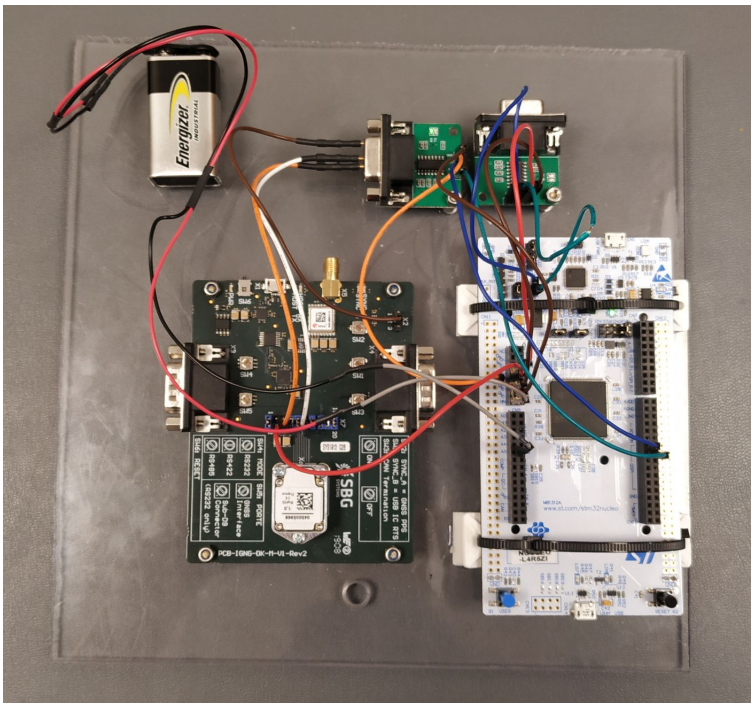
Kommunikasjon mellom MCU og IMU skjer med serielt grensesnitt over RS232. RS232 er den fysiske protokollen og definerer blant annet spenningsnivåene. Spenningsnivåene til RS232 er for store til å kommunisere direkte med MCUen og signalet må derfor konverteres til TTL-nivå som er spenningsnivået til MCUen. I denne oppgaven bruker vi en omformer som kan kjøpes fra Elfa Distrilec til cirka 25 kroner. Seriellkommunikasjon over RS232 er en vanlig måte og kommunisere med eksterne sensorer, og dette

grensesnittet blir i denne oppgaven også brukt mellom datalogger og bølgesensor. Derfor trenger vi 2 slike omformere.



Figur 3.4.: RS232-TTL omformer

Figur 3.5 viser bølgesensoren når alle delene er sammenkoblet. Vi ser at SBG-sensoren er montert på det mørkegrønne utviklingskortet fra SBG. Ingen av de elektroniske komponentene på dette kortet brukes under testing og utvikling av bølgesensoren, men kortet er praktisk for oppkobling og montering, samtidig som det er nyttig hvis man vil koble SBG-sensoren direkte til en PC, for enkel konfigurasjon eller testing.



Figur 3.5.: Bølgesensoren ferdig sammenkoblet

3.3. Konfigurering av mikrokontroller

For MCU-familien STM32 tilbyr ST et gratis integrert utviklingverktøy (IDE) kalt STM32CubeIDE. Dette verktøyet støtter programmeringsspråkene C og C++, og er basert på IDEen Eclipse og GNU-verktøysettet, som begge er fri og åpen kildekode.

GNU-verktøysettet består blant annet av kompilator og debugger. STM32CubeIDE inkluderer en kodegenerator for oppsett og initialisering av mikrokontrolleren som gjør at man enklere kommer i gang. ST tilbyr også en kodepakke kalt STM32Cube som kan installeres via IDEen og som er nødvendig for at kodegeneratoren skal fungere. STM32Cube kommer i flere utgaver avhengig av hvilken MCU man bruker og for familien STM32L4 og L4+ installerer man STM32CubeL4. Denne kodepakken inneholder drivere, middelvare (slik som Free-RTOS) og eksempelkode. IDEen tilbyr også en del kode utover det man finner i STM32CubeL4, som kan installeres etter behov.

Når man oppretter et nytt prosjekt velger man MCU eller evalueringskort. Man kan velge mellom C eller C++. I denne oppgaven er C valgt som språk. I prosjektet som opprettes ligger det allerede nå grunnleggende kode som kan kompileres og lastes over på mikrokontrolleren. Foreløpig går `main()`-funksjonen inn i en tom loop, så man må legge til mer kode for at noe skal skje.

I kodegeneratoren ønsker vi å konfigurere oscillatorer, timere, programmerer/debugger, serie-grensesnitt og FreeRTOS. I figur 3.6 ser vi alle 144 pinnene på MCUen. De grå pinnene er ikke konfigurert, mens de 9 grønne med navn på er de som vi har valgt å konfigurere.

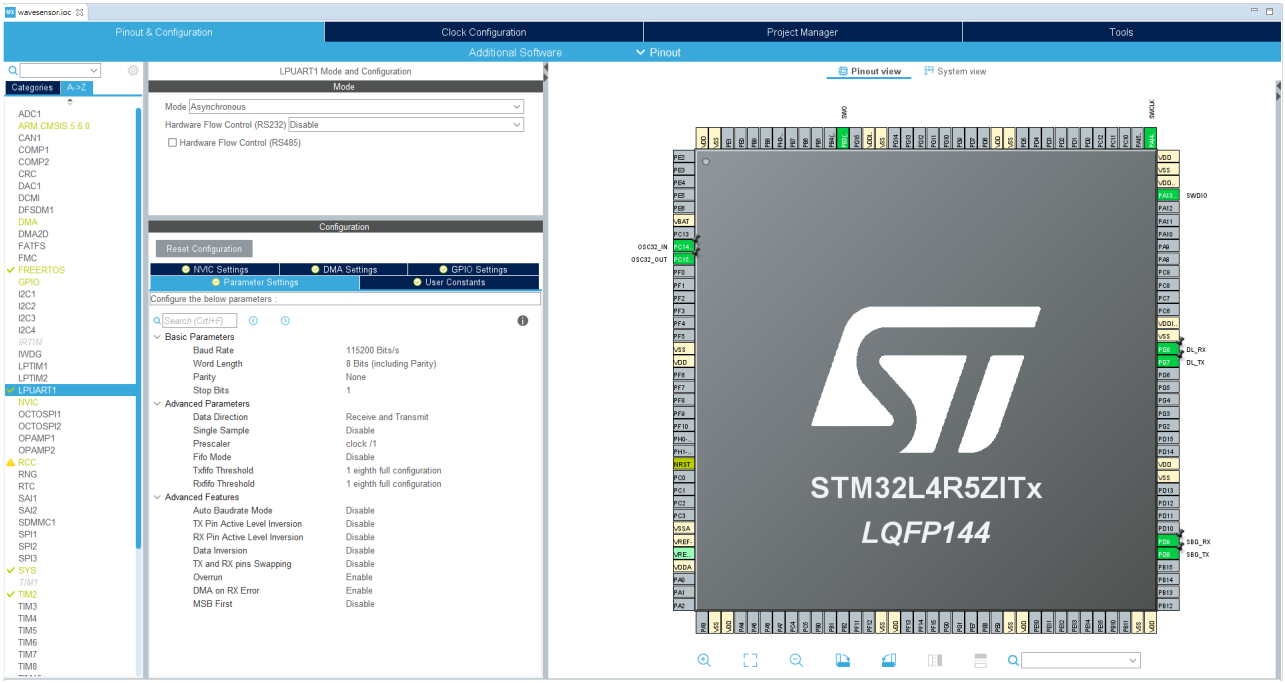
3.3.1. Programmerer

For å kunne laste over kode, og debugge MCUen trenger man en programmerer/debugger. Evalueringskortet Nucleo 144 kommer med en innebygd programmerer/debugger som heter ST-LINK og som bruker ARM sin SWD(Serial Wire Debug)-protokoll. Med denne er det bare å koble en USB-ledning mellom PCen og konnektor CN1 på Nucleokortet og man er klar til å laste over koden fra IDEen. ST-LINKen er tilkoblet de tre pinnene som heter SWIO, SWCLK og SWO. For å konfigurere støtte for ST-LINKen går man til *SYS* og setter *DEBUG* til *Trace Asynchronous Sw.*

3.3.2. Oscillatorer

Som standard er MCUen satt til å bruke MSI (Multispeed internal RC oscillator) som kilde til systemklokka. MSI er justerbar og som standard satt 4000 kHz. Oscillatorer slik som MSI som finnes internt i MCUen er ikke spesielt nøyaktige. Men i følge databladet [26] kan MSI auto-trimmes av LSE (32.768 kHz low-speed external crystal) til å få en nøyaktighet på minst $\pm 0.25\%$. En slik oscillator er tilgjengelig på Nucleo-kortet. Dette kan være nyttig for å få en mer nøyaktig tidsstempel på de data som bølgesensoren måler og genererer. For å konfigurere dette går vil til *RCC* og setter *Low Speed Clock til Crystal/Ceramic Resonator* og *RCC Parameters -> MSI Auto Calibration* til *Enabled.* De to grønne pinnene OSC32_IN og OSC32_OUT går til den eksterne krystallen på Nucleo-kortet.

For å spare litt strømforbruk setter vi *Power parameters -> Power Regulator Voltage Scale* til *2* (se side 28 i databladet [26]).



Figur 3.6.: Kodegeneratoren i STM32CubeIDE

3.3.3. UART

En hardware-enhet som implementerer seriell-kommunikasjon av typen RS-232 eller lignende, kalles UART, og på denne MCUen finnes det flere UARTer. Vi trenger et seriegrensesnitt mot datalogger og et mot SBG-sensoren. ST-Link brikken i Nucleo-kortet har en innebygd virtuell seriell port som ifølge manualen til Nucleo-kortet [25] er koblet til pinne PG7 og PG8 på MCUen . Ved å bruke disse to pinnene for kommunikasjon til datalogger, har vi mulighet til å kommunisere gjennom USB-kabelen som brukes under debugging. Hvis man ikke debugger, kobler man seg de to pinnene TX og RX på konnektor CN6 på Nucleo-kortet.

Hver konfigurerbar pinne på MCUen har flere alternative funksjoner som kan brukes. For eksempel kan PG7 og PG8 konfigureres til å bruke blant annet I2C og UART. For å konfigurere kommunikasjonen mot datalogger, må man trykke på pinnene *PG7* og *PG8* og velge *LPUART_TX* og *LPUART_RX*. Deretter går man til *LPUART1* og setter *Mode* til *Asynchronous*. For grensesnittet til SBG-sensoren bruker vi pinnene D1 og D0 på CN10 på Nucleo-kortet. Disse er koblet til PD8 og PD9. Vi velger derfor disse to pinnene til å bruke USART3 på samme måte som før. For UART3 er standard innstillingen i kodegeneratoren for baudrate og ordlengde satt til 115200 bit/s og 8 bits. Disse verdiene er vanlig å bruke, og blir også brukt i denne oppgaven. For LPUART1 er standardinnstillingen satt annerledes og man må derfor endre disse.

DMA

Prosessoren kjører på en mye høyere frekvens enn UARTene og for at prosessoren skal slippe å vente når UARTen kommuniserer er det lurt å bruke DMA(Direct Memory Access). DMA er også lurt å bruke fordi da slipper prosessoren og bli avbrutt når UARTen mottar en melding. Derfor er i denne oppgaven DMA konfigurert for TX(Sending) og RX(Mottak) til LPUART1 (datalogger), og for RX til USART3 (SBG-sensor). For å konfigurere DMA for USART3 RX, går vi til *DMA*, trykker *ADD* og velger *USART3_RX*. Vi setter *Mode* til *Circular*. Sirkulær betyr at vi bruker et sirkulært buffer. På tilsvarende måte konfigureres LPUART1 RX og LPUART1 TX, men for LPUART1 TX lar vi *Mode* være lik *Normal* som er standard innstilling.

3.3.4. FreeRTOS

Software til bølgesensoren kan deles inn i 4 forskjellige oppgaver:

1. Motta IMU-data
2. Beregne sanntids hiv og orientering, og sende sanntidsdata til datalogger
3. Beregne bølgestatistikk for hvert tidsintervall.
4. Motta og behandle meldinger fra datalogger

Det er ønskelig at sanntidsdata har liten forsinkelse slik at når IMU-data er mottatt, bruker bølgesensoren kort tid på å beregne og overføre hiv og orientering. Derfor bør MCUen prioriterer å gjøre denne oppgaven før de to andre. Da er det hensiktsmessig å bruke en operativsystemkjerne med støtte for multitasking slik som FreeRTOS. En annen fordel med å bruke en OS-kjerne er at koden kan bli mer strukturert og gi bedre muligheter på sikt for å kunne utvide software med mer funksjonalitet. FreeRTOS konfigureres med kodegeneratoren.

For å aktivere FreeRTOS går man til *FREERTOS* og setter *Interface* til *CMSIS_V2*. Under fanen *Config parameters* settes *ENABLE_FPU* til *Enabled*.

Som standard vil det genereres et interrupt hvert millisekund som gjøre at TICK-timeren inkrementeres og OS-kjernen vil sjekke om det er noen tråder i vente-modus med høyere prioritet. Hvis det er det, vil kjernen gjøre et tråd-bytte fra den tidligere tråden til den ventende tråden med høyest prioritet. Parameteren *TICK_RATE_HZ* bestemmer hvor ofte dette interruptet genereres, noe som ikke bør skje oftere enn nødvendig. I denne oppgaven setter vi *TICK_RATE_HZ* til *100*, slik at interruptet genereres hvert hundredels-sekund.

Vi setter *USE_IDLE_HOOK* til *Enabled*. Med denne genereres en funksjon som kalles når IDLE-tråden kjører, noe som skjer når ingen andre tråder venter på å kjøre. Det er lurt å legge en sleep-instruks i IDLE-hook-funksjonen for å spare strøm.

Vi setter *TIMER_TASK_PRIORITY* til *40* slik at FreeRTOS-timerene vi definerer får høyere prioritet enn trådene vi skal definere.

Under fanen *Tasks and Queues* definerer vi de fire trådene som henholdsvis: *sbgTask*, *ahrsTask*, *waveTask* og *receiveMsgTask*. For hver tråd definerer man *Stack Size* og setter *Allocation* til *Static*. Vi setter også prioritet for sbg-tråden og ahrs-tråden til *osPriorityAboveNormal* slik at disse to trådene får høyere prioritet enn wave-tråden og receiveMsg-tråden.

Under fanen *Timers and Semaphores* defineres to timere: *sbgTimer* og *receiveMsgTimer*. Sbg-timeren er en periodisk FreeRTOS-timer som for hvert OS-tick (hundredelssekund) kaller en callback-funksjon som sender melding til sbg-tråden om å sjekke om data fra IMU-sensor er mottatt. ReceiveMsg-timeren er helt lik bortsett fra at callback-funksjonen i stedet sender melding til receiveMsg-tråden om å sjekke om det er mottatt en melding fra datalogger.

Vi definerer to semaforer: *transferDataBinarySem* og *waveDataBinarySem*. TransferData-semaforen er en binær semafor som kontrollerer at bare én tråd om gangen kan sende data til datalogger. WaveData-semaforen kontrollerer tilgang til bølgedataene, slik at bare én tråd om gangen kan lese eller skrive til disse dataene. For begge timere og semaforer er *Allocation* satt til *Static*.

3.3.5. Drivere

Det er nyttig å bruke ferdige drivere når man programmerer MCUen. For kodepakken STM32CubeL4 kommer med to typer drivere: STM32Cube Hardware Abstraction Layer (HAL) og low-layer APIs (LL) [27]. HAL er mer generell enn LL, mens LL består av mindre kode og ligger nærmere hardware. Det ble prøvd å bruke HAL i begynnelsen, men det ville ikke fungere med DMA for både RX og TX på UART mot dalalogger. Når HAL-driveren ikke fungerer er det vanskelig å feilsøke fordi koden er ganske kompleks. LL ble prøvd i stedet og da fungerte alt bra. En annen fordel med LL er at man får mer innsikt i hvordan MCUen fungerer, fordi man programmerer hardware mer direkte. Man kan bruke LL og HAL samtidig, men i denne oppgaven brukes bare LL. For å bruke LL i kodegeneratoren, går man til *Project Manager -> Advanced Settings* og bytter fra *HAL* til *LL* for hver enhet.

3.3.6. Timere

Når FreeRTOS settes opp med kodegeneratoren vil FreeRTOS bli satt til å bruke SysTick-timeren som er maskinvare-timeren i Cortex-M kjernen. Det er denne timeren som vil generere et interrupt som inkrementerer Tick-timeren og gir styring til OS-kjernen. I kodegeneratoren under *SYS* er *Timebase Source* satt til *SysTick* som standard. Vi endrer denne til *TIM1* fordi SysTick nå blir brukt av FreeRTOS. Timebase Source genererer et interrupt hvert millisekund og brukes av HAL-driveren. Siden vi ikke bruker HAL-driveren, vil vi deaktivere denne timeren. Det er dessverre ikke mulig å gjøre i kodegeneratoren, så i stedet kan man for eksempel skrive `return HAL_OK;` øverst i `HAL_InitTick()`-funksjonen slik at denne timeren ikke initialiseres.

Siden tick-timeren til FreeRTOS har litt lav oppløsning kan det være greit å ha en ekstra timer som teller millisekund. Dette er dessuten nødvendig for å kunne måle kjøretiden til hver enkelt tråd når man tester software. For å legge til en ekstra timer går vi til *TIM2* og setter *Clock Source* til *Internal Clock*. Vi setter *Prescaler* til *4000* siden klokka til TIM2 er satt til å kjøre på 4MHz og 4MHz/1000Hz = 4000. Vi setter *Counter Period* til *0xffffffff* slik at timeren teller til 2^{32} før den begynner å telle på nytt.

3.3.7. Generere kode

Når MCUen er ferdig konfigurert i kodegeneratoren, genererer man kode ved å velge Project -> Generate Code i IDEen. Koden skal nå være klar til å kompilere.

3.4. Eksempel: Initialisering og bruk av UART transmitter

3.4.1. Initialisering

LPUART1 er UARTen som brukes til å kommunisere med datalogger. Mesteparten av koden for initialisering av Tx (sender) til LPUART1 er generert av kodegeneratoren. Denne koden med forklaring kan leses i vedlegg A.1. Det er noe mer kildekode enn dette, men den har ingen effekt så lenge koden kjøres etter en resett. Etter at denne koden har kjørt er det noen ting man må gjøre selv før initialiseringen er ferdig. For LPUART1 må man aktivere DMA modus for sending. Det gjøres ved å sette bit 7 (DMA enable transmitter) i Control register 3 (LPUART1->CR3):

```
LL_LPUART_EnableDMAReq_TX(LPUART1);
```

For DMA må vi registrere de to adressene som DMA skal kopiere data mellom. Channel 2 memory address register (DMA1->CMAR2) settes lik adressen til `uint8_t Lpuart1_txBuffer[]` som er bufferet DMA kopierer fra. Channel 2 peripheral address register (DMA1->CPAR2) settes lik adressen til Transmit data register for LPUART1 (LPUART1->TDR) som er registeret DMA kopierer data til:

```
LL_DMA_ConfigAddresses(DMA1, LL_DMA_CHANNEL_2, (uint32_t)
   Lpuart1_txBuffer, LL_LPUART_DMA_GetRegAddr(LPUART1,
   LL_LPUART_DMA_REG_DATA_TRANSMIT),
   LL_DMA_GetDataTransferDirection(DMA1, LL_DMA_CHANNEL_2));
```

Vi må også konfigurere DMA1 på kanal 2 til å sende et interrupt signal når hele overføringen er fullført. Det gjøres ved å sette bit 1 (Transfer complete interrupt enable) i Channel 2 configuration register (DMA1->CCR2):

```
LL_DMA_EnableIT_TC(DMA1, LL_DMA_CHANNEL_2);
```

3.4.2. Sende sanntidsdata til datalogger

Henter semafor og kopierer data

For å unngå at flere tråder prøver å sende til datalogger samtidig bruker vi semaforen `transferDataBinarySemHandle` for å kontrollere tilgangen. Denne semaforen må derfor først anskaffes:

```
osSemaphoreAcquire(transferDataBinarySemHandle,
   osWaitForever);
__asm volatile( "" ::: "memory" );
```

Den siste linjen er en kompilator minne-barriere som gjør at kildekoden som kommer nedenfor denne linjen, vil i assemblerkoden komme etter kildekoden ovenfor linjen. Dermed er vi sikker på at koden ikke vil optimaliseres på en slik måte at noe av den etterfølgende koden vil flyttes til før semaforen er anskaffet. For mer informasjon se seksjon 3.5.4.

Nå kan vi trygt kopiere dataene som skal sendes til den globale variabelen `txMsg` og etterpå kjøre funksjonen `LPUART1_SendMsg()`:

```
txMsg.Data[0] = CMD_LOG;
txMsg.Len = 1;
uint32_t tick = GetTick();
memcpy(txMsg.Data + txMsg.Len, (void *)&tick, 4);
txMsg.Len += 4;
memcpy(txMsg.Data + txMsg.Len, (void *)ahrs_q, 16);
txMsg.Len += 16;
memcpy(txMsg.Data + txMsg.Len, (void *)heaveRT, 4);
txMsg.Len += 4;
memcpy(txMsg.Data + txMsg.Len, (void *)sbgData.acc, 12);
txMsg.Len += 12;
memcpy(txMsg.Data + txMsg.Len, (void *)sbgData.gyro, 12);
txMsg.Len += 12;
memcpy(txMsg.Data + txMsg.Len, (void *)sbgData.mag, 12);
txMsg.Len += 12;

LPUART1_SendMsg();
```

Implementasjonen som brukes her er enkel og rett fram, men det er potensielt risikabelt at man alltid må huske på å hente transferData-semaforen før man modifiserer variabelen `txMsg` og kaller funksjonen `LPUART1_SendMsg()`. Hvis man glemmer dette, kan feilsøkingen fort bli tidkrevende. Til senere bør det nok derfor implementeres en mekanisme som forhindrer at man glemmer dette.

Legger til sjekksum og byte-stuffing

Funksjonen `LPUART1_SendMsg()`:

```
void LPUART1_SendMsg()
{
  uint16_t count_out;
  CHK_ComputeAndAdd(&txMsg);
  count_out = ByteStuffCopy(Lpuart1_txBuffer, &txMsg);
  LPUART1_Transmit(count_out);
}
```

Vi legger først til en byte med sjekksum. Overføringsprotokollen bruker byte-stuffing og dataene som skal sendes til UARTen legges i bufferet `uint8_t Lpuart1_txBuffer[]`. Siste linje kaller funksjonen `LPUART1_Transmit()`:

Aktiverer DMA

```
void LPUART1_Transmit(int count_out)
{
```

Vi må fortelle DMA hvor mange bytes som skal overføres. Det gjøres ved å sette Channel 2 number of data register (DMA1->CNDTR2) til antall bytes:

```
  LL_DMA_SetDataLength(DMA1, LL_DMA_CHANNEL_2, count_out);
```

I henhold til referansemanualen [24] side 1825 må bit 6 (Transmission complete) i Interrupt status register til LPUART1 (LPUART1->ISR) settes til 0. Det gjøres ved å sette bit 6 (Transmission complete clear flag) i Interrupt flag clear register til

```
  LPUART1 (LPUART1->ICR):
  LL_LPUART_ClearFlag_TC(LPUART1);
```

Til slutt startes overføringen ved å aktivere DMA1 kanal 2. Det gjøres ved å sette bit 0 (Channel enable) i Channel 2 configuration register (DMA1->CCR2):

```
  LL_DMA_EnableChannel(DMA1, LL_DMA_CHANNEL_2);
}
```

Avslutter overføring i interrupt-handleren

Når DMA har overført alle bytes vil den generere et interrupt på IRQ 11. Dette er interruptet til DMA1 kanal 2. Det gjør at funksjonen `DMA1_Channel2_IRQHandler()` blir kalt:

```
void DMA1_Channel2_IRQHandler(void)
{
```

Vi sjekker først at interruptet er forårsaket av at overføringen er ferdig. Det gjøres ved å sjekke bit 5 (Channel 2 transfer complete flag) i Interrupt status register til DMA1 (DMA1->ISR):

```
  if (LL_DMA_IsActiveFlag_TC2(DMA1))
  {
```

Deretter må dette flagget settes til 0. Det gjøres ved å sette bit 5 (Channel 2 transfer complete clear) i Interrupt flag clear register til DMA1 (DMA1->IFCR):

```
    LL_DMA_ClearFlag_TC2(DMA1);
```

Så deaktiverer vi DMA1 kanal 2. Det gjøres ved å nullstille bit 0 (Channel enable) i Channel 2 configuration register (DMA1->CCR2):

```
    LL_DMA_DisableChannel(DMA1, LL_DMA_CHANNEL_2);
```

Tilslutt tilbakegir vi den binære semaforen, slik at andre tråder får mulighet til å sende data. Men først legger vi inn en kompilator minne-barriere slik at kompilatoren, på grunn av optimalisering, ikke flytter tidligere instruksjoner til etter at semaforen er tilbakegitt:

```
    __asm volatile( "" ::: "memory" );
    osSemaphoreRelease(transferDataBinarySemHandle);
  }
}
```

3.5. Programvaren

De viktigste delene av kildekoden er tilgjengelig i vedlegg A.

3.5.1. Tredjeparts-drivere og -kildekode

Programvaren bruker følgende tredjeparts-drivere og -kildekode:

CMSIS DSP software-biblioteket fra ARM [1] til programmering av DSP som i hovedsak blir brukt til bølgeberegninger og implementasjon av kalman-filteret til hiv-beregning.

CMSIS RTOS2 som er et generelt RTOS-grensesnitt fra ARM [2], som i denne oppgaven brukes brukes mot FreeRTOS.

Driveren sbgECom versjon 1.10 fra SBG Systems, som blir brukt kommunikasjon med SBG-sensoren.

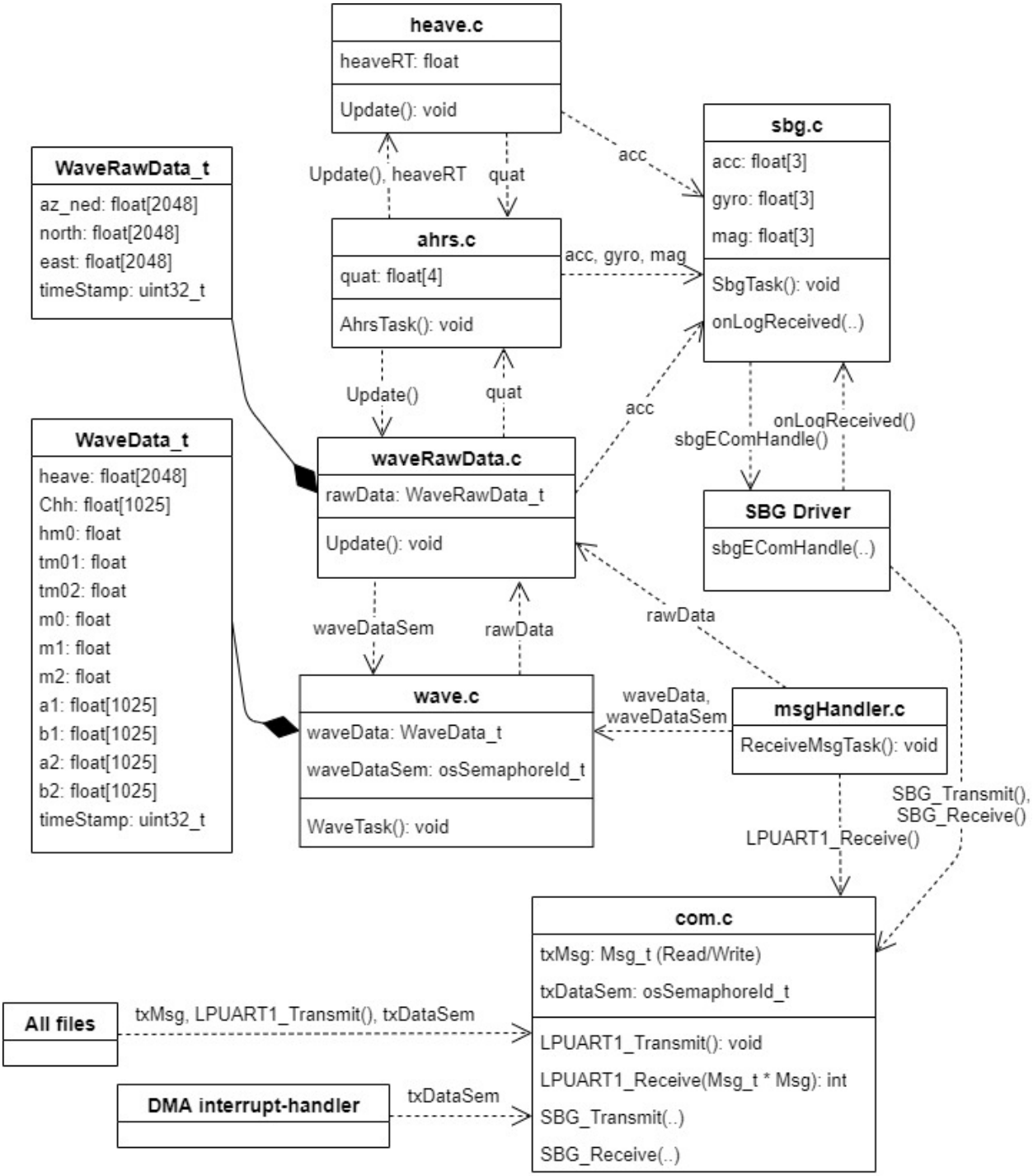
LL(Low-Layer)-driveren fra STMicroelectronics som brukes til programmering av periferi-enhetene på MCUen, slik som UARTer, DMA etc.

En byte-stuffing protokoll (se A.2.1 i vedlegg) fra software-pakken X-CUBE-MEMS1 [23] fra ST.

Til python-driveren på datalogger brukes kode fra Github [5] til visualisering av orientering i sanntid.

LL, CMSIS DSP og CMSIS RTOS2 med FreeRTOS følger med softwarepakken STM32Cube fra ST [22].

3.5.2. Design



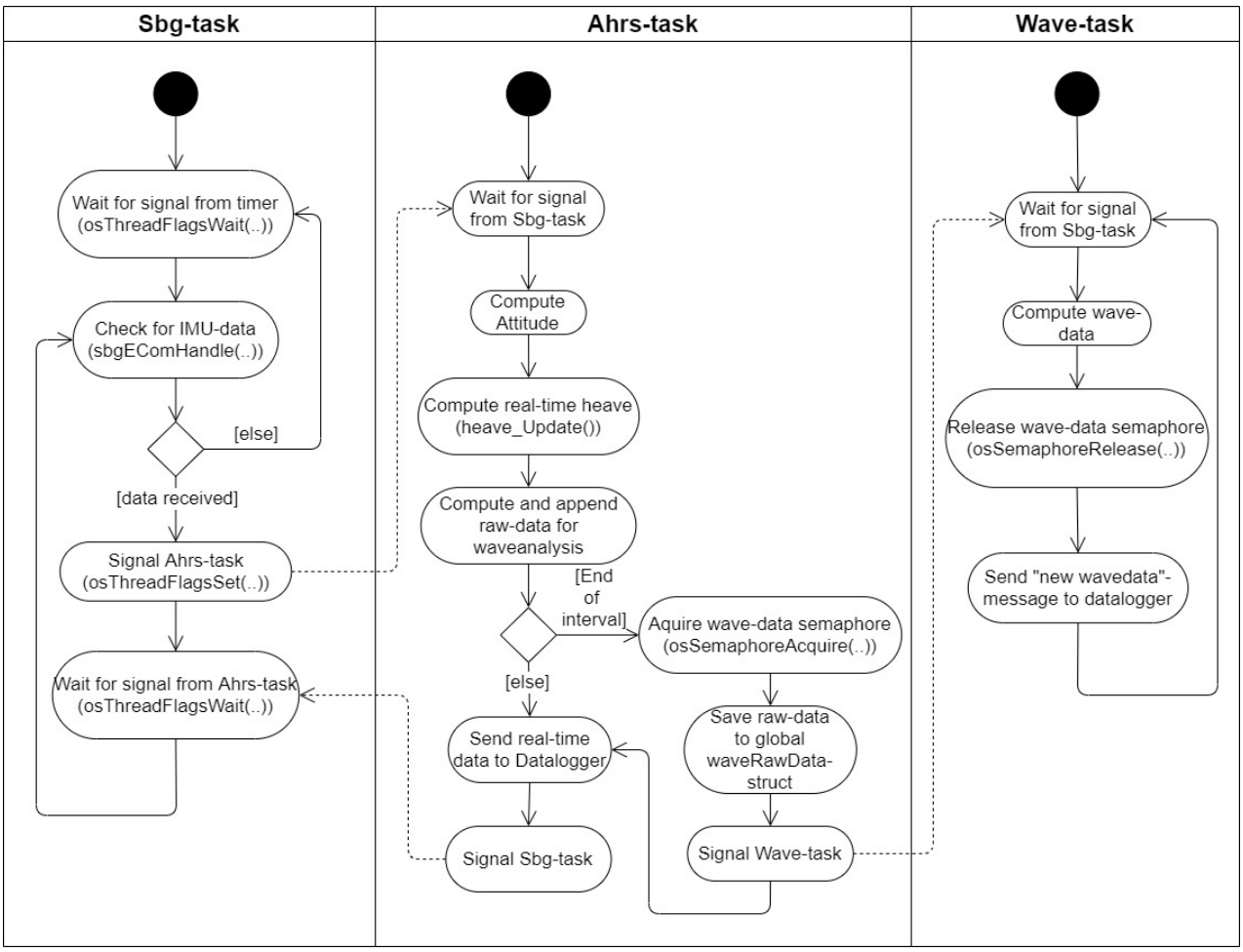
Figur 3.7.: Klasse-diagram

Implementeringen bruker idéer fra objektorientert programmering, slik som objekter og innkapsling. Hver fil kan sees på som en klasse eller objekt, ved at de utgjør en enhet med både data og funksjonalitet. Innkapsling brukes ved at bare noen av data-feltene og metodene i hver fil er *offentlige*, altså tilgjengelig fra andre filer. *Private* metoder og variabler er definert med nøkkelordet `static` slik at de er utilgjengelig for andre filer. Figur 3.7 viser et UML klassediagram med offentlige variabler og funksjoner for de viktigste filene. Alle offentlige variabler er ikke-skrivebare unntatt `txMsg` som er lesbar og skrivbar. Disse variablene er derfor implementert som `const`-pekere til sine private (static) variabler. Et alternativ til dette kunne vært å bruke aksessor-funksjoner.

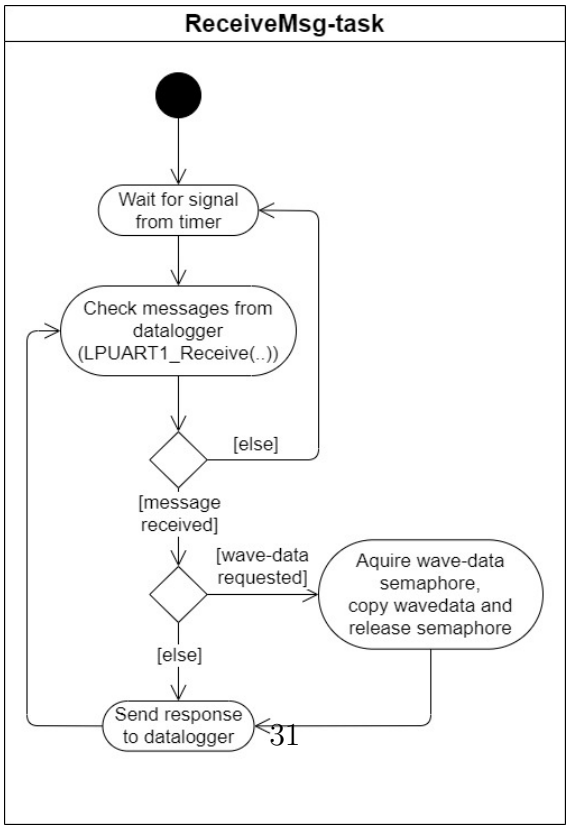
I klasse-diagrammet viser pilene med stiplet linje, avhengighet mellom de ulike filene. For eksempel ser vi at filen `heave.c`, som beregner sanntids hiv/bølgehøyde, henter akselerasjonsmålingen `acc` fra filen `sbg.c` og orienteringen `quat` fra filen `ahrs.c`. `heave.c` har en variabel `heaveRT` og en metode `void Update()` som aksesseres og kalles fra filen `ahrs.c`.

Figur 3.8 og 3.9, viser UML aktivitetsdiagram for de fire trådene i programvaren. Trådene kommuniserer ved bruk av to binære semaforer, ved kall til funksjonene `osSemaphoreRelease(..)` og `osSemaphoreAcquire(..)` . `waveDataSem` brukes til å beskytte variablene `WaveData_t waveData` og `WaveRawData_t rawData`, og `transferDataSem` beskytter variabelen `Msg_t txMsg` og `LPUART1_Transmit()` (se klasse-diagram, figur 3.7). Trådene kommuniserer også ved å sende signaler til hverandre med funksjonene `osThreadFlagsWait(..)` og `osThreadFlagsSet(..)`.

To FreeRTOS timere sender signal til henholdsvis Sbg-tråden og ReceiveMsg-tråden, hvert hundredels-sekund, slik at disse begynner å kjøre. Sbg-tråden sjekker da om det har kommet sensor-data fra IMUen. Hvis det har det, så sender den melding til Ahrs-tråden om at den skal kjøre. Ahrs-tråden beregner da orientering og sanntids bølgehøyde. Så kalles funksjonen `waveRawData_Update()`, som beregner og lagrer rådata til bølgeanalyse. Hvis dette er siste verdi i analyseperioden på 17 minutter, så hentes `waveDataSem`-semaforen og rådataene blir gjort tilgjengelig i pekeren til `rawData`, før wave-tråden signaliseres. Deretter sender Ahrs-tråden sanntidsdata til datalogger. Når wave-tråden blir signalisert, så beregner den bølgedata og bølgestatistikk, før den frigir `waveDataSem`-semaforen og sender melding til data-logger om at bølgedata for neste periode er klar.



Figur 3.8.: Aktivitets-diagram for Sbg-, Ahrs- og Wave-tråden



Figur 3.9.: Aktivitets-diagram for ReceiveMsg-tråden

3.5.3. Protokoll for kommunikasjon med datalogger

Som kommunikasjonsprotokoll mellom bølgesensor og datalogger, ble det valgt en protokoll fra software-pakken X-CUBE-MEMS1 [23] fra ST. Se vedlegg A.2 for kildekoden til denne protokollen. Som vi ser fra tabell 3.1 så består en melding av først en kommando på 1 byte. De ulike kommandoene er oppgitt i vedlegg A.2.2 En kommando kan ha verdi fra 0 til 63. Neste del i meldingen er eventuelle data som skal overføres. Denne delen kan ha en variabel lengde fra 0 til 2048 bytes. Tilslutt kommer en sjekksum på 1 byte som tilsvarer det negative av resten av meldingen, slik at summen av hele meldingen blir 0. Sjekksummen er for å sjekke om meldingen har endret seg under overføringen på grunn av feil. Kommunikasjon mellom datalogger og bølgesensor er to-veis, slik at begge kan sende og motta meldinger. Hvis datalogger eller bølgesensor skal svare på en melding, sendes en melding tilbake med samme kommando, men der man addere verdien 0x80 til kommandoen. Hvis det derimot ikke er mulig å gi et meningsfullt svar på meldingen, så sendes i stedet en melding der verdien 0x40 adderes til kommandoen. Data-delen har ulik lengde avhengig av kommando.

Kommando	Data	Sjekksum
1 byte	$\leq$ 2048 bytes	1 byte

Tabell 3.1.: Protokoll for kommunikasjon med datalogger

Et eksempel på en kommando er `CMD_LOG`, som brukes når bølgesensor sender sanntidsdata til datalogger. For denne kommandoen er data-delen alltid 60 bytes, og datalogger vil ikke svare på meldingen

Kommandoen `CMD_MSG` brukes når bølgesensor vil sende tekstmelding til datalogger. Lengden til data-delen avhenger av lengde på meldingen, og datalogger vil ikke gi noe svar. Denne meldingen kan for eksempel brukes til å sende feilmeldinger, for å gjøre feilsøking enklere.

Kommandoen `CMD_HEAVE` brukes av datalogger når den vil spørre etter tidsserien av hiv/bølgehøyde fra forrige periode. Datadelen består av 4 bytes med to positive heltall av typen `uint16_t`. Disse to tallene gir start- og stoppindeks for den delen av tidsserien man ønsker å hente fra bølgesensor, som gjør at man har mulighet til å overføre tidsserien i mindre deler, slik at bølgesensor får mulighet til å sende sanntidsdata med kort forsinkelse. Bølgesensoren svarer på meldingen med å sende den etterspurte delen av hiv-tidsserien.

For at mottakeren skal kunne forstå meldingene som sendes, så må den vite hvor meldingen starter og slutter. Derfor sendes en byte med verdien `0xF0`, etter meldingen. Da vet mottakeren når meldingen er slutt, og dermed vet den også at neste byte er starten på neste melding. Men hvis meldingen allerede inneholder en byte med verdien `0xF0` fra før, så får man et problem. For å løse dette brukes en teknikk som heter byte-stuffing. Kort fortalt så endrer man meldingen slik at hver byte med verdien `0xF1` bytter man ut

med to bytes lik `0xF1F1`, og hver byte med verdien `0xF0`, bytter man ut med to bytes lik `0xF1F2`. Når mottaker har mottatt meldingen, så endres meldingen tilbake ved å bytte ut `0xF1F1` med `0xF1`, og `0xF1F2` med `0xF0`.

Denne protokollen ble valgt fordi den er enkel. Sjekksummen for feilsjekking kan byttes ut med mer avanserte metoder, slik som syklisk redundanssjekk, hvis det er behov for noe mer robust. Byte-stuffing teknikken gjør at meldingene får en noe uforutsigbar lengde, noe som kanskje kanskje kan være en ulempe ved for eksempel dekoding av meldingen på oscilloskop. SBG-sensoren bruker i stedet for byte-stuffing en protokoll med to faste synkroniseringsbytes først i meldingen, en fast byte til slutt, og et felt med to bytes i begynnelsen, som gir lengden til hele meldingen. Ulempen med denne protokollen er at den virker mer komplisert, og at hvis verdien på lengde-feltet endres på grunn av feil under overføring, så risikerer man stor forsinkelse i overføringen. Dette var noe som skjedde under arbeid med oppgaven. I begynnelsen av oppgaven så ble SBG-sensoren koblet til mikrokontrolleren, ved å bruke en RS232-TTL-konverterer som ble koblet direkte til den ene D-sub kontakten på SBG-kortet. Det oppstod da mye støy på overføringssignalet, som kanskje skyldtes en dårlig konverterer, som siden ble defekt, eller at konverteren ble koblet for nært elektronikken på SBG-kortet. Uansett så førte støyen til at meldingen ofte endret seg under overføring. SBG-protokollen kan sende meldinger på opptil 4086 bytes, så hvis denne feilen ga endringen i lengde-feltet i meldingen, så ble verdien typisk mye større. Dette førte til at overføringen stoppet opp i mange sekunder, mens mikrokontrolleren ventet på at den lange meldingen skulle bli ferdigsent, eller at man fikk time-out i driveren, samtidig som SBG-sensoren fortsatte å sende korte meldinger som aldri ble mottatt. Dette er en svakhet i protokollen som ikke er mulig med byte-stuffing protokollen fordi da vil meldingen termineres med hver ende-byte `0xF0`. Hvis ende-byten blir korrupt vil meldingen termineres med ende-byten til neste melding.

3.5.4. Pålitelighet og vedlikeholdbarhet

Slik som forklart er programvaren inndelt i filer som fungerer som løst koblede moduler, og med bruk av innkapsling, noe som gjør programvaren mer pålitelig og vedlikeholdbar. Et annet moment er bruk av kodegenerator for konfigurasjon og initialisering av mikrokontrolleren. Dette gjør at kodeprosjektet kan åpnes i kodegeneratoren for å modifisere konfigurasjonen eller legge til og konfigurere flere periferienheter. Ved å se på koden som ble generert for initialisering av UARTen mot datalogger, i vedlegg A.1 forstår man at det å konfigurere dette i kodegeneratoren kan være betydelig enklere.

Foreløpig er det ikke implementert noe feilhåndtering i programvaren. Siden man er på et tidlig stadium i utviklingen, er det heller vektlagt å tilrettelegge for feilrapportering. Dette gjøres ved at alle trådene har mulighet til å sende tekstmeldinger for informering eller feilrapportering til datalogger, med kommandoen `CMD_MSG`.

Et annet tiltak for å gjøre programvaren mer pålitelig er å bruke kompilator minnebarrierer (`__asm volatile( "" ::: "memory" );`) i forbindelse med multitasking i FreeRT-

OS. Dette er for å unngå at problemer oppstår ved kompilator-optimalisering . Ved testing ble kildekoden kompilert med gcc-flagget -O3 som gjør at kompilatoren optimaliserer koden mest mulig. Et potensielt problem kan da være at kompilatoren flytter rundt på instruksjonene i kildekoden på en måte som skaper problemer for synkroniseringen mellom tråder i FreeRTOS (se [9]). Derfor brukes en kompilator minnebarriere *etter* hver henting av semafor eller mottak av signal fra tråd, og *før* hver frigivelse av semafor eller sending av signal til tråd.

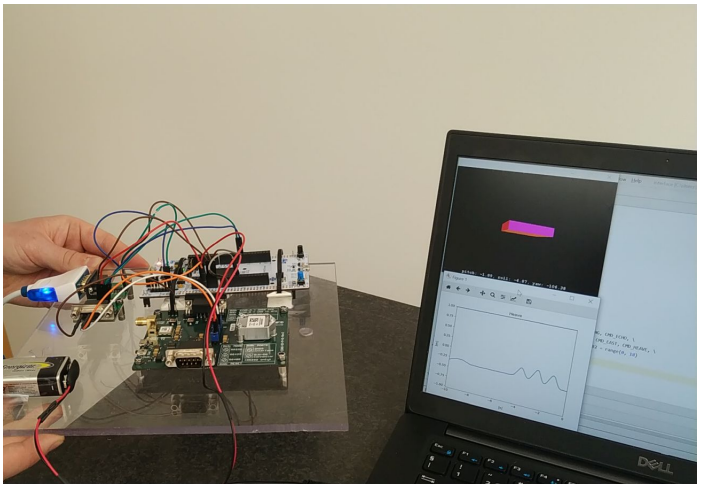
4. Resultater

4.1. Test av bølgesensor

For å teste bølgesensoren ble en PC med Windows brukt som datalogger. En driver ble implementert i python til kommunikasjon med bølgesensor. Denne driveren mottar sanntids-data og visualiserer hiv og orientering. Kode for sanntids visualisering av orientering ble funnet på Github, se [5]. Når driveren får beskjed om at bølgedata er ferdigberegnet, laster den ned all bølgedata fra bølgesensoren ved polling/spørring.

4.1.1. Visuell test

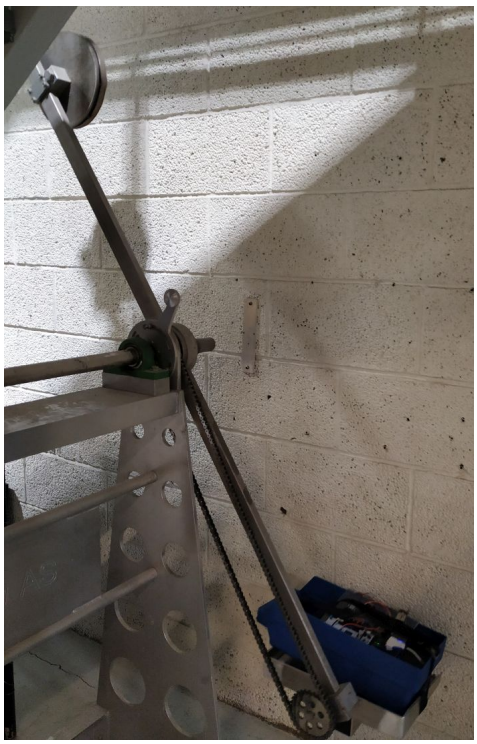
Real-time visualisering av bølgehøyde og orientering ble brukt som test for å se at estimeringen av bølgehøyde og orientering i sanntid fungerte etter forventningene. Det ble også gjort testing med en magnet, for å se hvordan forstyrrelser i magnetfelt virket inn på estimatene. Testen viste at AHRSen fungerte bra og magnetfeltforstyrrelser ikke hadde noen innvirkning på bølgehøyde, rull eller stamp. Det ble samtidig gjort test av standard implementasjonen av ECF-observeren fra seksjon 2.2.1, og testen demonstrerte at magnetfelt-forstyrrelsene gav store avvik i estimering av bølgehøyde, rull og stamp. De visuelle testene viste også at bølgehøyde/hiv-observeren fungerte brukbart. Videoklipp som viser de visuelle testene er vedlagt rapporten. Noe som ble observerte, men som ikke blir vist i videoklippene er at når bølgesensoren blir snudd opp ned over lengre tid, så vil bølgehøyden drifte vekk, fordi biasen-tilstanden da må estimeres på nytt.



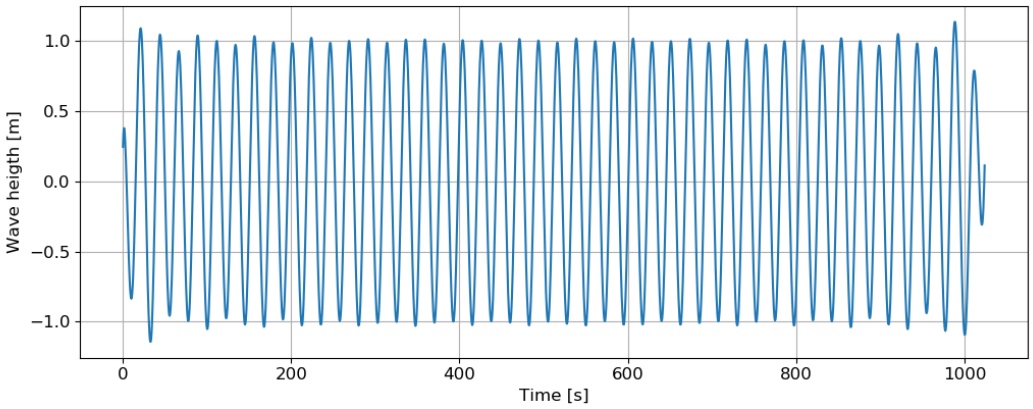
Figur 4.1.: Visuell test

4.1.2. Karusell-test

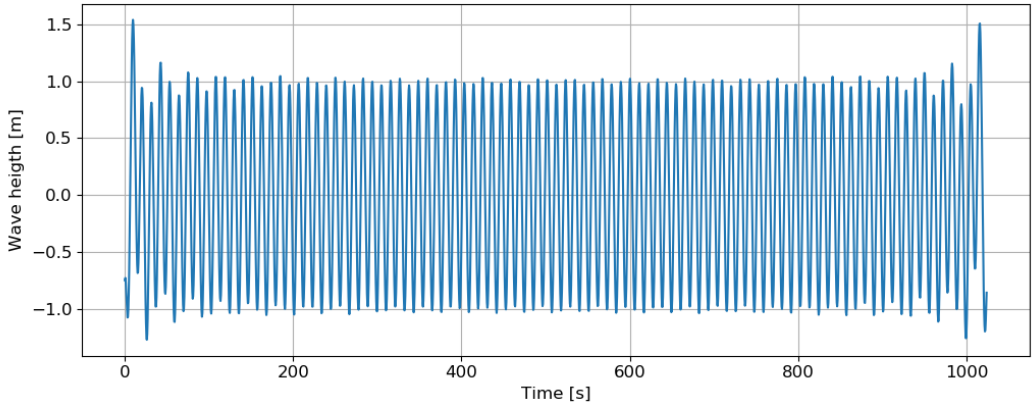
Bølgesensoren ble også testet i en test-«karusell» hos samarbeidsbedriften Fugro, slik som vist i figur 4.2, og i et vedlagt videoklipp. Test-karusellen roterer bølgesensoren i en vertikal sirkel bevegelse med radius lik 2 meter, i konstant hastighet. Testen ble utført for å teste estimeringen av bølgehøyden i tids- og frekvensdomenet for en estimeringsperiode på 17 minutter. Bølgehøyden i tidsdomenet ble også sammenlignet med sanntidsestimeringen av bølgehøyden. Testen ble utført to ganger, for simulering av bølgeperiode på 11 og 22 sekunder. Testen viste at estimering av bølgedata fungerte etter forventningene.



Figur 4.2.: Test-karusellen

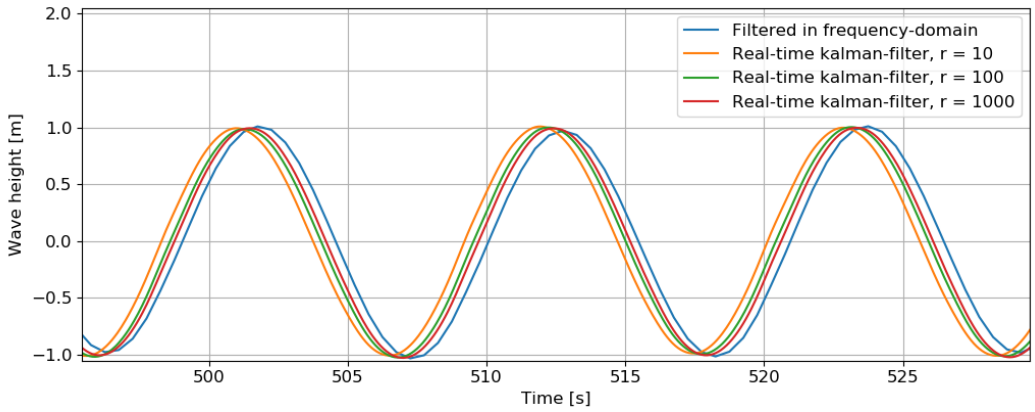


Figur 4.3.: Hiv for 22s-periode

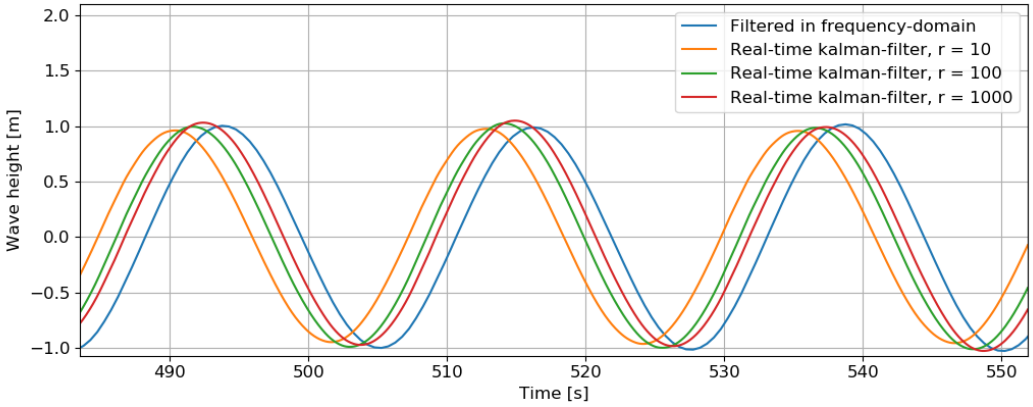


Figur 4.4.: Hiv for 11s-periode

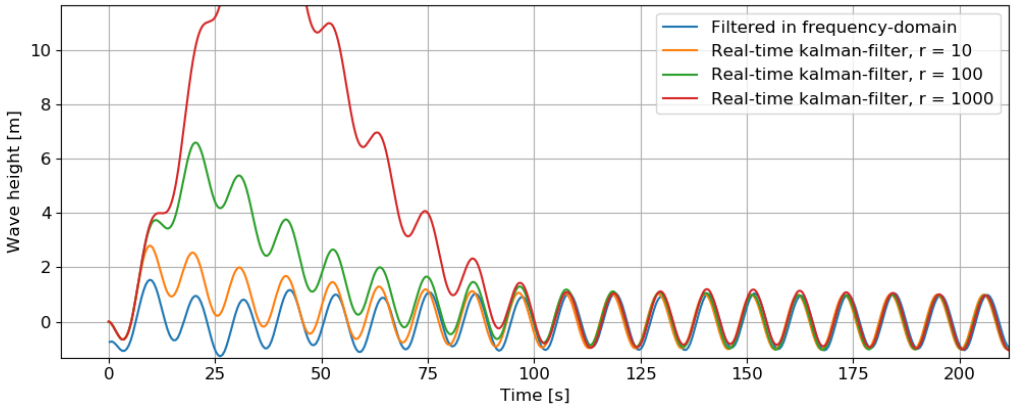
Vi ser av figur 4.3 og 4.4 at bølgehøyde-estimatet i tidsdomenet får oscillerende forstyrrelser på kantene av signalet (begynnelse og slutt). Denne forstyrrelsen er en slags transient, og som vi ser av figur 4.7 er denne forstyrrelsen mindre enn transienten vi får ved bruk av sanntidsestimering. Filtreringen i frekvensdomenet bruker et ideelt høypassfilter med cut-off frekvens på 35 sekund. Dette er et ikke-kausalt filter med en symmetrisk impulsrespons i form av en sinc-funskjon. I motsetning til sanntidsobserveren får vi derfor en også en transient på slutten av signalet. På grunn av denne forstyrrelsen bør man kutte vekk kantene når man estimerer bølgehøyden i tidsdomenet på denne måten, ved å for eksempel fjerne første og siste seksten-del av signalet.



Figur 4.5.: Hiv for 11s-periode

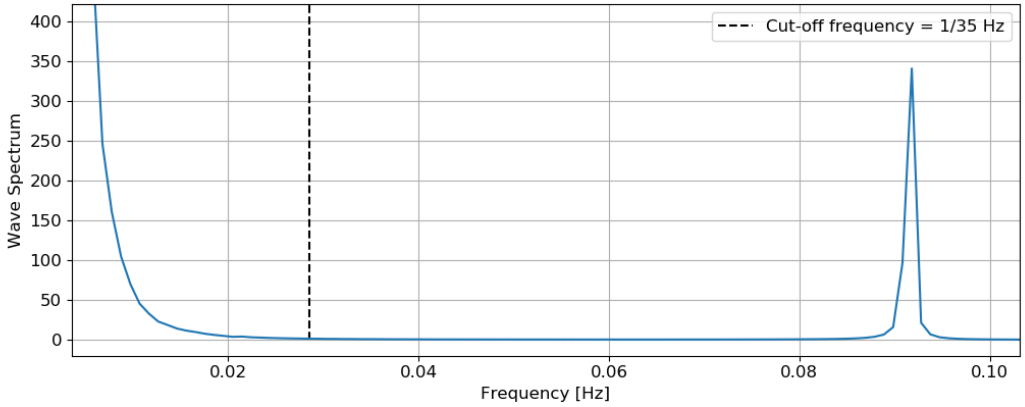


Figur 4.6.: Hiv for 22s-periode



Figur 4.7.: Hiv for 11s-periode

Fra figur 4.6 og ser vi at sanntidsestimeringen gir en faseforskyvning. Denne faseforskyvningen blir mindre, når frekvensen på bølgene øker, slik vi ser av figur 4.5. Man kan øke målestøy-faktoren r (eller minke prosess-støy-faktoren q) i filteret for å få mindre faseforskyvning, men da vil vi også få tregere innsvingning av filteret slik vi ser av figur 4.7. Ved å øke r (eller minke q) vil man også få en lavere cut-off frekvens, slik at langsomme frekvenser blir mindre dempet.

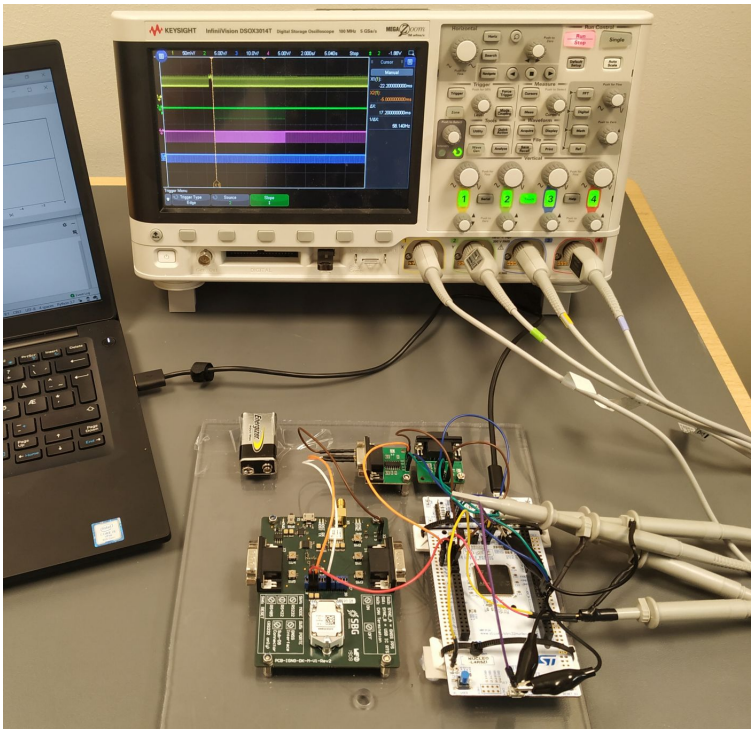


Figur 4.8.: Ufiltrert hiv-spektrum for 11s-periode

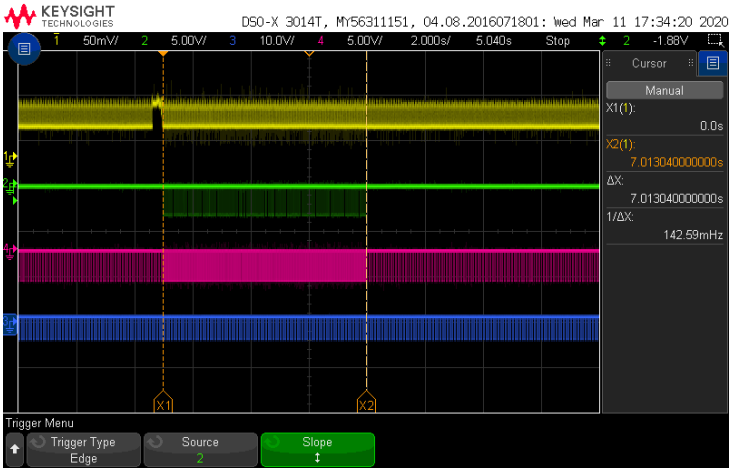
Figur 4.8 viser at frekvensspekteret av bølgehøyden, uten filtrering. Vi ser at etterhvert som perioden øker forbi 30 sekunder, vil hiv-signalet få stadig mer støy. Dette skyldes støy fra akselerometerene som dobbeltintegreres og derfor blir dominerende for lave frekvenser. Se seksjon 4.2.1 for mer informasjon om denne støyen.

4.1.3. Strømforbruk og responstid

Kommunikasjon mellom bølgesensor og datalogger, ble også testet ved bruk av oscilloskop, slik vi ser av figur 4.9. Oscilloskopet ble lånt fra Fugro. De fire probene på oscilloskopet ble koblet til mottaker og sender til datalogger, og mottaker fra SBG-sensor. Ved å se på og sammenligne disse signalene, fant man blant annet ut hvor lav forsinkelse man hadde, ved beregning av, og overføring av sanntidsdata. En probe ble også koblet over en 51 ohms motstand festet til jumper J5 (IDD) på Nucleo-kortet, som går til strømforsyningen på mikrokontrolleren. På denne måten kunne man måle strømforbruket. Fra strømforbruket kunne man se når mikrokontrolleren var i kjøre-modus eller sove-modus, og ved å sammenligne med kommunikasjonen med datalogger og SBG-sensor, fikk man innblikk i hva mikrokontrolleren jobbet med og hvor lenge.



Figur 4.9.: Oscilloskop-test

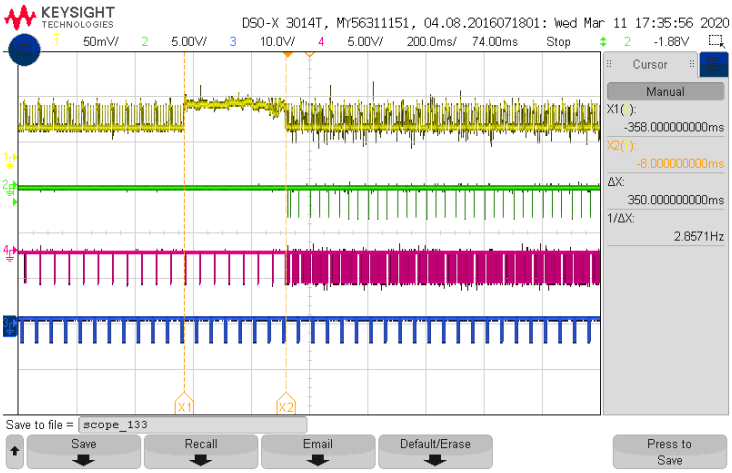


IDD DL_RX DL_TX SBG_RX

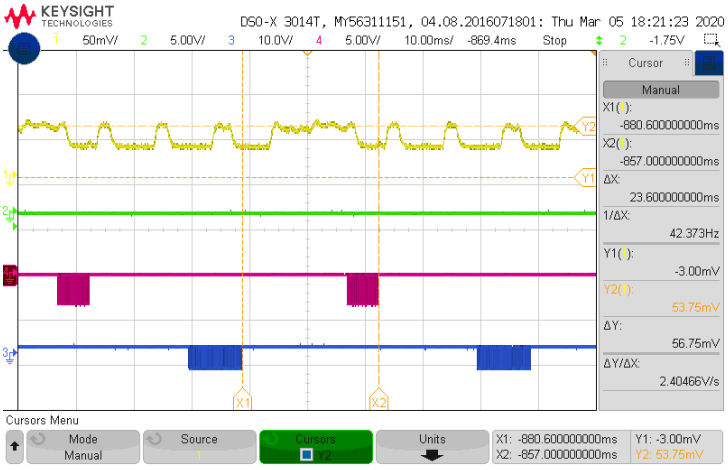
Figur 4.10.: Overføringstid med -O3

Av figur 4.10 ser vi at overføringstiden for bølgedata er 7.0 sekunder når vi bruker -O3 optimalisering for kompilator. Uten optimalisering (-O0) blir overføringstiden 10.5 sekunder (se figur B.1 i vedlegg). Det som her overføres er all data i variablene `WaveRawData_t rawData` og `waveData_t waveData` fra figur 3.7.

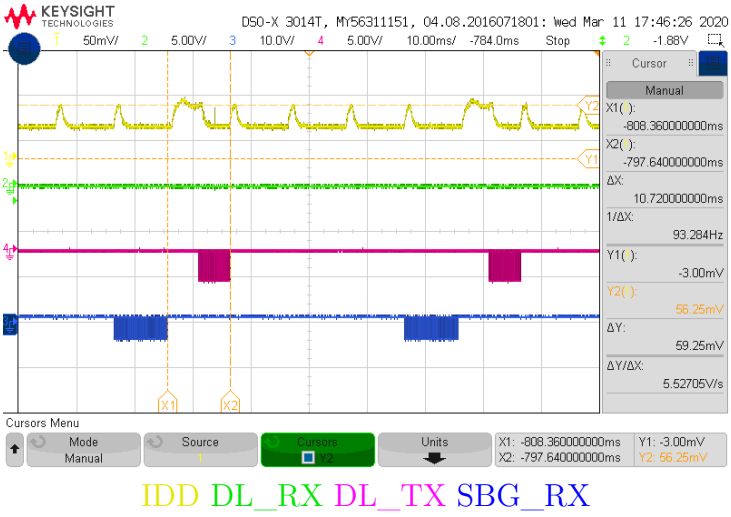
Fra figur 4.11 ser vi at forsinkelse for beregning av bølgedata er 350 ms med optimalisering, mens uten optimalisering blir forsinkelsen 660 ms (se figur B.2). Dette er beregningene i funksjonen `void wave_Calc()` fra vedlegg A.6, som tilsvarer python-koden i vedlegg A.8. Vi ser altså at overføringen av bølgedata tar mye lengre tid enn selve beregningen. Vi ser også at ved full kompilator-optimalisering (-O3) av koden, så går både beregningen og overføringen betydelig raskere, enn uten optimalisering (-O0).



Figur 4.11.: Bølgeberegningstid med -O3



Figur 4.12.: Forsinkelse med -O0

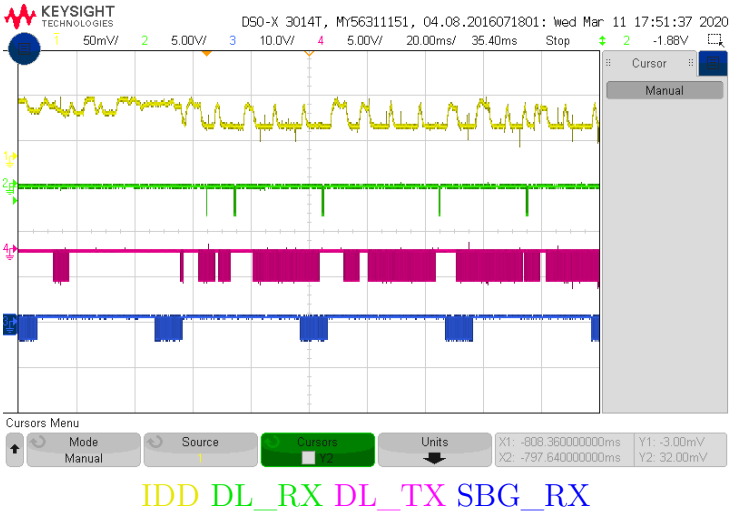


Figur 4.13.: Forsinkelse med -O3

Fra figur 4.13 og figur 4.12 ser vi at forsinkelsen for real-time data er 10.7 ms med optimalisering (-O3) og 23.6 ms uten. Av strømforbruket ser vi at MCU våkner opp av SLEEP med intervall på 10 ms, for å sjekke innkommende data fra SBG sensor (SBG_RX) eller datalogger (DL_RX).

Strømforbruket til MCU måles ved at forsyningsstrømmen går gjennom en 51 ohms motstand. Spenningen i kjøremodus og sove-modus måles til henholdsvis 59 mV og 35 mV. Ved Ohms-lov får vi at strømforbruket til MCU i kjøre- og sove-modus er på henholdsvis 1.2 mA og 0.7 mA, ved 3.3 V forsyningsspenning. Siden MCU er i sove-modus 82 % av tiden blir gjennomsnittlig strømforbruk 0.77 mA, som gir en effekt på cirka 2.5 mW, når man kjører på 4 MHz og kompilerer med -O3 optimalisering.

Effektforbruket ble også målt for både SBG-sensor og MCU med et vanlig amperemeter (FLUKE 189). SBG-sensoren (som er oppgitt til å bruke 500 mW) ble målt til å bruke 51 mA @ 5.0 V = 255 mW, mens MCU ble målt til 0.70 mA @ 3.3 V = 2.3 mW, noe som stemmer bra med det som ble estimert utifra oscilloskop-målingen. SBG-sensoren har altså et energiforbruk som er mer enn 100 ganger større enn MCUen. For eksempel på et vanlig smart-mobil batteri på 3.7 V med 3000 mAh, betyr det at SBGen ville vart i cirka 43 timer, mens MCUen ville vart i cirka 4800 timer som er nesten 7 måneder.



Figur 4.14.: Dataoverføring med -O3

Fra figur 4.14 ser vi hvordan datalogger laster ned bølgedata, samtidig som den mottar sanntidsdata. Vi ser at nedlastingen starter når MCU sender melding (kommando `CMD_NEW_DATA`) til datalogger, etter at bølgedata er ferdig beregnet. Da begynner datalogger og laste ned bølgedata ved spørring. Første datapakke som lastes ned består av bølgeparametrene og er derfor mindre, enn resten av pakkene som består av dataserier. Når datalogger laster ned lange dataserier, laster den bare ned en del av dataserien om gangen ved å spesifisere start- og stopp-indeks. Driveren til datalogger er satt til å laste ned 64 verdier om gangen. Siden verdiene er i `float` blir det $64 \cdot 4 = 256$ bytes. Når MCU først begynner å overføre meldingen med bølgedata vil den forsette til hele meldingen er overført. Derfor må ikke driveren laste ned for lange datapakker om gangen hvis den vil ha god responstid. Av figuren ser vi at den siste meldingen med sanntidsdata har en forsinkelse på cirka 24 ms, fordi MCU er opptatt med å overføre bølgedata. Av strømforbruket (IDD) ser vi at MCU da går i sove-modus, mens Ahrs-tråden venter på at transferData-semaforen skal bli ledig. Dette viser at skeduleringen fungerer slik som forventet.

4.1.4. Kjøretid og minneforbruk

For å finne mye tid hver tråd brukte på å kjøre, og minneforbruket til hver tråd, ble bølgesensoren først kjørt i to intervaller inkludert 2 overføringer av bølgedata, som tilsvarer cirka 2060 sekunder (se figur B.3 og figur B.4 i vedlegget). Deretter ble MCU stoppet med debugger, og for å finne ledig stack-minne ble FreeRTOS-funksjonen `UBaseType_t uxTaskGetStackHighWaterMark( TaskHandle_t xTask )` brukt. Resultatene er oppgitt i tabell 4.1 og 4.2.

Tråd	Kjøretid (-O0)	Kjøretid (-O3)
Sbg	15.6 %	4.9 %
Ahrs	9.5 %	5.4 %
Wave	0.035 %	0.027 %
ReceiveMsg	2.5 %	1.5 %
Idle (sove-modus)	60.7 %	82.4 %
Timer	11.7 %	5.7 %

Tabell 4.1.: Kjøretid for tråder

Vi ser av tabell 4.1 at veldig liten tid totalt går med til å beregne bølgedata. Driveren fra SBG bruker forholdsvis mye ressurser, men har også god nytte av optimaliseringen, og kjører da 3 ganger hurtigere.

Tråd	Stack-minne allokert	Brukt stack	Prosent brukt
Sbg	20 kB	8.7 kB	43 %
Ahrs	2 kB	3.2 kB	20 %
Wave	2.4 kB	3.6 kB	25 %
ReceiveMsg	1.2 kB	1.9 kB	19 %

Tabell 4.2.: Minneforbruk for tråder (med -O0)

Fra tabell 4.2 ser vi at med unntak av Sbg-tråden så bruker trådene lite minne i forhold til kapasiteten til RAM på 640 KB. Det er fordi de store dataseriene ikke lagres på stacken, men i statisk allokert minne, det vil si minne allokert ved kompileringstid. Når man lagrer minne på stack, må man passe på at den ikke overfylles, og det er derfor lagt til en margin på 4-5 ganger, for sikkerhetsskyld. Sbg-tråden derimot bruker driveren fra SBG som oppretter lange dataserier på stack, som gjør at den bruker hele 8.7 kB med minne. For SBG-tråden brukes en noe lavere margin.

	RAM	RAM prosent	Flash	Flash prosent
Kapasitet	640 KB	100 %	2048 KB	100 %
Konstanter			80.6 KB	3.9 %
Kode			60.4 KB	2.9 %
Data initialisert (4096 samples)	48.4 KB (96.4 KB)	7.6 % (15.1 %)	48.4 KB (96.4 KB)	2.4 % (4.7 %)
Data ikke-initialisert (4096 samples)	137.8 KB (225.8 KB)	21.5 % (35.3 %)		
Annet	1.5 KB	0.2 %	0.4 KB	0.0 %
Sum (4096 samples)	187.7 KB (323.7 KB)	29.3 % (50.6 %)	189.8 KB (237.9 KB)	9.3 % (11.6 %)

Tabell 4.3.: Minneforbruk med 2048 og 4096 samples med -O3

	RAM	RAM prosent	Flash	Flash prosent
Kapasitet	640 KB	100 %	2048 KB	100 %
Sum Brukt	187.8 KB	29.4 %	215.8 KB	10.5 %

Tabell 4.4.: Minneforbruk med 2048 samples med -O0

Når koden er kompilert kan minneforbruk leses av i *Build Analyzer*-vinduet til IDEen (se figur B.5, B.7 og B.6 i vedlegg). Resultatet er oppsummert i tabell 4.3 og 4.4. Vi ser at når vi bruker et estimeringsintervall på 4096 samples (som tilsvarer 34 minutter på 2 Hz) så brukes 237.9 KB av RAMen på 640 KB, som tilsvarer 50.6 %. Det viser at vi har enda mer å gå på med 640 KB RAM. Likevel er det litt knapt, og ved utvidelse av programvaren med flere funksjoner, bør man kanskje vurdere å utvide kapasiteten.

4.2. Testing på rådata

4.2.1. Rådata

For testing og justering av AHRSen ble observeren testet på rådata fra en av samarbeidsbedriften Fugro sine målebøyer. Ideelt sett burde bølgeestimatoren vært testet på virkelige rådata målt med SBG-sensoren. Fordi slike data ikke har vært tilgjengelig og kunne tatt mye tid å fremskaffe, har vi i denne oppgaven heller testet bølgeestimatoren på rådata bestående av akselerometer-, gyro- og magnetometer-målinger, fra en SEA-WATCH Wind Lidar Buoy fra Fugro, se figur 4.15. Disse rådataene var tilgjengelig i form av filer, der hver fil inneholder 8192 samples på 6 Hz per tidsserie, som tilsvarer 1365.33 sekunder eller cirka 23 minutter. Det finnes én fil for hvert 1024 sekund, som

betyr at rådata i filene overlapper slik at den siste sjettedelen av hver fil, overlapper med den første sjettedelen av neste fil. Rådataene som ble brukt kommer fra en målebøye lokalisert i Sula-fjorden. Det ble gjort beregninger for tidsperioden i en periode på cirka 24 timer, fra 2020-01-08 11:56 til 2020-01-09 12:09 (GMT), som tilsvarer 144 filer. Alle grafene i tidsdomenet viser estimater i perioden fra mellom kl 23:46 til 00:09 (GMT). Disse tidsperiodene ble valgt, fordi de hadde mest bølger. Dette er en fordel ved testing av AHRSen fordi vinkelutslagene på flytende målebøyer, normalt sett er ganske små. Fra figur 4.19 ser vi at signifikant bølgehøyde var på mellom 2.1 og 3.5 meter i denne perioden, mens gjennomsnittlig bølgeperiode var mellom 7.3 og 10.5 sekunder. Variasjonen til rull, stamp, gir og hiv ble funnet ved å beregne kvadratisk gjennomsnitt (RMS) over standardavvikene til hver periode, og man fikk da 5.7° for rull, 3.4° for stamp, 5.2° for gir, og 0.68 meter for hiv/bølgehøyde.

Rådata ble brukt til å teste estimering av bølgedata, orientering og hiv ved bruk av programmeringsspråket python. Hiv ble beregnet i frekvensdomenet. Når beregningene ble gjort, ble rådata-filene behandlet hver for seg. Dermed trengte man ikke ta hensyn til at rådata ikke var sammenhengende, ved at filene overlapper med hverandre. Beregningene kunne også gjøres mye rasker ved å bruke multiprosessering i python på denne måten:

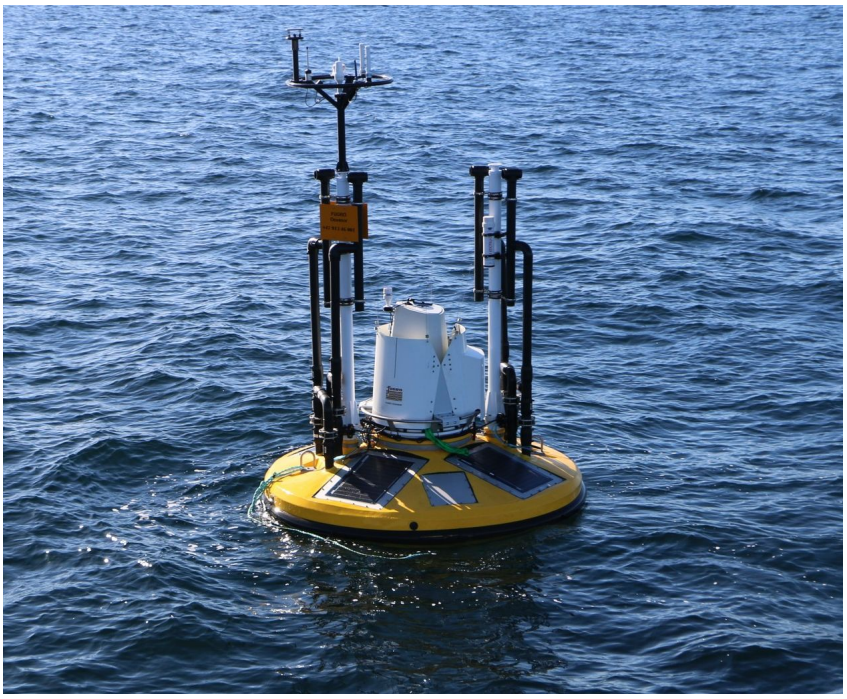
```
from itertools import repeat
from multiprocessing import Pool

with Pool() as pool:
  results = pool.starmap(Estimator.run_file, zip(repeat(estimator),
     rawFiles))
```

Ved beregning av frekvensspektrum og sammenligning av avvik mellom ulike tidsserier, ble det bare sett på estimatene for siste halvdel av hver fil. Dette fordi AHRS-observeren bruker tid på konvergere. Dette utgjør 4096 samples per fil som blir 11 minutt og 23 sekund. Når man regner med overlappingen mellom filene betyr det bare ble beregnet estimater for 2/3-deler av tiden. Ved sammenligning av tidsserien til bølgehøyden, ble transientene fjernet, ved at første og siste seksten-del av signalet ble fjernet.

Det var først etter litt over to måneder etter oppstart av oppgaven at disse rådata ble tilgjengelig. I begynnelsen av oppgaven ble det gjort beregninger på andre rådata, der bølgehøyden var vesentlig mindre. Sammen med de nye målingene, ble det også gjort GNSS-målinger, med mulighet for å beregne RTK(Real Time Kinematics)-korrigerte posisjonsmålinger fra GNSS-sensoren i masta på bøya, montert cirka 3 meter over IMU-sensoren. Dette kunne gitt en god mulighet for å vurdere nøyaktigheten av estimatene, men fordi rådata ble tilgjengelig såpass sent ut i oppgaven, ble det ikke gjort noen slik sammenligning. Det var derfor en utfordring at man ikke hadde noen ideell referanse å vurdere observeren mot. En slik sammenligning av observeren med GNSS-RTK vil være et godt utgangspunkt for videre arbeid med bølgesensoren.

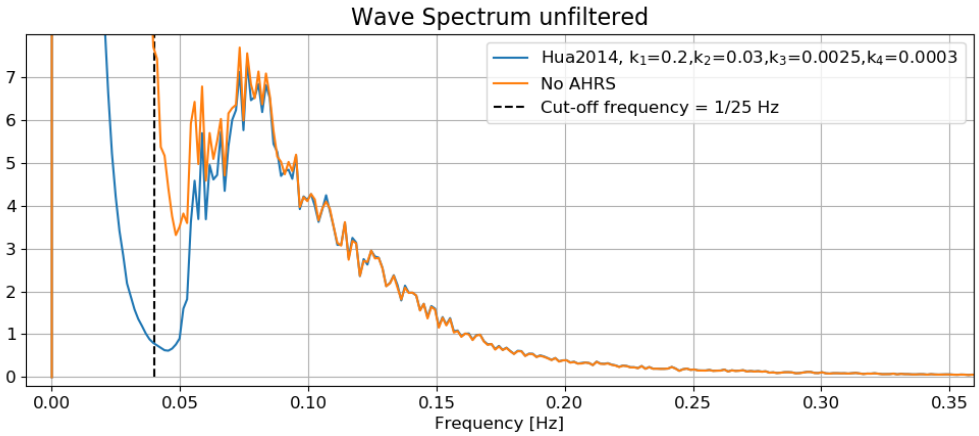
Akselerometer-støy

I figur 4.16 ser vi gjennomsnittet av bølgespektrene mellom 11:56 den 8. januar til kl 12:09 9. januar. Vi ser vi at for frekvenser under 0.04 Hz vil det estimerte bølgespekteret bli stadig mer dominert av støy. Cut-off frekvensen for høy-pass filteret ved beregning av bølgehøyde og bølgeparametre ble satt til 0.04 Hz basert på at støyen ser ut til å bli dominerende når frekvensen blir lavere enn dette. Hvis vi antar en perfekt AHRS, slik at akselerometeret er orientert helt riktig, så vil denne støyen bestå av støy fra akselerometeret. En enkel måte å modellere akselerometer-støyen på er å anta at man har hvit støy pluss en konstant bias. Før man beregner frekvensspekteret av akselerasjonen blir gjennomsnittet trukket i fra, og dermed kan man til dels se bort i fra biasen. Men hvis den konstante biasen er forskjellige i de tre ulike retningene, og orienteringen til akselerometeret varierer mye, så vil også biasen kunne bidra til en varierende støy. Det virker likevel rimelig å anta at den varierende støyen fra akselerometeret er hvit støy. Når man integrerer hvit støy får man Brownsk støy, som er en type støy produsert av en Wiener prosess, også kalt «Random Walk». Energi Spekteret til Brownsk støy er gitt ved $S(\omega) = \frac{S_0}{\omega^2}$ der S_0 er konstant, og ω er frekvensen. For å finne hiv, dobbelt-integreres akselerasjonen, og energispekteret til støyen blir $S(\omega) = \frac{S_0}{\omega^4}$, som tilsvarer Brownsk støy opphøyd i andre. Som vi ser av formelen vil støyen derfor bli uendelig stor når frekvensen går mot 0. Heldigvis er denne støyen typisk ikke noe stort problem, fordi for lave frekvenser der støyen begynner å bli stor, så vil det være lite med bølger. Dette illustreres også av figuren, der vi ser at bølgespekteret faller brått en stund før det begynner å stige kraftig på grunn av støy.



Figur 4.15.:

4.2.2. Sammenligning med og uten AHRS



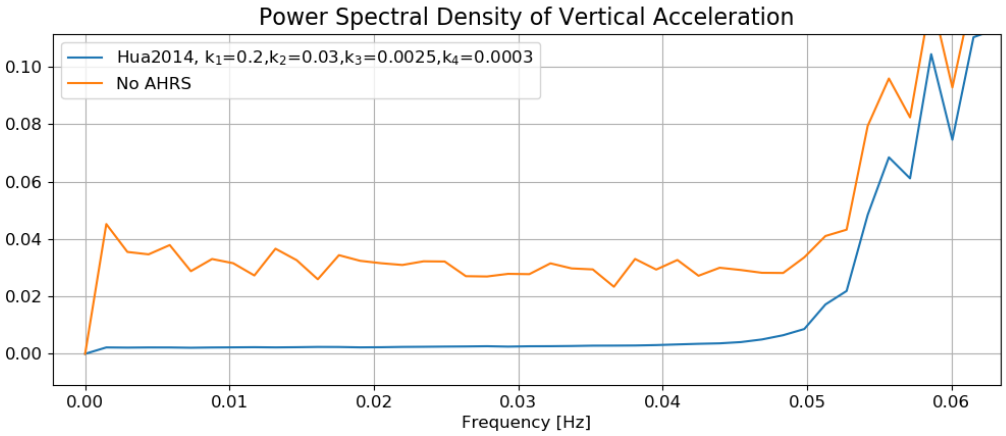
Figur 4.16.: Ufiltrert bølgespektrum over 24 timer

Figur 4.16 viser at når man ikke bruker noen AHRS, slik at akselerometer målingen brukes direkte, uten å rotere, så vil vi få enda mere støy for de lave frekvensene. Fra den lav-frekvente delen av figur 4.17 ser vi at denne støyen kommer fra en hvit-aktig støy, fordi energispekteret er forholdsvis flatt. Det er rimelig å anta at denne støyen kommer fra variasjon i målingen av den spesifikke normalkrafta som følge av akselerasjonen, siden denne krafta utgjør mesteparten av den målte akselerasjonen. Med AHRS vil gravitasjonen måles som en konstant som kan trekkes i fra, mens uten AHRS vil den varierende orienteringsfeilen gi et varierende avvik for denne kraften, som gir en hvit-aktig støy. Denne støyen er hovedårsaken til at man bør bruke tilte-kompensering når man estimerer bølger.

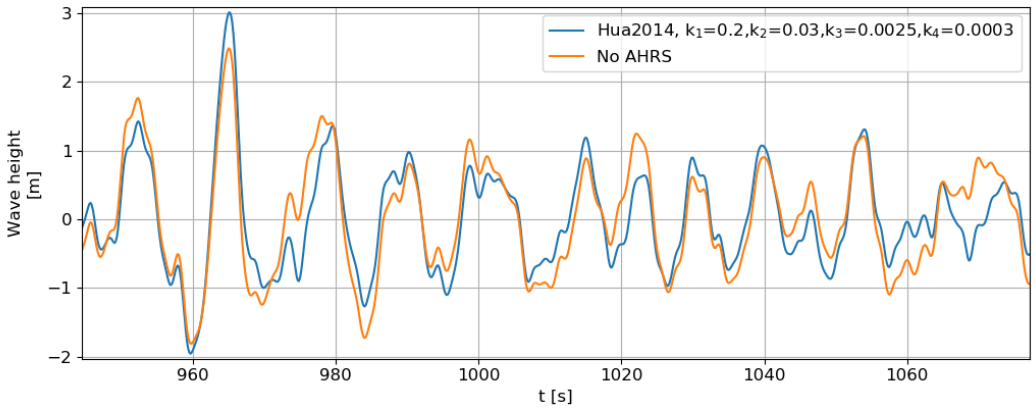
På den tiden da utviklingen av sensor- og datateknologi ikke hadde kommet så langt, var det aktuelt å estimere bølger uten bruk av tilte-kompensering, slik at man bare brukte et fastmontert akselerometer i body-ramma. I en artikkel av M. J. Tucker fra 1958 [30] ble feilen som da oppstår undersøkt, og slik som vi også har sett, ble det funnet at feil-signalet øker raskt ved langsommere frekvenser, som kan gjøre det vanskelig å måle langsomme bølger. Hoveddelen av bølgespekteret vil derimot ikke bli betydelig påvirket. Fra figur 4.16 ser at dette avviket begynner å bli betydelig når bølgeperioden øker over 16 sekunder. I teorien kan avviket for hver periode være større enn det figuren viser, fordi figuren viser gjennomsnittet over alle periodene for de 24 timene.

I figur 4.18 og 4.19 blir bølgehøyde, signifikant bølgehøyde og gjennomsnittlig bølgeperiode sammenlignet med og uten AHRS. Tabell 4.5 viser kvadratisk gjennomsnitt (RMS) av estimeringsavvikene med og uten AHRS. Prosentverdiene viser kvadratisk gjennomsnitt av prosentavvikene for hver periode/fil. For hver periode/fil er prosentavviket regnet ut i forhold til referanseobserveren, som er observeren til Hua et al. fra (2014) med justeringen $k_1 = 0.2$, $k_2 = 0.03$, $k_3 = 0.0025$ og $k_4 = 0.0003$. For bølgehøyden, rull, stamp

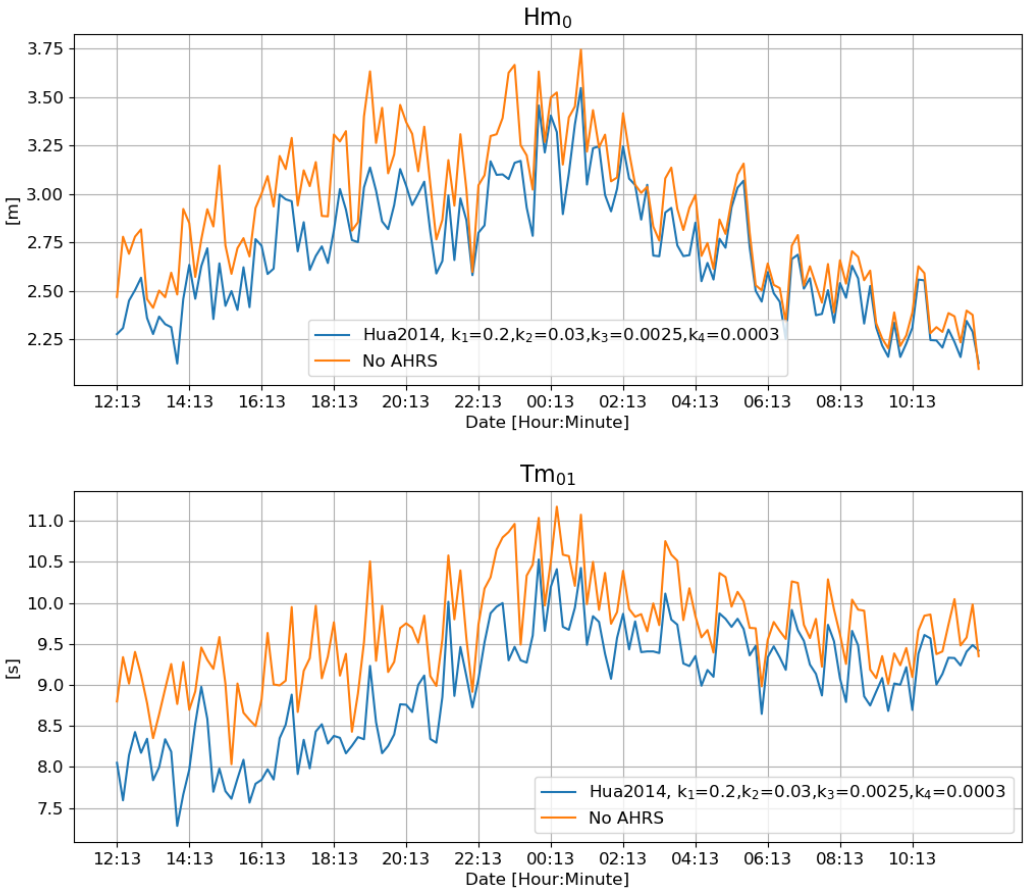
og gir i tidsdomenet betyr det at man for hver periode/fil tok kvadratisk gjennomsnitt (RMS) av estimeringsavvikene og delte på standardavviket for estimatene til referanse-observeren. Vi ser at for bølgehøyden er avviket 0.28 meter eller 41 %. Avviket for h_{m0} og t_{m01} er 8.3 % og 9 %. Cut-off frekvensen med og uten bruk av AHRS er begge satt til 0.04 Hz. Fra bølgespekteret ser vi at man kanskje burde satt cut-off frekvensen uten AHRS, til litt lavere, for eksempel 0.05 Hz. Da ville avvikene blitt mindre.



Figur 4.17.: Lav-frekvent energispektrum til vertikal-akselerasjonen over 24 timer



Figur 4.18.: Bølgehøyde/hiv



Figur 4.19.: Signifikant bølgehøyde h_{m0} og gjennomsnittlig bølgeperiode t_{m0}

	Bølgehøyde		H_{m0}		T_{m01}	
Ingen AHRS	0.28 m	41 %	0.23 m	8.3 %	0.77 s	9.1 %

Tabell 4.5.: Avvik fra referanse-AHRSen ved uten bruk av AHRS

4.2.3. Justering av AHRS

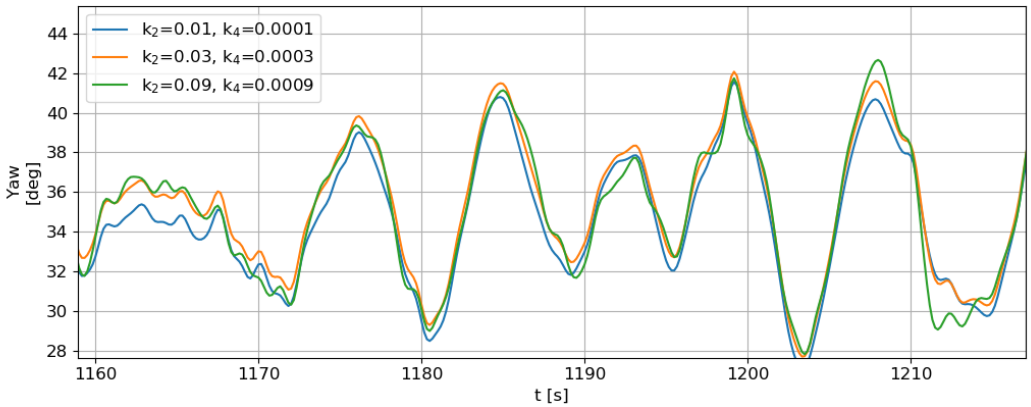
I artikkelen til Hua et al. [12] der denne AHRSen ble først beskrevet, står det også tips om hvordan den kan justeres. Proporsjonal-faktorene k_1 og k_2, settes for å få cut-off frekvensen som gir best balanse mellom lav-pass filtreringen av henholdsvis akselerometeret og magnetometeret, og høy-pass filtreringen av gyroen. I [12] blir AHRSen testet og justert for et flyvende fartøy og justeringsfaktorene som blir brukt er $k_1 = 1$, $k_2 = 0.2$, $k_3 = \frac{k_1}{32} = 0.031$ og $k_4 = \frac{k_3}{32} = 0.0063$. Der setter de k_2 til en femtedel av k_1 fordi magnetometeret har større usikkerhet enn akselerometeret. De setter også integral-faktorene k_3 og k_4 til 32 ganger mindre enn henholdsvis k_1 og k_2, og begrunner dette med at dynamikken til gyro-biasen er tregere enn dynamikken rull, stamp og gir. Justeringsfaktorene

som vi kom fram til i denne oppgaven er $k_1 = 0.2$, $k_2 = 0.03$, $k_3 = 0.0025$ og $k_4 = 0.0003$. Faktorene k_b og Δ brukes til integral-antiwindup-effekten, som det ikke ble bruk for i denne oppgaven, så disse to faktorene ble derfor satt til 0.

Vi ser at både k_1 og k_2 er vesentlig mindre for vår applikasjon. Kanskje skyldes dette noe så enkelt som at vår applikasjon har tregere dynamikk. Men det kan også være fordi vi bruker bedre gyroer. Hvis gyroene er gode nok, er det disse som gir riktig måleverdi, siden når observeren estimerer orienteringen med akselerometeret er det gravitasjonsretningen man ser på, men akselerometeret måler naturligvis annen akselerasjon i tillegg, noe som gir støy på estimatet. Derfor vil man i størst mulig grad bruke gyroer hvis disse er gode nok. Ellers ser vi at integral-faktorene vi bruker også er vesentlig lavere. Dette skyldes antageligvis også bedre gyrosensorer med mer stabil bias.

Justering av k_2 og k_4 for estimering av gir

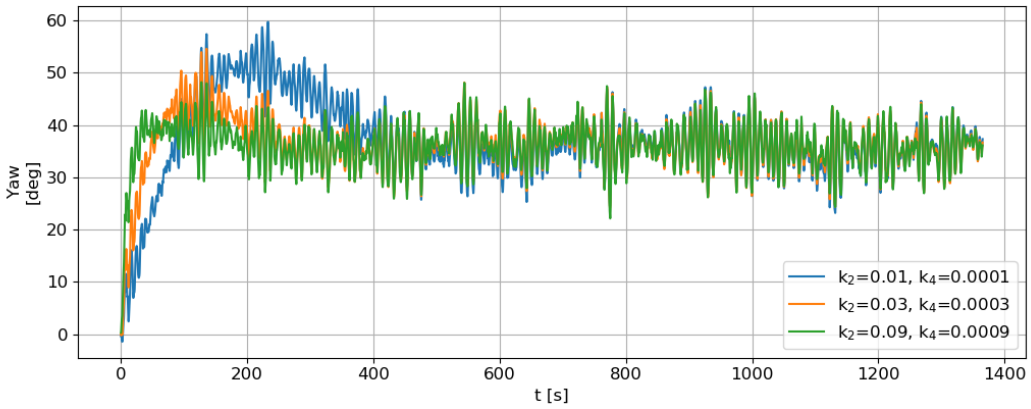
Ved justering av proporsjonal-faktoren k_2 for estimeringen av horisontal orientering / gir, kunne man observere at når k_2 ble for høy så fikk man en høyfrekvent komponent i estimatet, fra magnetometeret, mens hvis k_2 ble satt for lavt, så fikk estimatet en lavfrekvent drifting. I justeringsstrategien som ble brukt, ble disse to påvirkningene tolket som støy, og k_2 ble satt slik at man ikke fikk for mye av hverken den lav-frekvente eller høy-frekvente støyen. Det virket fornuftig å gradvis senke k_2 slik at støyen fra magnetometeret omtrent ble borte og man kunne se at den mer høyfrekvente formen til signalet begynte å bli meget lik det man får når man bare bruker gyro (altså når k_2 er lav). Verdiene som fungerte best var $k_2 = 0.03$, som vi ser av figur 4.20. Vi ser at $k_2 = 0.09$ blir for høyt, fordi man får høy-frekvente forstyrrelser i forhold til $k_2 = 0.03$ og $k_2 = 0.01$. $k_2 = 0.01$ blir igjen for lavt fordi man får et lav-frekvent avvik i forhold til $k_2 = 0.03$.



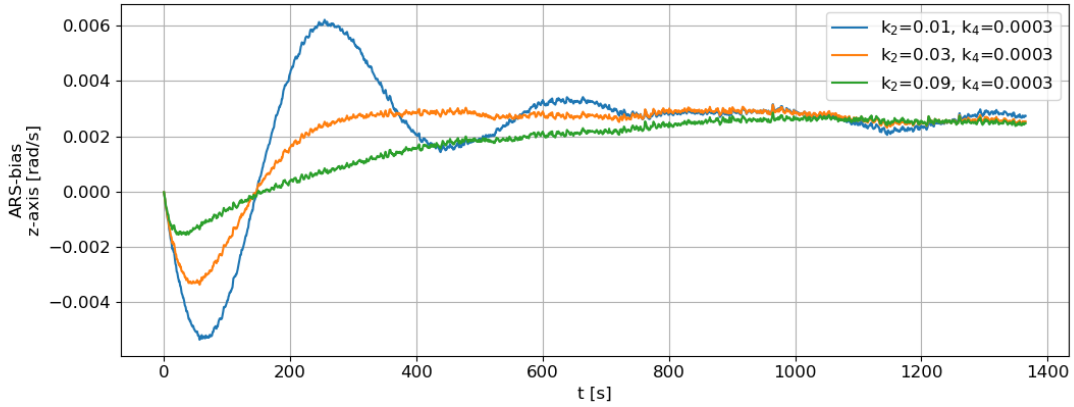
Figur 4.20.: Gir-vinkel for ulike verdier av proporsjonal-forsterkningen k_2

Fra figur 4.22 ser vi at det kan være fornuftig å bruke et konstant forhold mellom

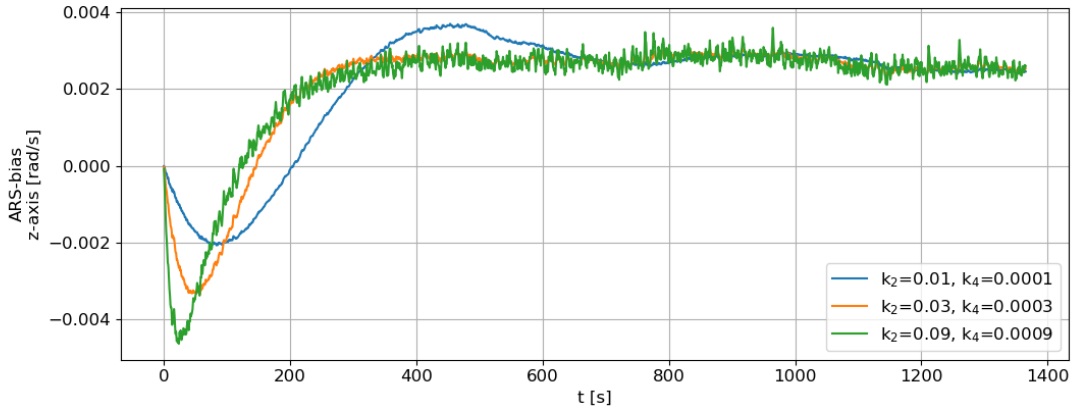
proporsjonal- og integral-faktoren når man justerer observeren. Dette for å unngå enten underdemping, eller for tregt innsving av bias. For høy integral-effekt gir underdemping. Figur 4.23 og 4.24 viser bias estimeringen for gyroens z-akse ved justering av k_2 med konstant forhold mellom k_2 og k_4. Vi ser at biasen er ganske stabil og at alle tre verdier for k_4 er tilstrekkelig for å estimere biasen. Hvis k_4 blir for høy, så får man støy på bias-estimatet, men ved videre undersøking ble det funnet at selv for den største verdien av k_4 så var denne støyen ikke av betydning for gir-estimatet. Av tabell 4.6, ser vi at valg av justeringsfaktorene k_2 og k_4 har i praksis ingenting å si for estimering av tiltingen og bølgehøyde. Dette viser at observeren har god dekobling. Når det gjelder gir-estimatet blir RMS av avvikene, på 0.7° og 1.0° ved valg av største eller minste justeringsverdi. Med tanke på at kompassretning målt med magnetometer ikke regnes for å være spesielt nøyaktig og kan gjerne ha avvik på flere grader, så er det antageligvis lite å hente på en mer nøyaktig justering. For mer nøyaktig justering, eller for verifisering, kan det være en mulighet til senere å sammenligne med målinger med GNSS med to antenner der målt retning er gitt av differansevektoren.



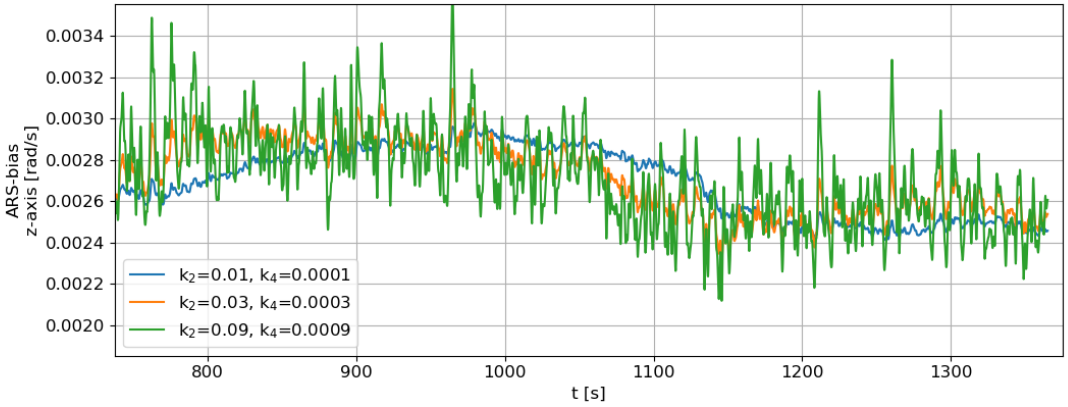
Figur 4.21.: Gir-vinkel for ulike verdier av proporsjonal-forsterkningen k_2



Figur 4.22.: Gyrobias i z-aksen for ulike verdier av proporsjonal-faktoren k_2, med konstant integral-faktor k_4



Figur 4.23.: Gyrobias i z-aksen for ulike verdier av proporsjonal-faktoren k_2



Figur 4.24.: Gyrobias i z-aksen for ulike verdier av proporsjonal-faktoren k_2

Justering	Rull		Stamp		Gir		Bølgehøyde	
k_2=0.01, k_4=0.0001	0.03°	0.6 %	0.04°	1.2 %	0.98°	34.4 %	0.0 cm	0.0 %
k_2=0.09, k_4=0.0009	0.08°	1.5 %	0.09°	2.9 %	0.69°	24.9 %	0.1 cm	0.1 %

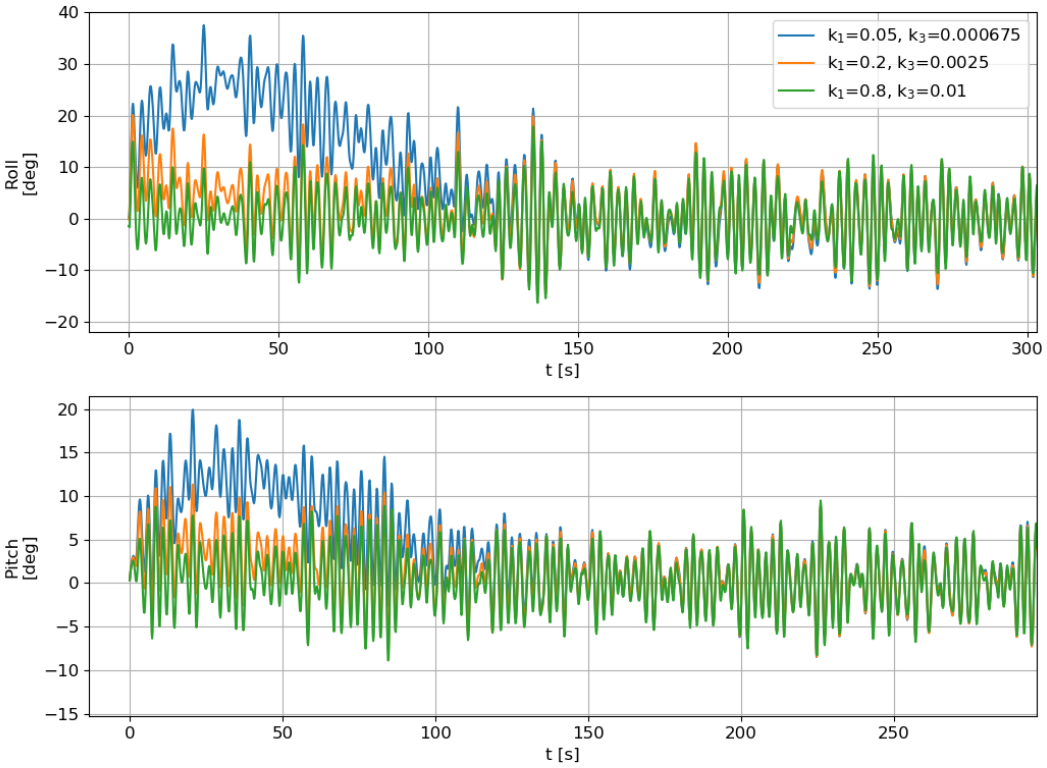
Tabell 4.6.: Avvik fra referansejusteringen, for ulike verdier av proporsjonal-faktoren k_2

Justering av k_1 og k_3 for estimering av rull og stamp

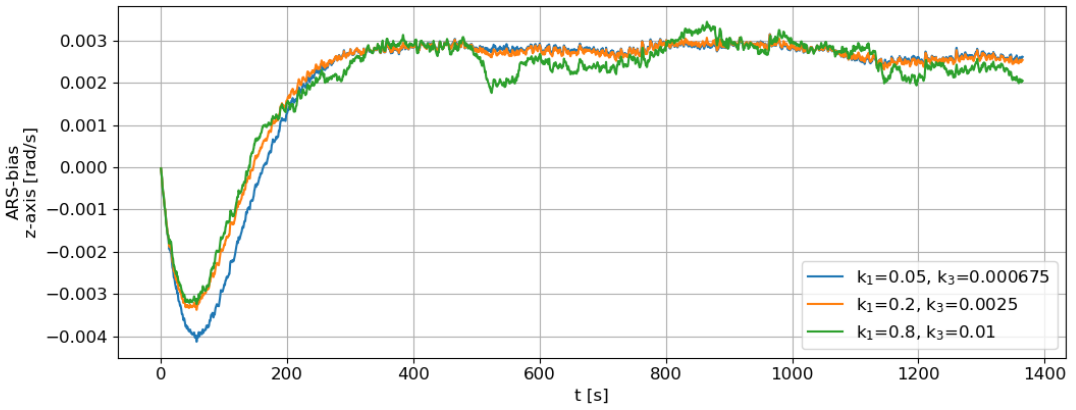
Prinsippene for justering av k_1 og k_3 er de samme som for k_2 og k_4 og derfor ble samme justeringsstrategi brukt her. Altså ble k_1 gradvis senket slik at støyen fra akselerometeret omtrent ble borte og man kunne se at den mer høyfrekvente formen til signalet begynte å bli meget lik det man får når man bare bruker gyro (altså når k_1 er lav). Vi ser av figur 4.27 at $k_1 = 0.2$ fungerte best i forhold til $k_1 = 0.05$ og $k_1 = 0.8$. $k_1 = 0.8$ blir for høyt fordi man får høy-frekvente forstyrrelser i forhold til $k_1 = 0.2$ og $k_1 = 0.05$. $k_1 = 0.05$ blir for lavt fordi man får et lavfrekvent avvik i forhold til $k_1 = 0.2$. Figur 4.25 viser initielt innsving av observeren, og tilsvarende som for justeringsfaktorene for gir-vinkel, så vil innsvinget ta lengre tid når k_1 blir lav. Figur 4.28 og 4.29 viser bias-estimeringen for gyroens x- og y-akse, mens figur 4.26 viser bias for gyroens z-akse. Det som står tidligere angående gyro-biasen ved estimering av gir, gjelder også her. Figur 4.31 viser bias-estimeringen for ulike valg av integral-faktoren k_3 og fra tabell 4.8 ser vi at hva man velger har forholdsvis lite å si for estimeringen. For eksempel når k_3 endres med en faktor på 5 vil RMS av avvikene for rull eller stamp bli 0.04-0.06°,mens hvis k_1 endres med en faktor på 4 vil RMS av avvikene bli 0.13-0.45°. Ellers kan man legge merke til at når k_1 settes til den største verdien ser det ut til at man får noe forstyrrelse på gyroens z-akse, og fra tabell 4.7 ser vi at man får mer avvik på gir. Avviket øker fra 0.18° til 0.66° når k_1 øker fra 0.4 til 0.8. 0.66° er ikke særlig mye, men økningen viser at dekoblingen mellom gir, og rull og stamp, ikke er fullstendig, og ved høye nok verdier for k_1, får man forstyrrelser i gir-estimatet.

Fra figur 4.30 ser vi at når $k_1 = 0.2$ og $k_3 = 0.0025$ så er de lav-frekvente komponentene til den vertikale akselerasjonen på et minimum. Det viste seg her også at det var k_1 og ikke k_3 som var den sensitive faktoren. Det gjennomsnittlige lav-frekvente vertikale akselerasjonsspekteret som figuren viser, var sensitivt nok for k_1 til at man kunne se små men tydelig økning hvis k_1 ble doblet til 0.4 eller halvert til 0.1. Fra tabell 4.7 ser vi at en slik endring gir et RMS avvik på rull eller stamp på mellom 0.13-0.45° og et RMS avvik på bølgehøyde på 0.5-0.8 cm, noe som kan sies å være ganske små avvik. Som tidligere forklart vil avvik i rull og stamp gi støy på den vertikale akselerasjonen, og siden det ikke skal være noe vertikal akselerasjon for de lavfrekvente komponentene vil dette gi økt støy i det lav-frekvente spekteret. Det virker derfor rimelig å anta at det vi ser i figur 4.30 er en økning i støy og at figuren gir en bekreftelse på at vi valgte riktig justering ved å studere rull- og stamp-estimatene i figur 4.20. Hvorvidt den beste justeringen ligger akkurat i minimum-punktet, og at $k_1 = 0.2$ er bedre enn $k_1 = 0.1$ eller 0.4 er kanskje ikke like sikkert. Men hvis det var tilfelle kan dette vise seg å være en enkel og effektiv måte å justere proporsjonal-faktoren k_1 på. For å verifisere om dette er en god metode og at $k_1 = 0.2$ er den beste verdien hadde det til senere vært interessant å sammenligne med målinger fra GNSS-RTK.

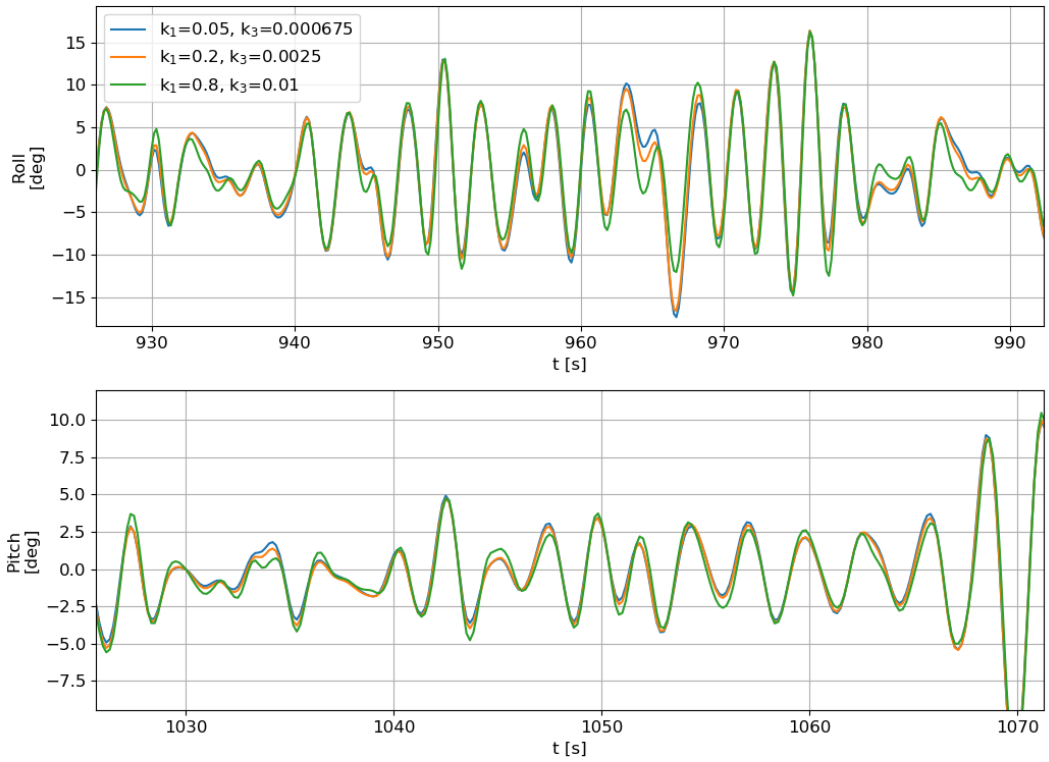
Når det gjelder avvik for bølgeparametrene h_{m0} og t_{m01} for ulike justeringer, så er disse ikke oppgitt i teksten, fordi de var så små at de kunne neglisjeres.



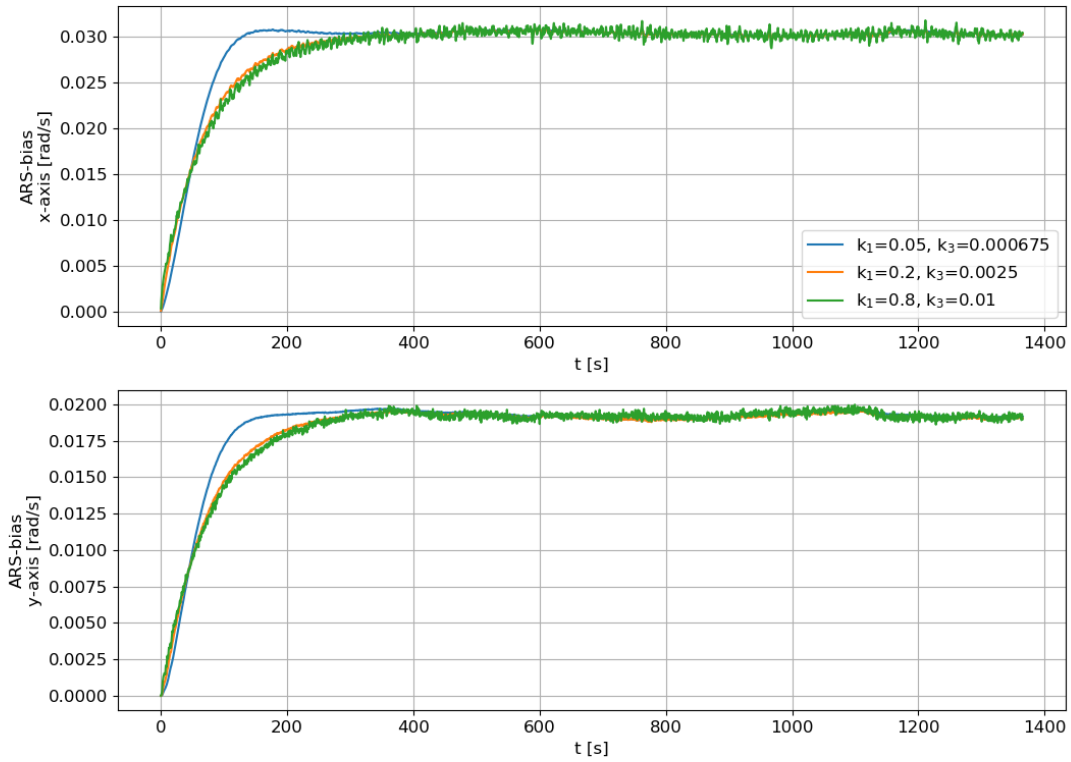
Figur 4.25.: Rull og stamp for ulike verdier av proporsjonal-faktoren k_1



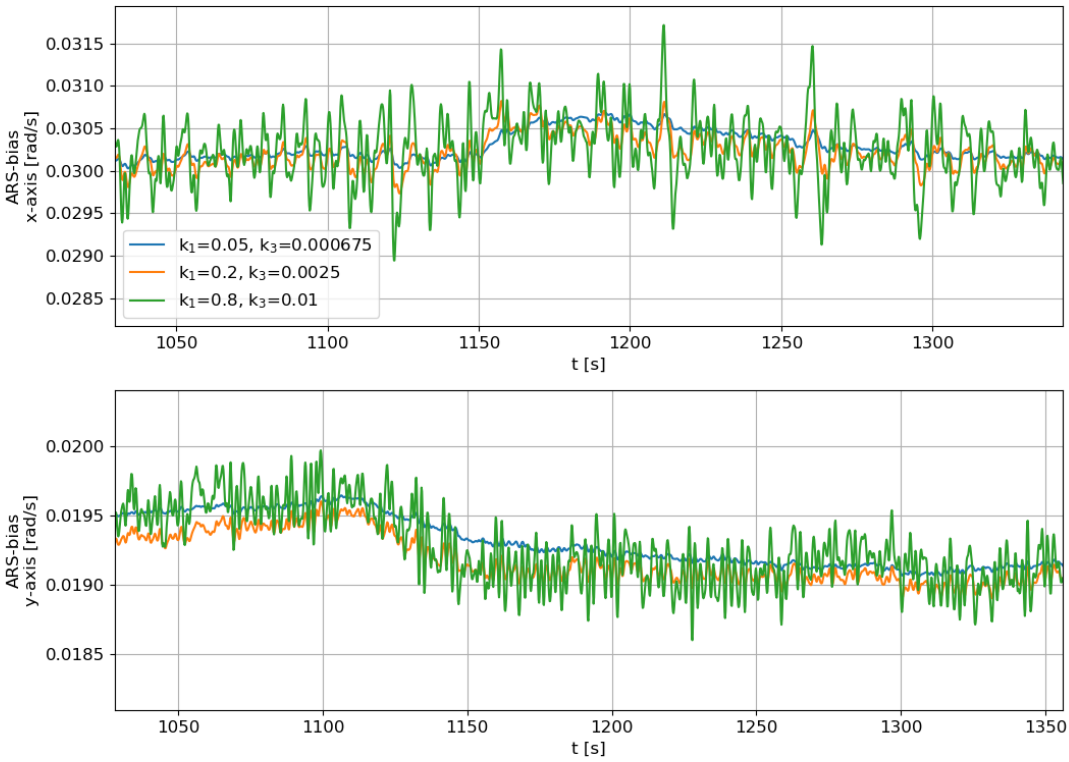
Figur 4.26.: Gyro-bias i z-aksen for ulike verdier av proporsjonal-faktoren k_1



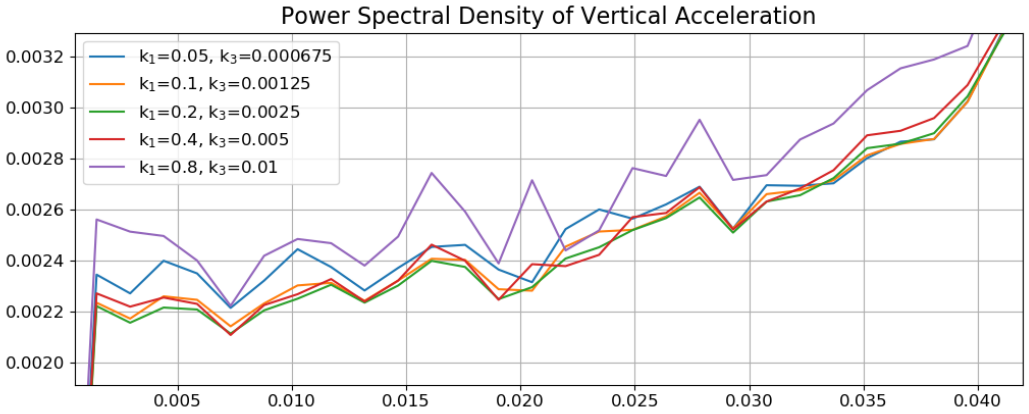
Figur 4.27.: Rull og stamp for ulike verdier av proporsjonal-faktoren k_1



Figur 4.28.: Gyrobias i x- og y-aksen for ulike verdier av proporsjonal-faktoren k_1



Figur 4.29.: Gyrobias i x- og y-aksen for ulike verdier av proporsjonal-faktoren k_1

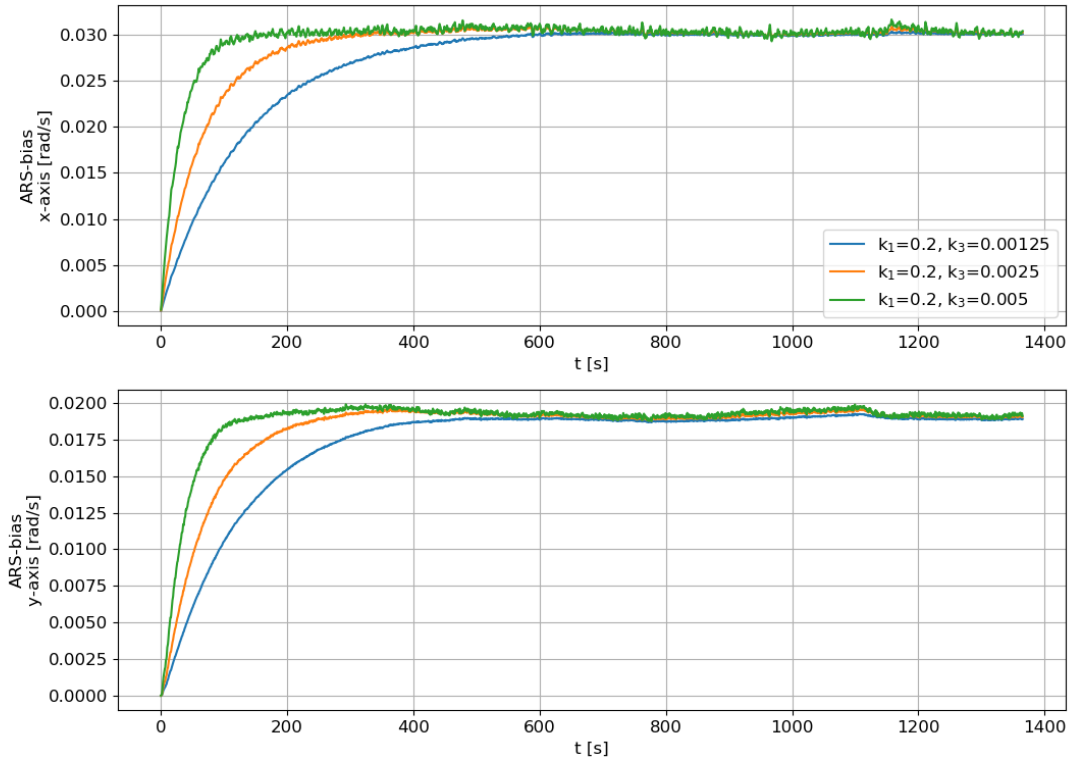


Figur 4.30.: Lav-frekvent energispektrum til vertikal-akselerasjonen over 24 timer, for ulike verdier av proporsjonal-faktoren k_1

Justering	Rull		Stamp		Gir		Bølgehøyde	
k_1=0.05, k_3=0.000675	0.46°	8.4 %	0.26°	8.0 %	0.11°	3.7 %	0.8 cm	1.2 %
k_1=0.1, k_3=0.00125	0.27°	5.0 %	0.13°	4.1 %	0.07°	2.6 %	0.5 cm	0.7 %
k_1=0.4, k_3=0.005	0.45°	8.4 %	0.20°	6.0 %	0.18°	6.2 %	0.8 cm	1.3 %
k_1=0.8, k_3=0.01	1.13°	20.9 %	0.53°	16.2 %	0.66°	22.6 %	2.5 cm	3.6 %

Tabell 4.7.: Avvik fra referanse-justeringen for ulike verdier av proporsjonal-faktoren k_1

Bias



Figur 4.31.: Gyrobias i x- og y-aksen for ulike verdier av integral-faktoren k_3

Justering	Rull		Stamp		Gir		Bølgehøyde	
k_1=0.2, k_3=0.00125	0.06°	1.2 %	0.05°	1.6 %	0.06°	2.1 %	0.1 cm	0.1 %
k_1=0.2, k_3=0.005	0.04°	0.7 %	0.04°	1.1 %	0.12°	4.3 %	0.0 cm	0.1 %

Tabell 4.8.: Avvik fra referanse-justeringen for ulike verdier av integral-faktoren k_3

4.2.4. Ikke-kausal estimering

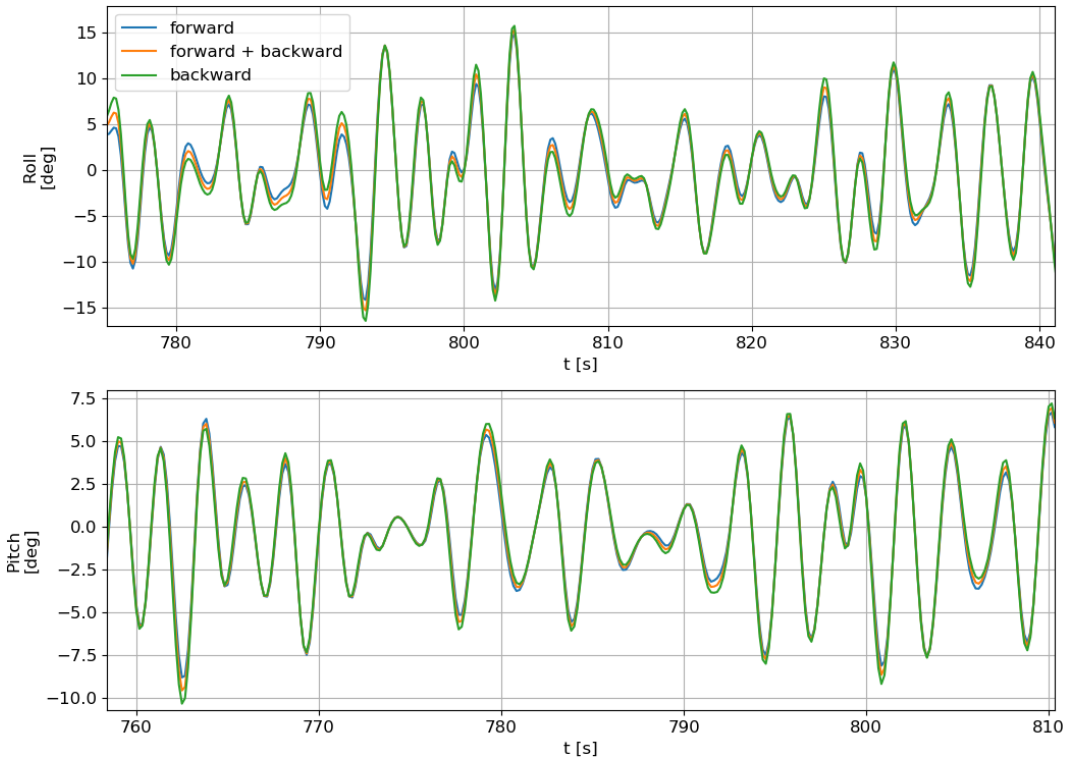
AHRSen vi har brukt i denne oppgaven er en kausal observer. Det betyr at estimatene er bare basert på tidligere målinger. En ikke-kausal observer bruker også fremtidige målinger for estimatene. Siden et ikke-kausalt observer derfor kan basere seg på mer informasjon, kan man potensielt få mer nøyaktige estimater. En enkel form for ikke-kausal observer, er å bruke en kausal observer til å regne seg bakover i tid, på målingene. For eksempel kan man da beregne to sett med estimater for måledata over en periode, der det ene settet er beregnet på vanlig måte, altså forover i tid, mens det andre er beregnet bakover i tid. Estimatene for begge tidsretninger burde da være like gode, og det virker derfor logisk at man får best estimater ved å ta gjennomsnittet av forover- og bakover-estimatene. En fordel med en slik observer er typisk at man eliminerer faseforskyvningen. For en målebøye kan en ikke-kausal observer være fornuftig fordi man ikke nødvendigvis trenger estimatene med en gang, og man kan derfor vente med beregningene til man har måledata for hele perioden. Det vil likevel være en ulempe at man må mellomlagre alle måledata fra perioden, noe man slipper når man bruker en vanlig rekursivt, ikke-kausal observer. En annen fordel med å teste ut estimering bakover i tid, er at det kan si noe om usikkerheten til estimatene. Hvis forskjellen mellom forover- og bakover-estimatene er stor, så må nødvendigvis usikkerheten ved estimatene også være stor, fordi begge estimatene skal være like nøyaktige. Vi har derfor testet denne formen for ikke-kausal filtrering i denne oppgaven.

Når man regner bakover med AHRSen må man huske å bytte fortegn på gyromålingene fordi hastigheten nå får motsatt retning. Man bør også huske å flytte gyromålingen én sample bakover i tid (mot venstre) i forhold til akselerometer og magnetometer-målingen. Dette fordi gyromålingen skal integreres og derfor får man motsatt forsinkelse når man regner bakover. Ved høy samplingsrate har ikke dette noe å si, mens siden samplingsraten her bare er 6 Hz vil det få litt å si. Når man har regnet seg forover for hele perioden, ble siste estimat brukt som initial-verdi for bakover-estimeringen, slik at observeren slipper å svinge seg inn på nytt. Dette ble gjort både for orienterings- og gyrobias-estimatet, men for gyrobias-estimatet må man huske på å skifte fortegn.

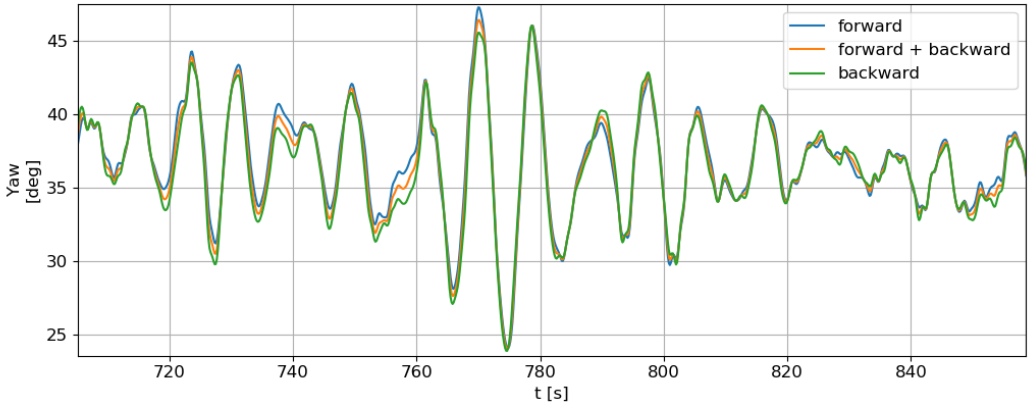
For å ta gjennomsnittet av orienteringsestimatet for forover- og bakoverestimeringen, ble formelen $q_{snitt} = \sqrt{q_{bakover} \otimes q_{forover}}$ brukt. For bruk av kvaternioner i python ble pakken *numpy-quaternion* brukt, og med denne kan man beregne kvadrat-rot av kvaternioner.

Fra figur 4.32 og 4.33 ser vi at når vi tar gjennomsnittet blir verdien for euler-vinklene liggende omtrent midt imellom forover- og bakoverestimatene. Fra tabell 4.9 ser vi at rull og gir estimatene forbedres med typisk en halv grad, og bølgehøyden med 0.5 cm. RMS avvik for bakoverestimeringen er på det dobbelte av dette. Disse avvikene kan tolkes som et minstemål på usikkerheten til observeren. Fra tabell 4.10 ser vi at for bølgeparametrene er avvikene rundt én promille, som er neglisjerbart. I figur 4.34 ser vi at det er mulig for middelverdien av orienteringsestimatet å gi det minste estimatet av bølgehøyden, noe som betyr at det å ta gjennomsnitt av orienteringsestimatene og det å ta gjennomsnitt av hiv-estimatene ikke gir det samme resultatet. Et annet litt merkverdig resultat var at for $k_1 < 0.2$ så fikk man mindre støy på det lav-frekvente vertikal-akselerasjonsspekteret,

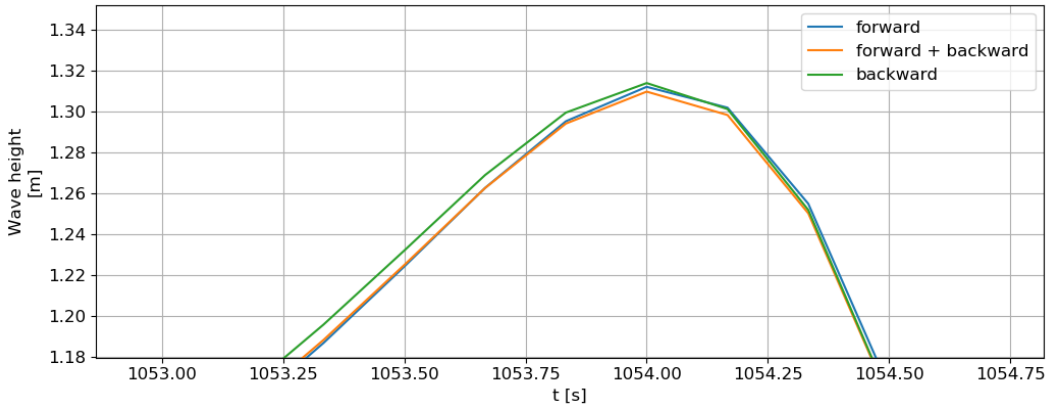
mens for $k_1 > 0.2$, fikk man mer støy, noe som illustreres av figur 4.35. Det er usikkert hvorfor, men det viser nok at det ikke alltid er den observeren med minst lavfrekvent vertikal-akselerasjonsspektrum, som gir de beste orienteringsestimatene.



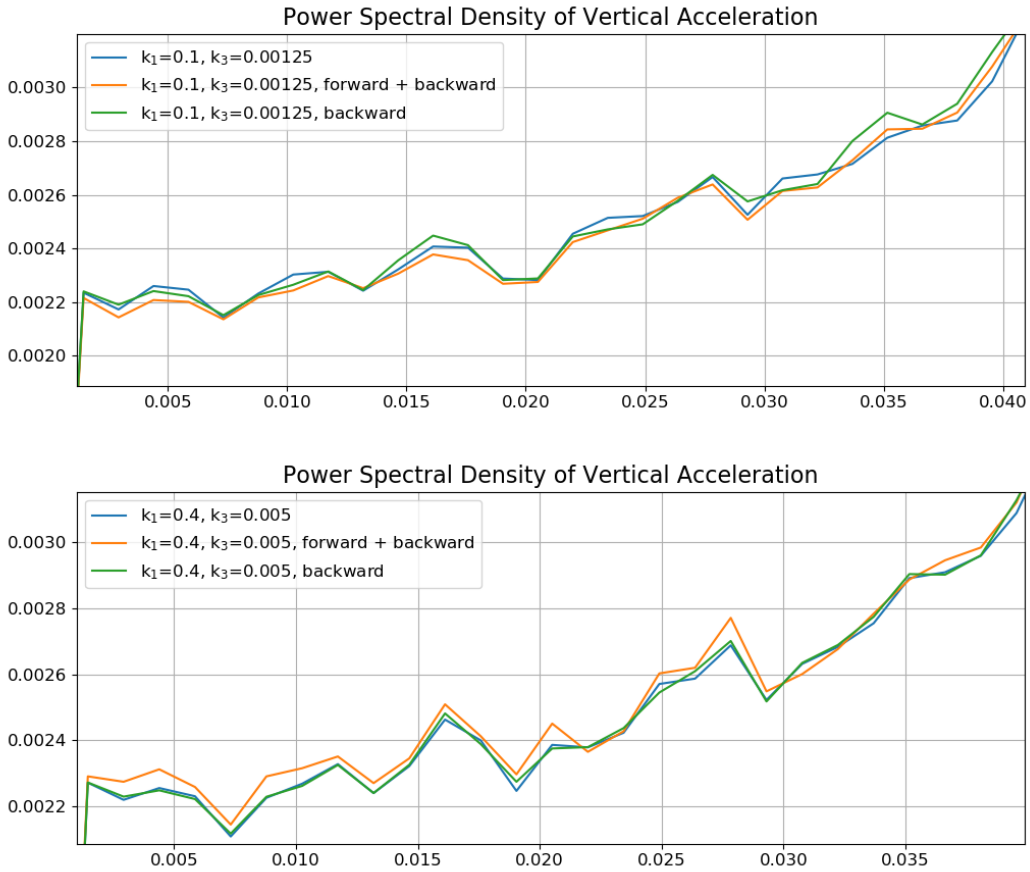
Figur 4.32.: Sammenligning av rull og stamp, med bakoverestimering



Figur 4.33.: Sammenligning av gir-vinkel, med bakoverestimering



Figur 4.34.: Sammenligning av bølgehøyde/hiv, med bakoverestimering



Figur 4.35.: Sammenligning av lav-frekvent energispektrum til vertikal-akselerasjonen over 24 timer, med bakoverestimering, for $k_1 = 0.1$ og $k_1 = 0.4$

AHRS type	Rull		Stamp		Gir		Bølgehøyde	
Forover + bakover	0.53°	9.9 %	0.22°	6.7 %	0.53°	18.8 %	0.6 cm	0.9 %
Bakover	1.07°	19.8 %	0.44°	13.4 %	1.06°	37.5 %	1.0 cm	1.5 %

Tabell 4.9.: Avvik fra referanseobserver, ved bruk av bakoverestimering

AHRS type	H_{m0}		T_{m01}	
Forover + bakover	0.3 cm	0.1 %	5 ms	0.1 %
Bakover	0.5 cm	0.2 %	9 ms	0.1 %

Tabell 4.10.: Avvik i bølgeparametrene fra referanseobserver, ved bruk av bakoverestimering

4.2.5. Akselerometer-bias

AHRSen vi har brukt i denne oppgaven, beregner og kompenserer for gyro-biasen, men den tar ikke hensyn til akselerometerbiasen. Fra figur 3.2 ser vi at IMUen Ellipse 2 Micro har oppgitt en bias-stabilitet på ±5 mg. I databladet er denne verdien oppgitt med merknaden «After one year of accelerated aging». Altså må man kanskje forvente en bias noen hundredels g etter noen år, hvis man ikke rekalibrerer. Dette er for en forholdsvis dyr IMU, mens for billige og strømgjerrige IMUer som er lite kalibrert og sensitiv for temperaturendringer, så vil man kunne forvente vesentlig mer bias enn dette. Det er derfor interessant å undersøke hvordan akselerometerbias påvirker estimatene. For å gjøre dette, ble det lagt til ulike biaser på rådata for å beregne avvikene som oppstod.

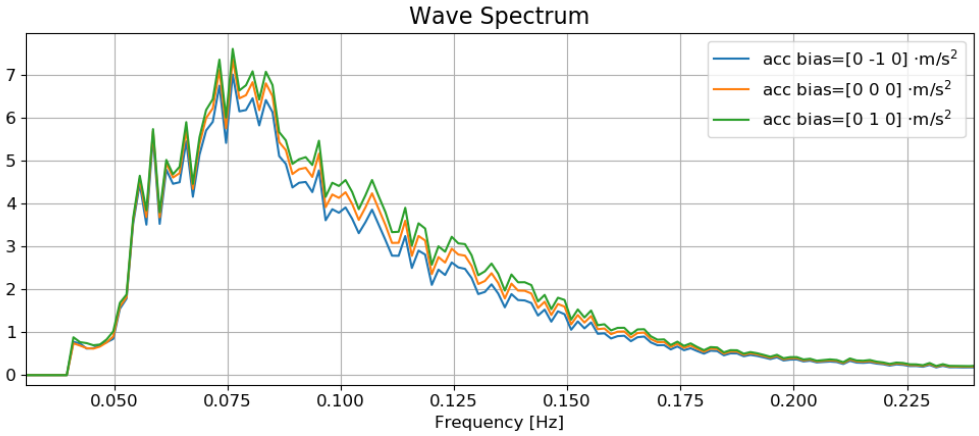
Figur 4.37, 4.38, 4.39 og tabell 4.11 viser hvordan rull, stamp, gir og bølgehøyde påvirkes, for ulike biaser på y-aksen til akselerometeret. Vi ser at ved bias på y-aksen får vi et konstant avvik på rull fremfor stamp, mens ved bias på x-aksen får man et konstant avvik på stamp fremfor rull, slik vi ser av tabell B.2. Det er fordi AHRSen vil tolke en bias på y-aksen eller x-aksen som en rotasjon rundt henholdsvis x-aksen eller y-aksen. Det er interessant å merke seg at avviket for gir / horisontalretningen, blir enda større enn for rull eller stamp. Det er fordi når man får en bias som gjør at akselerometer-vektoren (som peker oppover) begynner å tilte mot øst/vest, så vil AHRSen tolke det som at bøya tilter mot vest/øst, som vil gjøre at magnetometer-vektoren (som peker nedover mot nord) begynner å peke mer mot øst/vest. Dette er avhengig av at magnetometeret peker nedover, noe det gjør i større grad jo lengre nord man kommer (forutsatt at bøya befinner seg nord for ekvator). Vi ser at når akselerometer-biasen er på bare 0.15 m/s^2 for x-aksen eller y-aksen, så blir avviket for horisontal-retningen lik henholdsvis 1.7° og 2.8°. 0.15 m/s^2 er 3 ganger bias-instabiliteten som SBG-sensoren er oppgitt med, så det skal altså ikke spesielt mye til før man får avvik på gir, når man er så langt nord som 62 grader.

Fra tabell 4.12 ser vi at ved akselerometerbias på y-aksen så får man en del avvik på bølgeparametrene. For eksempel med bias på 1 m/s^2 blir avvik for h_{m0} 8.5 cm, som utgjør 3.1 prosent. Fra figur 4.36 ser vi hvordan bølgespekteret påvirkes av akselerometerbias på y-aksen på ±1 m/s^2. Vi ser at støyen for det meste ligger mellom 0.075 Hz og 0.15 Hz, altså vår vi mer høyfrekvent støy sammenlignet med det vi fikk når vi ikke brukte AHRS (se figur 4.16). Uten AHRS ville man bare brukt z-akselerasjonen i body-ramma, og man ville fått bølgespekteret i figur 4.16. Altså blir støyen verre for de høyere frekvenskomponentene, med bruk av AHRSen enn uten, når man har akselerometerbias på y- eller x-aksen.

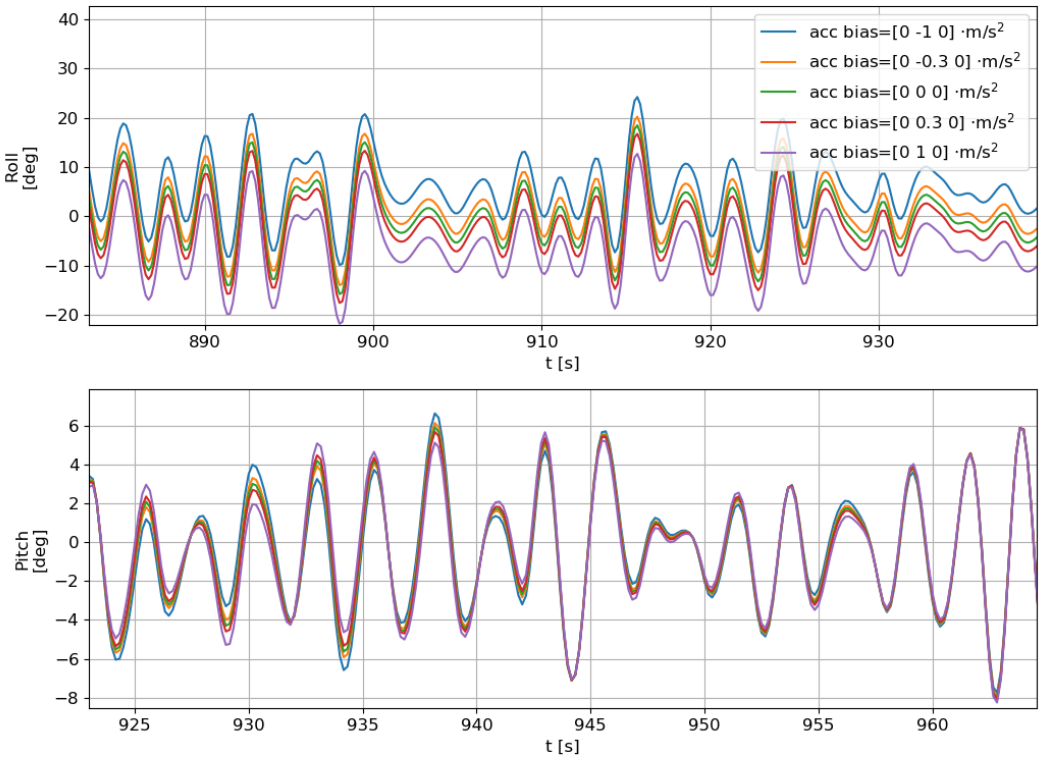
I tabell B.1 ser vi avvikene i bølgeparametre for akselerometerbias på x-aksen ikke blir så store. Her er avvik for h_{m0} på 1.6 cm ved bias på x-aksen på 1 m/s^2. Muligens er dette fordi bøya har mer variasjon i rull, enn for stamp, slik at man får mer avvik på bølgeparametrene hvis man har bias i samme retning som bøya ruller mot.

Figur 4.40, og tabell B.3 og B.4 viser avvikene man får ved akselerometerbias på z-aksen. Vi ser at avvikene er ganske like for positive og negative biaser. Bias på z-aksen gir lite

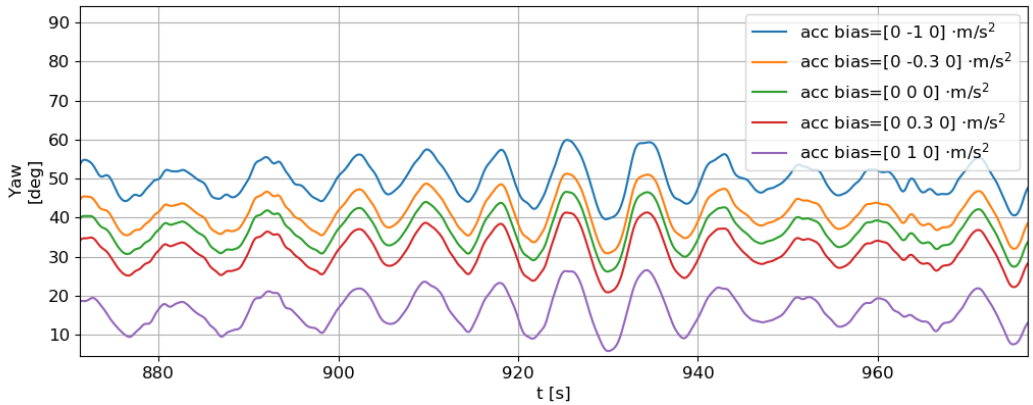
avvik på rull, stamp og gir. Vi ser også at vi får noe avvik på bølgehøyde, for eksempel gir bias på 1 m/s^2 et avvik på 2.4 cm, noe som ligger nært avviket på 2.9 cm for samme bias på x-aksen. For bølgeparametrene derimot, så er avvikene mindre. For bias på 1 m/s^2 på z-, x- eller y-aksen er avvikene for h_{m0} på henholdsvis 0.6 cm, 1.6 cm og 8.5 cm.



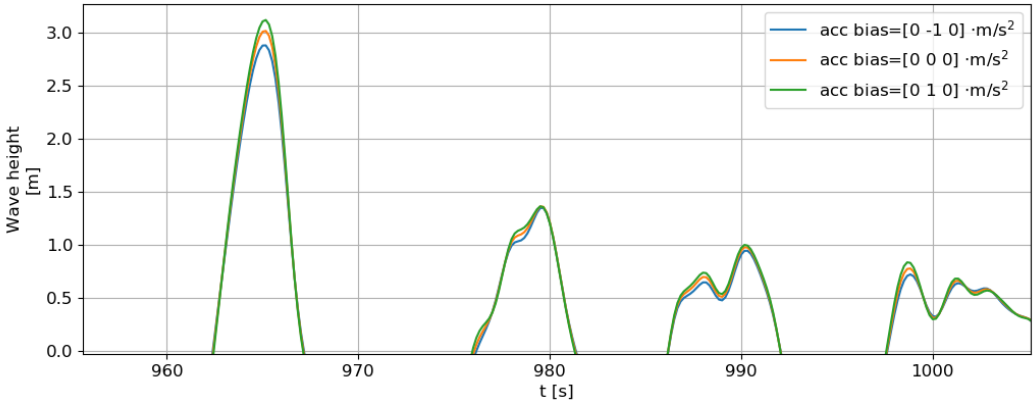
Figur 4.36.: Bølgespektrum over 24-timer, for ulike akselerometerbiaser på y-aksen



Figur 4.37.: Rull og stamp, for ulike akselerometerbiaser på y-aksen



Figur 4.38.: Gir-vinkel, for ulike akselerometerbiaser på y-aksen



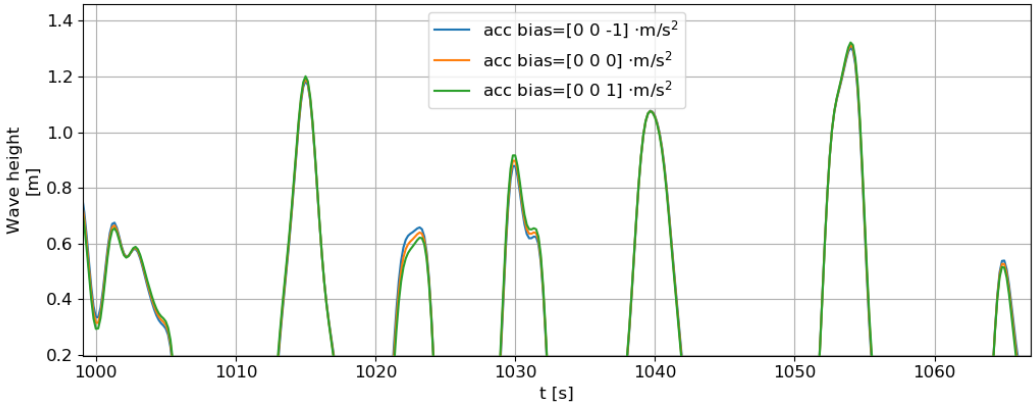
Figur 4.39.: Bølgehøyde/hiv, for ulike akselerometerbiaser på y-aksen

Akselerometerbias	Rull		Stamp		Gir		Bølgehøyde	
acc y-bias = 0.05 m/s^2	0.29°	5.6 %	0.02°	0.5 %	0.91°	35.3 %	0.2 cm	0.3 %
acc y-bias = 0.15 m/s^2	0.88°	16.9 %	0.05°	1.5 %	2.78°	108.0 %	0.7 cm	1.0 %
acc y-bias = 0.3 m/s^2	1.76°	33.7 %	0.10°	3.0 %	5.73°	222.4 %	1.3 cm	2.0 %
acc y-bias = 1 m/s^2	5.86°	112.3 %	0.33°	10.0 %	21.5°	829.9 %	4.4 cm	6.5 %
acc y-bias = 2 m/s^2	11.6°	222.3 %	0.65°	19.8 %	46.4°	1771.5 %	8.6 cm	12.7 %

Tabell 4.11.: Avvik i orientering og bølgehøyde, for ulike akselerometerbiaser på y-aksen

Akselerometerbias	H_{m0}		T_{m01}	
acc y-bias = 0.05 m/s^2	0.5 cm	0.2 %	0.006 s	0.1 %
acc y-bias = 0.15 m/s^2	1.4 cm	0.5 %	0.019 s	0.2 %
acc y-bias = 0.3 m/s^2	2.7 cm	1.0 %	0.038 s	0.4 %
acc y-bias = 1 m/s^2	8.5 cm	3.1 %	0.12 s	1.3 %
acc y-bias = 2 m/s^2	15 cm	5.5 %	0.23 s	2.5 %

Tabell 4.12.: Avvik i bølgeparametrene for ulike akselerometerbiaser på y-aksen



Figur 4.40.: Bølgehøyde/hiv for ulike akselerometerbiaser på y-aksen

4.2.6. Bias estimering

Vi har sett at akselerometerbias kan gi vesentlig avvik i rull og stamp, og enda større avvik i gir. Ved store biaser får man også en del avvik i hiv/bølgehøyden og man kan få en del avvik i bølgeparametrene. Hvis man bruker en IMU med god kalibrering av akselerometerene og høy grad av bias-stabilitet, så vil man få gode nok estimater. Men hvis det er lang tid siden akselerometerene er blitt kalibrert, eller man bruker en billig IMU med lite kalibrering og med sensitivitet for temperaturendring, så vil akselerometer-biasen kunne bli et problem. En mulighet kan da være å estimere akselerometerbiasen. Posisjons-observeren fra seksjon 2.3.1 kan estimere akselerometerbias, og kanskje kan det være en mulighet å bruke denne. Men det kan være et problem at denne observeren antar et korrekt orienteringsestimat. Siden observeren må bruke orienteringsestimatet fra AHRSen, som også er påvirket av akselerometerbiasen, så er det lett for at bias-estimatet ikke vil konvergerer. AHRSen og posisjonsobserveren påvirker hverandre gjensidig, og det kan tenkes at hvis man har nok variasjon/eksitasjon i orienteringstilstanden, så vil biasen konvergere, men siden en målebøye har relativt små variasjoner, virker dette også tvilsomt.

En mulig måte å gjøre AHRSen uavhengig av akselerometerbias på kan være bruke en antagelse om at kraften på bøya, i gjennomsnitt vil peke vertikalt. Om denne antagelsen stemmer vil være avhengig av både rotasjons- og translasjonsbevegelsene til bøya. Man kan tenke seg at antagelsen kan være oppfylt fordi bevegelsen til bøya er ganske symmetriske og at bøya i gjennomsnitt svinger rundt vertikal-linja. Dermed kan man modifisere AHRSen slik at orienteringsestimatet i gjennomsnitt svinger rundt vertikal-linja. En måte å gjøre dette på for AHRSen til Hua et al. er slik:

$$
\begin{aligned}
\dot{\hat{\boldsymbol{\tau}}} &= \begin{bmatrix} 2(q_j q_k + q_i q_r) & -2(q_i q_k - q_j q_r) & 0 \end{bmatrix}^T - k_{\tau 2}\hat{\boldsymbol{\tau}} \\
\boldsymbol{\epsilon}^b_{\text{mes}} &= k_1 \boldsymbol{u}^b \times \hat{\boldsymbol{u}}^b + k_2 \hat{\boldsymbol{u}}^b (\hat{\boldsymbol{u}}^b)^T \boldsymbol{v}^b \times \hat{\boldsymbol{v}}^b - k_{\tau 1}\hat{\boldsymbol{\tau}} \\
\dot{\hat{\boldsymbol{b}}}^b_{\text{gyro}} &= -k_b\Big(\hat{\boldsymbol{b}}^b_{\text{gyro}} - \text{sat}_\Delta(\hat{\boldsymbol{b}}^b_{\text{gyro}})\Big) - \frac{k_3}{k_1 + k_{\tau 1}}(k_1 \boldsymbol{u}^b \times \hat{\boldsymbol{u}}^b - k_{\tau 1}\hat{\boldsymbol{\tau}}) - k_4 \boldsymbol{v}^b \times \hat{\boldsymbol{v}}^b \\
\dot{\hat{\boldsymbol{q}}} &= \boldsymbol{T}_q(\hat{\boldsymbol{q}})\Big(\boldsymbol{\omega}^b_{\text{imu}} - \hat{\boldsymbol{b}}^b_{\text{gyro}} + \boldsymbol{\epsilon}^b_{\text{mes}}\Big)
\end{aligned}
\tag{4.1}
$$

der $k_{\tau 2} > 0$ hvis $k_1 = 0$. Hvis $k_1 > 0$, kan man sette $k_{\tau 2} = 0$. Hvis man setter $k_{\tau 1} = k_{\tau 2} = 0$ blir observeren lik observeren til Hua et al. Uttrykket for integral-effekten til tiltingen kan være enklere å forstå hvis man bruker euler-vinkler[1]:

$$
\dot{\hat{\boldsymbol{\tau}}} = \Big[\sin\hat{\phi}\cos\hat{\theta} \quad \sin\hat{\theta} \quad 0\Big]^T - k_{\tau 2}\hat{\boldsymbol{\tau}} \tag{4.2}
$$

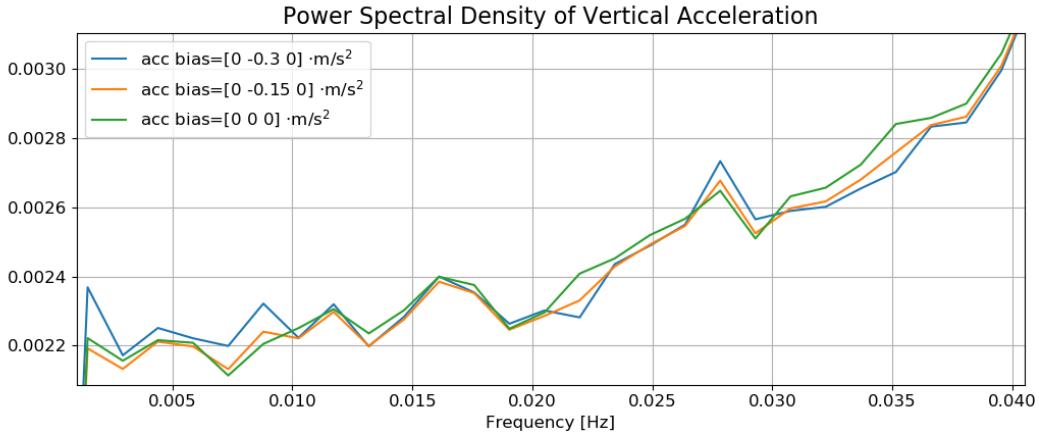
Denne AHRSen har en integral-effekt på rull og stamp, gitt av ligning 4.2, som gjør at gjennomsnittet til $\sin\hat{\phi}\cos\hat{\theta}$, som representerer rull, og $\sin\hat{\theta}$ som representerer stamp, vil over tid gå mot 0.

Antagelsen om at bøya i snitt svinger rundt vertikalen er neppe helt nøyaktig. For eksempel så kan det for visse bøyer og avhengig av værforhold være slik at fortøyningen til bøya, trekker i bøya slik at den tilter. Men hvis biasen til akselerometerene er stor, så kan kanskje en slik AHRS som dette være hensiktsmessig. Da kan AHRSen gi estimater av rull, stamp og gir, uavhengig av akselerometerbiasen, og det vil derfor fungere bedre å bruke orienteringsestimatet fra AHRSen i posisjonsobserveren, som igjen kan estimere akselerometerbiasen.

Med den samme antagelsen finnes det også andre måter å estimere biasen på. Man kan for eksempel se på målevektoren til akselerometeret når det er helt rolig sjø. Siden målevektoren skal tilsvare g-vektoren, kan man estimere biasen som differansen til g-vektoren. En annen mulighet er å se på gjennomsnittlig akselerometermåling for x- og y-aksen (i body-ramma). For rådata vi brukte i denne oppgaven så ser vi disse fra figur 4.42. Hvis våre antagelser gjelder, så bør disse i gjennomsnitt tilsvare biasen for x- og y-aksen. Isåfall har y-akselerometeret en bias på cirka 0.18 m/s^2, mens x-biasen er cirka lik -0.01 m/s^2 (Samtidig kan man kanskje estimere biasen til z-aksen ved å anta at den vertikale akselerasjonen (i NED-ramma) i snitt skal tilsvare gravitasjonskonstanten). Vi ser at det er noe variasjon av figuren, og det ser ut til at ihvertfall noe av denne skyldes annen støy enn akselerometerbias. Ved å se på vertikal-akselerasjonsspekteret for ulike akselerometerbiaser, kunne man se at bias ga mer støy. Fra figur 4.41 ser vi at når man legger til en bias på 0, -0.15 og -0.3 m/s^2, så får man minst støy med -0.15 m/s^2. Ved å undersøke dette spekteret enda mer nøye for ulike biaser, så det ut til at man fikk et

[1]For å konvertere $2(q_j q_k + q_i q_r)$ og $-2(q_i q_k - q_j q_r)$ til euler-vinkler kan man sammenligne rotasjonsmatrisene for enhets-kvaternioner og euler-vinkler.

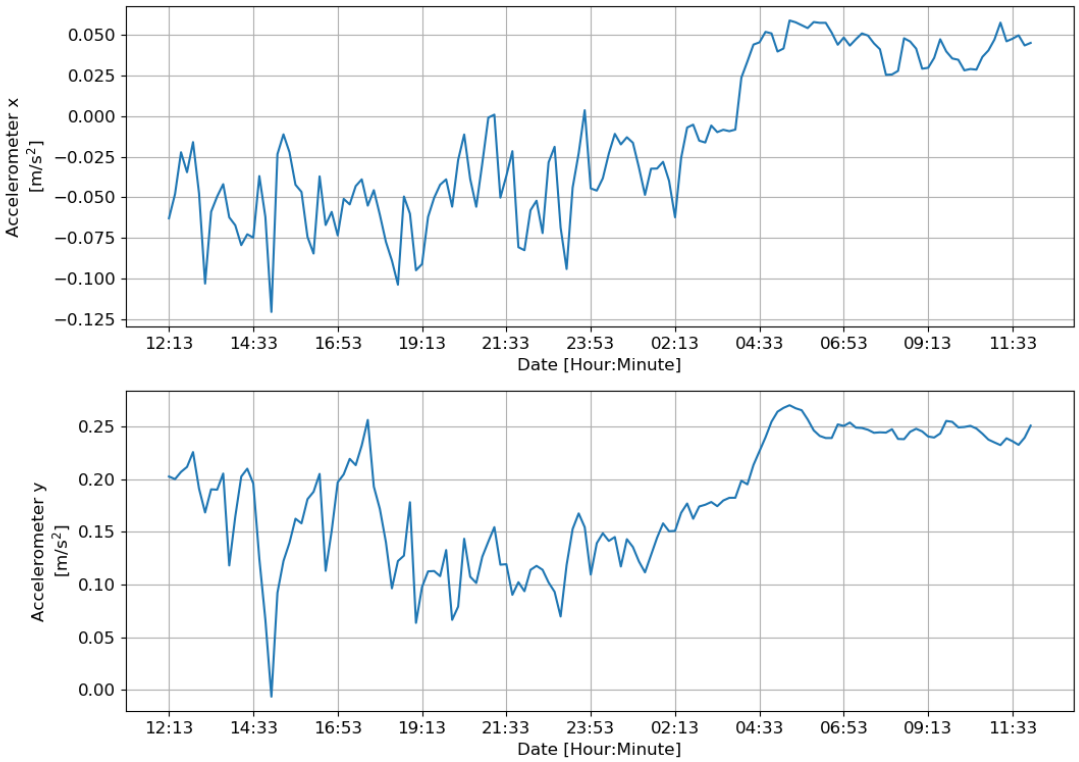
minimum med støy for cirka -0.16 m/s^2 på y-aksen, og -0.05 på x-aksen, som altså gir en bias på 0.16 m/s^2 og 0.05 m/s^2. (dette var ved å undersøke bias for x- og y-aksen hver for seg, noe som kanskje gir rom for forbedring). Fordelen med å undersøke akselerometerbiasen på denne måten er at hvis man har en rotasjonsfeil på bøya som følge av for eksempel fortøyningen, så bør ikke dette ha noe å si for vertikal-akselerasjonsspekteret, siden dette spekteret skal være det samme uavhengig av hvordan bøya er rotert. Hvis biasen for x- og y-aksen faktisk er 0.16 m/s^2 og 0.05 m/s^2 i gjennomsnitt, og siden man utifra antagelsen om at akselerasjonsmålingen i snitt peker vertikalt, kan estimere biasen til i gjennomsnitt 0.18 m/s^2 og -0.01 m/s^2, så er dette en forbedring. Som vi har sett tidligere fra tabell 4.11 så vil en forbedring på 0.15 m/s^2 på biasen for y-aksen, for våre rådata gi en forbedring i gir-vinkelen på cirka 3°. Dette tyder på at selv for våre rådata, der akselerometerbiasen er ganske lav, og man har store bølger, så kan man ved bruk av vår antagelse, få signifikante forbedringer i estimatene. Sett at man brukte en IMU som av ulike årsaker ikke var så god, så kan dette vise at metodene som er beskrevet for estimering av akselerometerbias, har potensial.



Figur 4.41.: Lav-frekvent energispektrum til vertikal-akselerasjonen over 24 timer, for ulike akselerometer-biaser på y-aksen

Som vi ser av spesifikasjonene til SBG sensoren i figur 3.2, så har en IMU flere støykilder enn gyro- og akselerometerbias. For eksempel ser vi at både gyro og akselerometer kan ha et forsterkningsavvik, altså et avvik som er proporsjonal med målingen. Vi ser at akselerometeret og gyro har et forsterkningsstabilitet på henholdsvis 1 promille og 0.5 promille, med merknaden «After one year of accelerated aging» i manualen. Forsterkningsavvikene vil antageligvis kunne være forskjellig langs de ulike aksene. Hvis man for enkelhets skyld antar at forsterkningsavviket er like stort langs alle tre akser, så vil for eksempel 1 prosent positivt avvik på forsterkningen vil bety at den målte akselerasjonen uten bias blir 1.01 ganger større. Det virker logisk at bølgehøyden og signifikant bølgehøyde da vil øke med 1 prosent, men at orienteringen vil være uendret (Det virker rimelig at det samme også gjelder for orienteringsavvik i forhold til forsterkningsavvik for gyroen). En forsterkningsstabilitet på en promille, slik som SBG-sensoren har vil

nok være tilstrekkelig for vår applikasjon. Tilsammenligning vil nok bias-stabiliteten på ±5 mg ha større betydning for estimeringen enn dette. Men om man skulle bruke en IMU som har vesentlig dårligere forsterkningsstabilitet, fordi den er billig eller av andre årsaker, så kan det lett tenkes at det blir et problem. Det virker nesten umulig å kunne estimere forsterkningsavviket for akselerometeret, fordi man ikke greier å skille mellom bias og forsterkningsavvik når en målebøye har så små vinkelutslag.



Figur 4.42.: Gjennomsnitt av akselerometermåling i x- og y-aksen

5. Diskusjon

Det ble i denne oppgaven designet og implementert programvare og algoritmer for en kombinert bølgesensor og sanntids INS på mikrokontroller for bruk på en målebøye. Programvaren henter måledata fra en IMU fra SBG med akselerometer, gyro og magnetometer. For bølgeanalyse beregnes bølge- og retningsspekteret, signifikant bølgehøyde h_{m0} og gjennomsnittlig bølgeperiode t_{m01}. Bølgeanalysen beregnes for en periode på 17 minutter, med en samplingfrekvens på 2 Hz, slik at man kan måle bølgefrekvenser på opptil 1 Hz. En enkel INS som estimerer av orientering og posisjon/hiv i sanntid er implementert i to deler, en AHRS for orientering, og et kalmanfilter for posisjon/hiv. Den samme AHRSen brukes for å beregne rådata til bølgeanalysen. INSen beregner sanntidsdata på 20 Hz, på 11 ms (etter at data er mottatt fra IMU). Sanntidsdata og rådata fra IMU, sendes over seriell port til datalogger. Data fra bølgeanalysen, inkludert rådata, sendes over den samme seriell-porten, ved spørring fra datalogger.

Til oppgaven ble det valgt en 32-bits mikrokontroller fra STMicroelectronics av typen STM32L4+, som kan kjøre på en klokkefrevens på opptil 120 MHz, og har 2 MB flashminne og 640 KB SRAM. Minneforbruk for SRAM og flash var på henholdsvis 29 % og 9 %. Hvis man økte perioden for bølgeanalyse til 4096 (34 minutter) så økte minneforbruk av SRAM til 51 %. Mikrokontrolleren ble klokket til 4 MHz, og ved beregning, lagring og overføring av data ble enkel-presisjons flyttall brukt. Mikrokontroller var i sove-modus 82 % av tiden. Mesteparten av beregningstid gikk med til å hente sensordata fra IMUen og beregning av sanntidsdata. Forsinkelsen for beregning av bølgeanalysen var 0.35 sekunder per periode (17 min), og av dette var selve beregningstiden på 0.28 sekunder (se figur B.4). Baud-rate for seriell-porten ble satt til 115.2 kbit/s og overføring av data fra bølgeanalysen inkludert rådata, tok 7.0 sekunder.

Testing av bølgesensoren viser at mikrokontrolleren fra ST med ARM Cortex-M, var et godt valg for denne applikasjonen. Prosesseringskraft har man mer enn nok av, og effektforbruket er lavt. Effektforbruket til MCUen ble målt til 2.3 mW, noe som er 0.9 % av forbruket til IMU-sensoren fra SBG på 255 mW. Algoritmene og beregningene som blir brukt, er forholdsvis lite beregningskrevende, men ekstra prosesseringskraft kan være nyttig ved videreutvikling til mer avanserte algoritmer og funksjoner. Som vi ser av figur 4.13 så er forsinkelsen for sanntidsdata på 11 ms, beregnet fra sensor data er mottatt av MCU til sanntidsdata er mottatt av datalogger. Over halvparten av denne forsinkelsestiden skyldes overføringstid. Så hvis man trenger enda hurtigere respons, så begynner baud-raten på 115.2 kbit/s å bli en flaskehals. Nå er forsinkelsestiden til sanntidsdata i utgangspunktet lite kritisk for en målebøye, så 11 ms er antageligvis mye bedre enn nødvendig. Men dette er greit fordi det ikke er noe i programvaren som er

blitt gjort mer komplisert for å oppnå så lav forsinkelse. Dette kan dessuten gjøre bølgesensoren relevant for andre applikasjoner, for eksempel der man vil bruke bølgeanalyse til å gi bedre estimering av sanntidsdata. For eksempel i [10] brukes bølgeamplituden og bølgefrekvensen som parametre til en adaptiv hiv-observer.

Når det gjelder minneforbruk, så fungerer det foreløpig greit med en SRAM-kapasitet på 640 KB. Når man øker perioden for bølgeanalyse fra 17 til 34 (4096 sampler på 2 Hz) minutter så øker SRAM forbruket fra 29 til 51 %, som viser at mesteparten av minneforbruket går med til bølgeanalysen. Men det er antageligvis ikke nødvendig å beregne hele frekvensspekteret for 17 minutter heller. 17 minutter (1024 s) gir en oppløsning i frekvensspekteret $\frac{1}{1024\text{s}} = 0.001\text{Hz}$. I følge [29] bør oppløsningen være 0.005Hz, for å ikke få for mye sampling-støy. Isåfall er det tilstrekkelig å bruke perioder på 256 sekunder, og for lengre perioder ta gjennomsnitt over flere 256-sekunders perioder. Dette vil man spare mye minne på å gjøre, og bølgeanalysen vil også bli vesentlig mindre beregningskrevende.

Kommunikasjon mellom bølgesensor og datalogger ser ut til å fungere bra. Når en tråd prøver å sende mens UART er opptatt så vil tråden gå i blokkerings-modus, som gir andre tråder mulighet til å kjøre eller gjør at MCUen går i sove-modus. Hvis flere tråder venter på at UART skal bli ledig, så vil tråden med høyest prioritet få bruke UARTen først. En ulempe med en slik enkel implementering er at man risikerer at en tråd blir forsinket fordi den ikke kan forsette å kjøre før UART blir ledig. En mer komplisert løsning vil derfor være å bruke en egen meldingsbehandler, som lagrer en kø med alle meldingene, og sender etterhvert som UART blir ledig. Men da bør man også vurdere å ha ulike prioriteter for meldingene, noe som også vil bidra til mer kompleksitet.

AHRSen til Hua et al. (2014) [12], viser seg å fungere bra til oppgaven. AHRSen gir bra dekobling mellom estimering av gir-vinkel, og tilting. Derfor vil ikke treg innsvingning av gir-vinkel, støy på magnetfelt, eller støy på magnetometer få påvirkning for estimeringen av rull, stamp, bølgehøyde og andre estimater som skal være uavhengig av den horisontale orienteringen / gir-vinkelen til bøya.

AHRS og bølgestimeringen ble testet på rådata fra en Fugro SEAWATCH bøye. AHRSen ble justert ved å se på tidsseriene fra rull, stamp og gir-vinkel, og gyro-biasen. Valg av justering har ingenting å si for estimeringen av bølgeparametrene h_{m0} og t_{m01}. Justering av integral-faktorene for bias-estimering har heller ikke så mye å si, fordi gyro-biasen til rådata var så stabil. Justeringen av proporsjonal-faktoren k_1 hadde litt å si for estimering av rull, stamp, gir og hiv/bølgehøyde, mens k_2 har litt å si for estimering av gir-vinkelen.

Justering av k_1 kunne gjøres enda mer nøyaktig ved å minimere støy i det lav-frekvente energispekteret til vertikal-akselerasjonen, men det er usikkert om det er minimal-punktet som gir best justering. For å finne ut av det bør man sammenligne med en referanse, slik som for eksempel GNSS-RTK.

Sensitivitet for akselerometerbias ble også undersøkt, og det ble funnet at ved betydelig bias får man avvik i alle parametrene. Dette gjelder hovedsaklig for bias i x- og y-aksen, men ikke så mye for bias i z-aksen. Avviket for bølgeparametrene kan også bli vesentlig ved store biaser. En akselerometerbias på 1 m/s^2, ga et avvik i h_{m0} på 8.5 cm eller

3.1 %. Når det gjelder avvik for gir-vinkelen, så avhenger den av at bøya ligger langt unna ekvator, slik at vertikal-komponenten til magnetfeltet blir betydelig. Rådataenes nordlige breddegrad er 62°, og det gjør at avviket for gir-vinkelen blir flere ganger større enn for rull og stamp. SBG-sensoren er oppgitt med en akselerometerbias på 0.05 m/s^2, og med denne biasen i y-aksen får man et avvik på rull og gir-vinkel på henholdsvis 0.3° og 0.9°.

5.1. Estimering av hastighet/posisjon og akselerometerbias i sanntid

Til sanntidsestimering av hiv, ble hiv-observeren i ligning 2.2 implementert. Dette er en enkel lineær hiv-observer på tilstandsrom-form, som estimerer vertikal posisjon og hastighet, ved å dobbelt-integrere vertikal-akselerasjonen. Observeren bruker en bias-tilstand i z-aksen. Hiv-observeren fungerte helt greit, men det var tydelig at man fikk avvik i estimatet på grunn av faseforskyvning, ved lengre bølgeperioder. Dette var tydelig allerede ved 11-sekunder bølgeperiode. Til forbedring av hiv-observeren finnes det flere observer-design i litteraturen som kan vurderes, se for eksempel [4, 10, 14, 21]. I software er hiv-observeren implementert ved at man legger inn matrisene fra observerens tilstandsrom-modell, og beregner kalman-forsterkningen K og kovariansmatrisen P for hvert tidssteg. Det er derfor ikke mye arbeid å bytte til en annen observer på tilstandsrom-form, også hvis observeren er ulineær.

Hiv-observeren kan utvides til å estimere relativ posisjon og hastighet horisontalt, ved å legge til 4 tilstander, slik som for de to tilstandene for posisjon- og hastighet vertikalt. Når det gjelder bias-tilstanden, så er det ikke sikkert at man trenger å legge til flere slike. Biasen til hiv-observeren er i z-aksen til NED-ramma, og tilsvarer gjennomsnittet til bøyens vertikale akselerasjon. AHRSen leser målingen fra akselerometeret inkludert bias, som gravitasjon, slik at akselerometermålingen i snitt peker vertikalt. Derfor vil man antageligvis ikke få noe sidelengs bias i NED-ramma uansett. Men for større tilte-bevegelser er det kanskje ikke optimalt å bruke en bias i z-aksen til NED-ramma. Hvis pendel-bevegelsen øker eller minsker så vil biasen forandres fordi gjennomsnittet til vertikal-akselerasjonen blir forandret. Dessuten vil biasen bare gi riktig verdi i gjennomsnitt. Hvis man for eksempel prøver å snu bølgesensoren opp-ned, så vil bias-tilstanden i NED-ramma få motsatt retning i forhold til sensoren, slik at man får store avvik mens biasen vil prøve å svinge seg inn på nytt til den motsatte verdien. Derfor vil det kanskje være en bedre idé å bruke observeren i ligning 2.1. Denne observeren er i prinsippet laget for å estimere den reelle biasen til akselerometeret, og estimerer bias i body-ramma, slik at biasen vil roteres med sensoren. Man kan da velge om man ønsker å tilbakekoble bias-estimatet til AHRSen, eller ikke. Det må man gjøre hvis det skal være noe mulighet for å estimere den reelle biasen til akselerometeret. Men det er ikke sikkert akkurat det er mulig i praksis uansett. For det første så er observeren designet med utgangspunkt i at orienteringsestimatet er korrekt uavhengig av akselerometerbias, og når det ikke er tilfelle vil man ihvertfall være avhengig av nok variasjon i orienteringstilstanden, og

for en målebøye er denne variasjonen liten. Kanskje vil det da være en bedre løsning å kombinere posisjonsobserveren med AHRSen i et kalman-filter, men man vil fremdeles være avhengig av nok variasjon i orienteringstilstanden. Foreløpig virker det også noe uklart hvilken gevinst det har i praksis å estimere den reelle biasen til akselerometerene. Derfor burde man starte med å først å teste de observerene som er foreslått, før man vurderer å bruke en mer avansert observer. Et annet alternativ er å bruke antagelsen om at kraften som virker på bøya i gjennom snitt peker vertikalt (utifra antagelsen om at bøya svinger rundt vertikal-linja). Utifra denne antagelsen burde det gå greit å estimere den reelle akselerasjonen for eksempel med AHRSen i ligningssett 4.1 i kombinasjon med posisjonsobserveren i ligning 2.1.

6. Videre arbeid

Programvaren som er blitt implementert er foreløpig bare en prototype, og det gjenstår mer arbeid før programvaren kan fungere som et ferdig produkt. Viktig funksjonalitet mangler enda, slik som mulighet for å konfigurere programvaren over seriell-porten, og lagre konfigurasjonen i flash-minne. Det er heller ikke implementert noen feilhåndtering for potensielle feil som kan oppstå. Bølgeanalysen som gjøres i software beregner bare det mest essensielle. Det kanskje viktigste forbedringspunktet er at bølge- og retningsspekteret har for høy oppløsning, som gjør at man får for mye støy på frekvensspekteret. Dette kan forbedres ved at frekvensspektrene glattes, eller at man beregner frekvenspekteret over kortere perioder, og tar gjennomsnitt over periodene. Det mangler også beregning av mange parametre, slik som gjennomsnittlig bølgeretning og bølgespredning.

Foreløpig estimeres bare bølgehøyden og orienteringen i sanntid, og ikke posisjon og hastighet og alle tre retninger. Her kan det arbeides videre med å prøve ut for eksempel de løsningene for bevegelsesestimering som er foreslått, og i hvilken grad akselerometerbias bør estimeres og hvordan.

Effekt-forbruket til SBG-sensoren ble målt 255 mW som er 100 ganger større en effekt-forbruket til mikrokontrolleren. SBG-sensoren er en moderne og konkurransedyktig IMU-sensor, og i utgangspunktet skal ikke effekt-forbruket være særlig dårligere enn konkurrentene. SBG-sensoren kan bruke en output-rate på opptil 200 Hz og en baud-rate på opptil 921600 bps, så det kan være at den er litt dårlig på å nedskalere effektforbruket ved lavere frekvenser. For en enklere IMU slik som for eksempel utvidelses-kortet X-NUCLEO-IKS01A2 fra ST til cirka 300 kroner, med akselerometer, gyro og magnetometer, så er effektforbruket oppgitt til cirka 2 mW. Til videre arbeid kunne det vært interessant å teste estimeringsalgoritmene på sensor-data fra en slik IMU og sammenligne med for eksempel SBG-sensoren.

A. Kodesnutter

A.1. Generert kode for initialisering av UART TX

DMA

For LPUART1 Tx ønsker vi å bruke kanal 1 på DMA 1. Først må klokka til DMAMUX1 og DMA1 aktiveres ved å sette bit 0 (DMA1 clock enable) og bit 2 (DMAMUX clock enable) i AHB1 peripheral clock enable register (RCC->AHB1ENR):2

```
LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_DMAMUX1);
LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_DMA1);
```

Når DMA er ferdig med å overføre alle bytes i bufferet, må prosessoren få beskjed. Vi ønsker derfor å bruke interrupt for DMA1. Først må prioriteten til interruptet settes. I kodegeneratoren er standard prioritet lik 5 når man bruker FreeRTOS. Fra referanse manualen [24] side 473 er IRQ lik 12 for kanal 2 på DMA1. Derfor setter vi Interrupt priority register NVIC->IP[12] = 0x50:

```
NVIC_SetPriority(DMA1_Channel2_IRQn, NVIC_EncodePriority(
   NVIC_GetPriorityGrouping(),5, 0));
```

For å aktivere IRQ 12 setter vi bit 12 i Interrupt set-enable register 0 (NVIC->ISER[0]):

```
NVIC_EnableIRQ(DMA1_Channel2_IRQn);
```

For å koble DMA1 kanal 2 til LPUART1_TX må bit-felt 0-6 (DMA request identification) settes i DMAMUX request line multiplexer channel 1 configuration register (DMAMUX1->DMAMUX_C1CR). I følge referansemanualen [24] side 399, skal verdien være 35:

```
LL_DMA_SetPeriphRequest(DMA1, LL_DMA_CHANNEL_2,
   LL_DMAMUX_REQ_LPUART1_TX);
```

Siden dataoverføringen skal skje fra RAM til UART (ikke fra UART til RAM) settes Bit 4 (Data transfer direction) til 1 i Channel 2 configuration register (DMA1->CCR2):

```
LL_DMA_SetDataTransferDirection(DMA1, LL_DMA_CHANNEL_2,
   LL_DMA_DIRECTION_MEMORY_TO_PERIPH);
```

For at minneadressen skal inkrementeres etter hver byte som overføres, må bit 7 (Memory increment mode) settes til 1 i Channel 2 configuration register (DMA1->CCR2):

```
LL_DMA_SetMemoryIncMode(DMA1, LL_DMA_CHANNEL_2,
   LL_DMA_MEMORY_INCREMENT);
```

GPIO

TX for LPUART1 skal bruke port G pinne 7. Klokka til port G må derfor aktiveres ved å sette bit 6 (GPIOG clock enable) i registeret APB1 peripheral clock enable register 2 (RCC->AHB2ENR):

```
LL_AHB2_GRP1_EnableClock(LL_AHB2_GRP1_PERIPH_GPIOG);
```

Som et unntak fra de andre pinnene får Port G, pinne 2 til 15 forsyningsstrøm fra pinne Vdd_io2. For å kunne bruke disse pinnene må vi derfor sette bit 9 (VDDIO2 Independent I/Os supply valid) i Power control register 2 (PWR->CR2):

```
LL_PWR_EnableVddIO2();
```

GPIO port mode register x kan settes til 4 forskjellige moduser: Input, General purpose output, Alternate function og Analog. Siden vi skal bruke LPUART bruker vi Alternate function på port G7. Vi setter bit-felt 14-15 (MODER7) i GPIOG port mode register (GPIOG->MODER) til 10_2.

```
LL_GPIO_SetPinMode(GPIOG, LL_GPIO_PIN_7,
   LL_GPIO_MODE_ALTERNATE);
```

Når UART brukes som Alternate function blir hastigheten på GPIO-pinnen satt til det høyeste som standard i kodegeneratoren. Altså blir bit-felt 14-15 (OSPEEDR7) på GPIOG port output speed register (GPIOG->OSPEEDR) satt til 11_2:

```
LL_GPIO_SetPinSpeed(GPIOG, LL_GPIO_PIN_7,
   LL_GPIO_SPEED_FREQ_VERY_HIGH);
```

Port G7 har flere alternative funksjoner. Fra databladet [26] side 132 ser vi at LPUART1_TX er alternativ funksjon 8. Derfor settes bit-felt 28-31 (AFRL7) i GPIOG alternate function low register (GPIOG->AFRL) til 0x8:

```
LL_GPIO_SetAFPin_0_7(GPIOG, LL_GPIO_PIN_7, LL_GPIO_AF_8);
```

UART

Vi aktiverer klokka for LPUART1 ved å sette bit 0 (LPUART1 clock enable) i APB1 peripheral clock enable register 2 (RCC->APB1ENR2):

```
LL_APB1_GRP2_EnableClock(LL_APB1_GRP2_PERIPH_LPUART1);
```

For å aktivere sender og mottaker for LPUART1 må bit 3 (Transmitter enable) settes til 1 i Control register 1 (LPUART1->CR1):

```
WRITE_REG(LPUART1->CR1, USART_CR1_TE);
```

For å konfigurere baud-rate setter man USART Baud rate register (LPUART1->BRR). Hva man setter dette registeret til, avhenger av baudrate, klokkefrekvensen og prescaleren til LPUART1. Klokkefrekvensen er satt til standard-verdi i kodeeditoren, dvs til system klokka som er satt til 4 MHz, mens prescaler er satt til 1 som er lik reset value og standard-verdi i kodegenerator. Baud rate skal være 115 200 Bits/s:

```
LL_LPUART_SetBaudRate(4000000, LL_LPUART_PRESCALER_DIV1,
   115200);
```

Til slutt må LPUART1 aktiveres ved å sette bit 0 (USART enable) i Control register 1 (LPUART1->CR1):

```
LL_LPUART_Enable(LPUART1);
```

A.2. Kommunikasjonsprotokoll

A.2.1. Kommunikasjons-protokoll fra software-pakken X-CUBE-MEMS1 fra ST

```
/**
 ******************************************************************************

 * @file    serial_protocol.c
 * @brief   This file implements some utilities for the serial protocol
 ******************************************************************************

 * @attention
 *
 * <h2><center>&copy; COPYRIGHT(c) 2019 STMicroelectronics</center></h2
    >
 *
 * Redistribution and use in source and binary forms, with or without
    modification,
 * are permitted provided that the following conditions are met:
 *   1. Redistributions of source code must retain the above copyright
    notice,
 *      this list of conditions and the following disclaimer.
 *   2. Redistributions in binary form must reproduce the above
    copyright notice,
 *      this list of conditions and the following disclaimer in the
    documentation
 *      and/or other materials provided with the distribution.
 *   3. Neither the name of STMicroelectronics nor the names of its
    contributors
 *      may be used to endorse or promote products derived from this
    software
```

```
 *      without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
    "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO
    , THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
    PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS
    BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
    CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
    GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
    HOWEVER
 * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
    LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
    OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 ******************************************************************************

 */

/* Includes
   ------------------------------------------------------------------
   */
#include <stdint.h>

/* Exported defines
   -----------------------------------------------------------*/
#define Msg_EOF                 0xF0
#define Msg_BS                  0xF1
#define Msg_BS_EOF              0xF2

#define Msg_maxLen              2054

#define CMD_Reply_Add   0x80U
#define CMD_Nack_Add    0x40U

/* Exported types
   -----------------------------------------------------------*/
/**
 * @brief  Serial message structure definition
 */
typedef struct
{
  uint32_t Len;
  uint8_t Data[Msg_maxLen];
} Msg_t;
```

```c
/**
 * @brief  Byte stuffing process for one byte
 * @param  Dest destination
 * @param  Source source
 * @retval Total number of bytes processed
 */
static int ByteStuffCopyByte(uint8_t *Dest, uint8_t Source)
{
  int ret = 2;

  switch (Source)
  {
    case Msg_EOF:
      Dest[0] = Msg_BS;
      Dest[1] = Msg_BS_EOF;
      break;

    case Msg_BS:
      Dest[0] = Msg_BS;
      Dest[1] = Msg_BS;
      break;

    default:
      Dest[0] = Source;
      ret = 1;
      break;
  }

  return ret;
}

/**
 * @brief  Byte stuffing process for a Msg
 * @param  Dest destination
 * @param  Source source
 * @retval Total number of bytes processed
 */
int ByteStuffCopy(uint8_t *Dest, Msg_t *Source)
{
  uint32_t i;
  int32_t count = 0;

  for (i = 0; i < Source->Len; i++)
  {
    count += ByteStuffCopyByte(&Dest[count], Source->Data[i]);
  }
  Dest[count] = Msg_EOF;
  count++;
  return count;
}
```

```
/**
 * @brief  Reverse Byte stuffing process for two input data
 * @param  Source0 input data
 * @param  Source1 input data
 * @param  Dest the destination data
 * @retval Number of input bytes processed (1 or 2) or 0 for invalid
    sequence
 */
int ReverseByteStuffCopyByte2(uint8_t Source0, uint8_t Source1, uint8_t
    *Dest)
{
  if (Source0 == (uint8_t)Msg_BS)
  {
    if (Source1 == (uint8_t)Msg_BS)
    {
      *Dest = Msg_BS;
      return 2;
    }
    if (Source1 == (uint8_t)Msg_BS_EOF)
    {
      *Dest = Msg_EOF;
      return 2;
    }
    return 0; // invalid sequence
  }
  else
  {
    *Dest = Source0;
    return 1;
  }
}

/**
 * @brief  Compute and add checksum
 * @param  Msg pointer to the message
 * @retval None
 */
void CHK_ComputeAndAdd(Msg_t *Msg)
{
  uint8_t chk = 0;
  uint32_t i;

  for (i = 0; i < Msg->Len; i++)
  {
    chk -= Msg->Data[i];
  }
  Msg->Data[i] = chk;
  Msg->Len++;
}

/**
 * @brief  Compute and remove checksum
```

```
 * @param  Msg pointer to the message
 * @retval A number different from 0 if the operation succeeds, 0 if an
     error occurs
 */
int CHK_CheckAndRemove(Msg_t *Msg)
{
  uint8_t chk = 0;
  uint32_t i;

  for (i = 0; i < Msg->Len; i++)
  {
    chk += Msg->Data[i];
  }
  Msg->Len--;
  return (int32_t)(chk == 0U);
}

void BuildReplyHeader(Msg_t *Msg) {
  Msg->Data[0] += CMD_Reply_Add;
}

void BuildNackHeader(Msg_t *Msg) {
  Msg->Data[0] += CMD_Nack_Add;
}

/**
 * @}
 */

/**
 * @}
 */

/******************* (C) COPYRIGHT STMicroelectronics *****END OF FILE
    ****/
```

A.2.2. Kommandoer

```
typedef enum
{
  CMD_LOG = 0,
  CMD_MSG,
  CMD_NEW_DATA,
  CMD_GET_TICK,
  CMD_WAVE_PARAMS,
  CMD_AZ_NED,
  CMD_NORTH,
  CMD_EAST,
  CMD_HEAVE,
  CMD_CHH,
  CMD_RESET,
  CMD_A1,
```

```
  CMD_B1,
  CMD_A2,
  CMD_B2
} SerialCMD;
```

A.3. UART Rx med DMA (sirkulært buffer)

```
#define Lpuart1_rxBufferSize 512
static uint8_t Lpuart1_rxBuffer[Lpuart1_rxBufferSize];

Msg_t rxMsg;

void LPUART1_RX_Init()
{
  /* LPUART1 Rx */
  LL_LPUART_EnableDMAReq_RX(LPUART1);

  LL_DMA_ConfigAddresses(DMA1, LL_DMA_CHANNEL_1,
                         LL_LPUART_DMA_GetRegAddr(LPUART1,
                            LL_LPUART_DMA_REG_DATA_RECEIVE),
                         (uint32_t)Lpuart1_rxBuffer,
                         LL_DMA_GetDataTransferDirection(DMA1,
                            LL_DMA_CHANNEL_1));
  LL_DMA_SetDataLength(DMA1, LL_DMA_CHANNEL_1, Lpuart1_rxBufferSize);

  LL_DMA_EnableChannel(DMA1, LL_DMA_CHANNEL_1);
}

/**
 * @brief  Check if a message is received via UART
 * @retval 1 if a complete message is found, 0 otherwise
 */
int LPUART1_ReceivedMSG()
{
  int i, i2, eof;
  int dma_counter, length;
  int inc;

  static uint16_t startOfMsg = 0;

  dma_counter = Lpuart1_rxBufferSize - LL_DMA_GetDataLength(DMA1,
     LL_DMA_CHANNEL_1);

  eof = startOfMsg;

  while (1)
  {
    if (eof == dma_counter)
    {
      return 0;
    }
```

```
    else if (Lpuart1_rxBuffer[eof] == Msg_EOF)
    {
      break;
    }
    eof = (eof+1) % Lpuart1_rxBufferSize;
  }

  if (eof == startOfMsg)
  {
    return 0;
  }

  i = startOfMsg;
  length = 0;
  while (i != eof)
  {
    if (length >= lpuart1_rxMsgMaxSize)
    {
      startOfMsg = (eof + 1) % Lpuart1_rxBufferSize;
      return 0;
    }

    i2 = (i + 1) % Lpuart1_rxBufferSize;
    inc = ReverseByteStuffCopyByte2(Lpuart1_rxBuffer[i],
       Lpuart1_rxBuffer[i2], rxMsg.Data+length);
    if (inc == 0U)
    {
      startOfMsg = (eof + 1) % Lpuart1_rxBufferSize;
      return 0; /* Invalid sequence */
    }
    i = (i + inc) % Lpuart1_rxBufferSize;
    length++;
  }

  rxMsg.Len = length;
  startOfMsg = (i + 1) % Lpuart1_rxBufferSize; /* skip EOF */

  if (CHK_CheckAndRemove(&rxMsg) == 0) /* check message integrity */
  {
    return 0;
  }

  return 1;
}
```

A.4. Beregning av sanntids bølgehøyde for mikrokontroller

```
#define Q_A 1.0f
#define Q_B 1.0f
```

```
#define R_H  4.0f

#define G 9.80665f

#define H SBG_TIMESTEP

#define N 3

static void Kalmand();

static float R_[] = {R_H/H};
static float P_[N*N];

static float x_[N];
static float u_[1];
static float K_[N];

const float * heaveRT = x_;

static float C_[] = {1.0f, 0.0f, 0.0f};
static float Q_[] = {
    0.0f,   0.0f,   0.0f,
    0.0f,   Q_A*H,  0.0f,
    0.0f,   0.0f,   Q_B*H
};
static float A_[] = {
    1.0f,   H,      0.0f,
    0.0f,   1.0f,   -H,
    0.0f,   0.0f,   1.0f
};
static float B_[] = {0.0f, H, 0.0f};

static float I_[] = {
    1.0f,   0.0f,   0.0f,
    0.0f,   1.0f,   0.0f,
    0.0f,   0.0f,   1.0f
};

static arm_matrix_instance_f32 P, C, Q, R, A, B, x, u, K, I;

void heave_Init()
{
  arm_mat_init_f32(&P, N, N, P_);
  arm_mat_init_f32(&C, 1, N, C_);
  arm_mat_init_f32(&Q, N, N, Q_);
  arm_mat_init_f32(&R, 1, 1, R_);
  arm_mat_init_f32(&A, N, N, A_);
  arm_mat_init_f32(&B, N, 1, B_);
  arm_mat_init_f32(&x, N, 1, x_);
  arm_mat_init_f32(&u, 1, 1, u_);
  arm_mat_init_f32(&K, N, 1, K_);
  arm_mat_init_f32(&I, N, N, I_);
```

```
  arm_mat_scale_f32(&Q, 10.0f, &P);
}

void heave_Update()
{

  float acc_ned[3];

  QUAT_ROTATE(ahrs_q, sbg_data->acc, acc_ned);

  *u_ = acc_ned[2] + G;

  Kalmand();
}

/*
def kalmand(x, P, u, A, B, C, Q, R):
    x = A.dot(x) + B.dot(u)
    P = A.dot(P).dot(A.T) + Q
    K = P.dot(C.T).dot(scipy.linalg.inv(C.dot(P).dot(C.T) + R))
    temp = np.eye(len(P)) - K.dot(C)
    P = temp.dot(P).dot(temp.T) + K.dot(R).dot(K.T)
    x = x - K.dot(C).dot(x)

    return x, P
 */
static void Kalmand()
{
  float temp1_[N*N];
  float temp2_[N*N];
  float temp3_[N*N];

  size_t m = sizeof(C_) / sizeof(C_[0]) / N;
  arm_matrix_instance_f32 temp1;
  arm_matrix_instance_f32 temp2;
  arm_matrix_instance_f32 temp3;

  /* x = A.dot(x) + B.dot(u) */
  arm_mat_init_f32(&temp1, N, 1, temp1_);
  arm_mat_mult_f32(&A, &x, &temp1);

  arm_mat_init_f32(&temp2, N, 1, temp2_);
  arm_mat_mult_f32(&B, &u, &temp2);

  arm_mat_add_f32(&temp1, &temp2, &x);

  /* P = A.dot(P).dot(A.T) + Q */
  arm_mat_init_f32(&temp1, N, N, temp1_);
  arm_mat_trans_f32(&A, &temp1);

  arm_mat_init_f32(&temp2, N, N, temp2_);
```

```
arm_mat_mult_f32(&P, &temp1, &temp2);

arm_mat_mult_f32(&A, &temp2, &temp1);

arm_mat_add_f32(&temp1, &Q, &P);

/* K = P.dot(C.T).dot(scipy.linalg.inv(C.dot(P).dot(C.T) + R)) */
arm_mat_init_f32(&temp1, N, m, temp1_);
arm_mat_trans_f32(&C, &temp1);

arm_mat_init_f32(&temp2, N, m, temp2_);
arm_mat_mult_f32(&P, &temp1, &temp2); /* temp2 = P.dot(C.T) */

arm_mat_init_f32(&temp1, m, m, temp1_);
arm_mat_mult_f32(&C, &temp2, &temp1);

arm_mat_init_f32(&temp3, m, m, temp3_);
arm_mat_add_f32(&temp1, &R, &temp3);

arm_mat_inverse_f32(&temp3, &temp1); /* temp1 = inv(C.dot(P).dot(C.T)
    + R) */

arm_mat_mult_f32(&temp2, &temp1, &K);

/* temp = np.eye(len(P)) - K.dot(C) */
arm_mat_init_f32(&temp1, N, N, temp1_);
arm_mat_mult_f32(&K, &C, &temp1);

arm_mat_init_f32(&temp2, N, N, temp2_);
arm_mat_sub_f32(&I, &temp1, &temp2); /* temp2 = temp */

/* P = temp.dot(P).dot(temp.T) + K.dot(R).dot(K.T) */
arm_mat_trans_f32(&temp2, &temp1);

arm_mat_init_f32(&temp3, N, N, temp3_);
arm_mat_mult_f32(&P, &temp1, &temp3);

arm_mat_mult_f32(&temp2, &temp3, &temp1); /* temp1 = temp.dot(P).dot(
   temp.T) */

arm_mat_init_f32(&temp2, m, N, temp2_);
arm_mat_trans_f32(&K, &temp2);

arm_mat_init_f32(&temp3, m, N, temp3_);
arm_mat_mult_f32(&R, &temp2, &temp3);

arm_mat_init_f32(&temp2, N, N, temp2_);
arm_mat_mult_f32(&K, &temp3, &temp2); /* temp2 = K.dot(R).dot(K.T) */

arm_mat_add_f32(&temp1, &temp2, &P);

/* x = x - K.dot(C).dot(x) */
```

```
  arm_mat_init_f32(&temp1, m, 1, temp1_);
  arm_mat_mult_f32(&C, &x, &temp1);

  arm_mat_init_f32(&temp2, N, 1, temp2_);
  arm_mat_mult_f32(&K, &temp1, &temp2);

  arm_mat_init_f32(&temp1, N, 1, temp1_);
  arm_mat_sub_f32(&x, &temp2, &temp1);

  arm_copy_f32(temp1_, x_, N);
}
```

A.5. Beregning av orientering for mikrokontroller

```
/*
  import sympy as sp
  from sympy.algebras.quaternion import Quaternion
  q = Quaternion(*(sp.symbols('(q)[0:3]')))
  R = q.to_rotation_matrix()
  R = R.subs(q.norm()**2, 1)
 */
#define QUAT_ROTATE(q, in, out) \
  do  { \
    (out)[0] =  (-2*(q)[2]*(q)[2] - 2*(q)[3]*(q)[3] + 1)  * (in)[0]  +
        \
                (-2*(q)[0]*(q)[3] + 2*(q)[1]*(q)[2])      * (in)[1]  +
                    \
                (2*(q)[0]*(q)[2] + 2*(q)[1]*(q)[3])       * (in)[2];
                      \
\
    (out)[1] = (2*(q)[0]*(q)[3] + 2*(q)[1]*(q)[2])        * (in)[0]  +
        \
               (-2*(q)[1]*(q)[1] - 2*(q)[3]*(q)[3] + 1)   * (in)[1]  +
                   \
               (-2*(q)[0]*(q)[1] + 2*(q)[2]*(q)[3])       * (in)[2];
                     \
\
    (out)[2] = (-2*(q)[0]*(q)[2] + 2*(q)[1]*(q)[3])       * (in)[0]  +
        \
               (2*(q)[0]*(q)[1] + 2*(q)[2]*(q)[3])        * (in)[1]  +
                   \
               (-2*(q)[1]*(q)[1] - 2*(q)[2]*(q)[2] + 1)   * (in)[2];
                     \
  }while(0)


/*
  import sympy as sp
  x = sp.Matrix(sp.symbols('(x)[0:3]'))
  y = sp.Matrix(sp.symbols('(y)[0:3]'))
  x.cross(y)
 */
```

```
#define CROSS_PRODUCT(out, x, y)  \
  do  { \
    (out)[0] = (x)[1]*(y)[2] - (x)[2]*(y)[1];  \
    (out)[1] = -(x)[0]*(y)[2] + (x)[2]*(y)[0]; \
    (out)[2] = (x)[0]*(y)[1] - (x)[1]*(y)[0]; \
  }while(0)

static float q[4] = {1.0f, 0.0f, 0.0f, 0.0f};
static float bias[3];

static const float k1 = 0.25f;
static const float k2 = 0.5f;
static const float k3 = 0.005f;
static const float k4 = 0.005f;

static const float kb = 0.0f;
static const float delta = 0.0f;

static void HuaMahonyAHRSupdate()
{
  float norm;

  float mag[3]; float acc[3];

  float m_proj[3];

  float v[3];
  float w[3];

  float e_acc[3];
  float e_mag[3];
  float e[3];
  float e_bias[3];

  float temp[3];
  float temp2[3];

  const float h = SBG_TIMESTEP;

  arm_copy_f32(sbg_data->acc, acc, 3);
  arm_copy_f32(sbg_data->mag, mag, 3);

  /* Normalise accelerometer measurement */
  arm_dot_prod_f32(acc, acc, 3, &norm);
  arm_sqrt_f32(norm, &norm);
  arm_scale_f32(acc, 1/norm, acc, 3);

/*
  import sympy as sp
  from sympy.algebras.quaternion import Quaternion

  acc = sp.Matrix(sp.symbols('acc[0:3]'))
```

```
  mag = sp.Matrix(sp.symbols('mag[0:3]'))
  gyro = sp.Matrix(sp.symbols('gyro[0:3]'))

  mag_proj = (acc.dot(acc) * sp.eye(3) - acc * acc.T) * mag
*/
  m_proj[0] = -acc[0]*acc[1]*mag[1] - acc[0]*acc[2]*mag[2] + mag[0]*(
     acc[1]*acc[1] + acc[2]*acc[2]);
  m_proj[1] = -acc[0]*acc[1]*mag[0] - acc[1]*acc[2]*mag[2] + mag[1]*(
     acc[0]*acc[0] + acc[2]*acc[2]);
  m_proj[2] = -acc[0]*acc[2]*mag[0] - acc[1]*acc[2]*mag[1] + mag[2]*(
     acc[0]*acc[0] + acc[1]*acc[1]);

  arm_dot_prod_f32(m_proj, m_proj, 3, &norm);
  arm_sqrt_f32(norm, &norm);
  arm_scale_f32(m_proj, 1/norm, m_proj, 3);

 /*
  q = Quaternion(*(sp.symbols('q[0:4]')))
  R = q.to_rotation_matrix()
  R = R.subs(q.norm()**2, 1)

  v = R.T * sp.Matrix([0, 0, -1])
  w = R.T * sp.Matrix([1, 0, 0])
*/

  /* Estimated direction of gravity and magnetic field */

  v[0] = 2*q[0]*q[2] - 2*q[1]*q[3];
  v[1] = -2*q[0]*q[1] - 2*q[2]*q[3];
  v[2] = 2*q[1]*q[1] + 2*q[2]*q[2] - 1;

  w[0] = -2*q[2]*q[2] - 2*q[3]*q[3] + 1;
  w[1] = -2*q[0]*q[3] + 2*q[1]*q[2];
  w[2] = 2*q[0]*q[2] + 2*q[1]*q[3];

  CROSS_PRODUCT(e_acc, acc, v);
  CROSS_PRODUCT(e_mag, m_proj, w);


/*
  k1, k2 = sp.symbols('k1, k2')
  v = sp.Matrix(sp.symbols('v[0:3]'))
  e_mag = sp.Matrix(sp.symbols('e_mag[0:3]'))

  e = k1 * e_acc + k2 * v * v.T * e_mag
 */

  e[0] = e_acc[0]*k1 + e_mag[0]*k2*v[0]*v[0] + e_mag[1]*k2*v[0]*v[1] +
     e_mag[2]*k2*v[0]*v[2];
  e[1] = e_acc[1]*k1 + e_mag[0]*k2*v[0]*v[1] + e_mag[1]*k2*v[1]*v[1] +
     e_mag[2]*k2*v[1]*v[2];
  e[2] = e_acc[2]*k1 + e_mag[0]*k2*v[0]*v[2] + e_mag[1]*k2*v[1]*v[2] +
```

```
      e_mag[2]*k2*v[2]*v[2];


  if (k3 > 0 || k4 > 0)
  {
    e_bias[0] = -e_acc[0]*k3 - e_mag[0]*k4;
    e_bias[1] = -e_acc[1]*k3 - e_mag[1]*k4;
    e_bias[2] = -e_acc[2]*k3 - e_mag[2]*k4;

    do
    {
      if (kb != 0)
      {
        arm_dot_prod_f32(bias, bias, 3, &norm);
        arm_sqrt_f32(norm, &norm);
        if (norm > delta)
        {
          bias[0] += h*(-bias[0]*kb*(-delta/norm + 1) + e_bias[0]);
          bias[1] += h*(-bias[1]*kb*(-delta/norm + 1) + e_bias[1]);
          bias[2] += h*(-bias[2]*kb*(-delta/norm + 1) + e_bias[2]);
          break;
        }
      }
      bias[0] += e_bias[0]*h;
      bias[1] += e_bias[1]*h;
      bias[2] += e_bias[2]*h;
    }while(0);

    arm_add_f32(sbg_data->gyro, e, temp2, 3);
    arm_sub_f32(temp2, bias, temp, 3);

/*
  h = sp.symbols('h')
  temp = sp.Matrix(sp.symbols('temp[0:3]'))
  q = Quaternion(*(sp.symbols('q[0:4]')))

  q += 0.5 * q * Quaternion(0, *temp) * h
  q = sp.simplify(q)
 */

    q[0] += -0.5*h*(q[1]*temp[0] + q[2]*temp[1] + q[3]*temp[2]);
    q[1] += 0.5*h*(q[0]*temp[0] + q[2]*temp[2] - q[3]*temp[1]);
    q[2] += 0.5*h*(q[0]*temp[1] - q[1]*temp[2] + q[3]*temp[0]);
    q[3] += 0.5*h*(q[0]*temp[2] + q[1]*temp[1] - q[2]*temp[0]);

    arm_dot_prod_f32(q, q, 4, &norm);
    arm_sqrt_f32(norm, &norm);
    arm_scale_f32(q, 1/norm, q, 4);
  }

}
```

A.6. Beregning av bølgestatistikk for mikrokontroller

```
#define QUAT_TO_EAST(q) -2*((q)[1]*(q)[3] - (q)[2]*(q)[0])
#define QUAT_TO_NORTH(q) 2*((q)[2]*(q)[3] - (q)[1]*(q)[0])

static WaveRawData_t rawData1;
static WaveRawData_t rawData2;

const WaveRawData_t * waveRawData;

void waveRawData_Update()
{

  static int firstRun = 1;
  static WaveRawData_t * currentData = &rawData1;

  static int j = 0;
  static int i = 0;

  static float az_ned_mean = 0.0f;
  static float north_mean = 0.0f;
  static float east_mean = 0.0f;

  int n = lroundf(1 / (SBG_TIMESTEP * WAVE_DATA_RATE));

  float acc_ned[3];

  QUAT_ROTATE(ahrs_q, sbg_data->acc, acc_ned);

  az_ned_mean += acc_ned[2];
  east_mean += QUAT_TO_EAST(ahrs_q);
  north_mean += QUAT_TO_NORTH(ahrs_q);

  if (++j == n)
  {
    j = 0;
    currentData->az_ned[i] = az_ned_mean/n;

    az_ned_mean = 0.0f;

    currentData->north[i] = north_mean/n;
    currentData->east[i] = east_mean/n;

    north_mean = 0.0f;
    east_mean = 0.0f;

    if (++i == WAVE_DATA_LENGTH)
    {
      i = 0;


      if (!firstRun)
```

```
      {
        osSemaphoreAcquire(waveDataBinarySemHandle, osWaitForever);
        __asm volatile( "" ::: "memory" );
      }

      currentData->timeStamp = GetTick();

      if (currentData == &rawData1)
      {
        currentData = &rawData2;
        waveRawData = &rawData1;
      }
      else
      {
        currentData = &rawData1;
        waveRawData = &rawData2;
      }

      __asm volatile( "" ::: "memory" );
      osThreadFlagsSet(waveTaskHandle, WAVE_RUN_FLAG);
    }
  }
}

#define DATA_LENGTH 2048
#define FREQ_LENGTH (DATA_LENGTH/2 + 1)

static WaveData_t waveData;

static float temp1[DATA_LENGTH];
static float temp2[DATA_LENGTH];
static float temp3[DATA_LENGTH];

void wave_Calc()
{

  float *H;
  float g;

  arm_copy_f32(waveRawData->az_ned, temp1, DATA_LENGTH);

  float *N, *E;

  float *a1, *b1, *a2, *b2;

  arm_copy_f32(waveRawData->north, temp2, DATA_LENGTH);
  arm_copy_f32(waveRawData->east, temp3, DATA_LENGTH);

  waveData.timeStamp = waveRawData->timeStamp;

  /* Subtract gravity (or mean) */
  arm_mean_f32(temp1, DATA_LENGTH, &g);
  arm_offset_f32(temp1, -g, temp1, DATA_LENGTH);
```

```
H = waveData.a1b1; /* Temporary lending memory-space */
arm_rfft_fast_f32(&arm_rfft_instance, temp1, H, 0);

N = waveData.heave; /* Temporary lending memory-space */
arm_rfft_fast_f32(&arm_rfft_instance, temp2, N, 0);
HpFilterF(N);

E = waveData.a2b2; /* Temporary lending memory-space */
arm_rfft_fast_f32(&arm_rfft_instance, temp3, E, 0);
HpFilterF(E);

HpFilterF(H);

arm_mult_f32(H, freqInv2, temp1, DATA_LENGTH); /* Double integration,
    by dividing with frequency squared */
arm_scale_f32(temp1, 1 / (4 * PI * PI), temp1, DATA_LENGTH);
arm_copy_f32(temp1, H, DATA_LENGTH);

CrossSpectra(H, H, waveData.Chh, NULL, temp1, temp2, temp3);

CrossSpectra(N, N, waveData.Cxx, NULL, temp1, temp2, temp3);
CrossSpectra(E, E, waveData.Cyy, NULL, temp1, temp2, temp3);
CrossSpectra(N, E, waveData.Cxy, NULL, temp1, temp2, temp3);
CrossSpectra(H, N, NULL, waveData.Qhx, temp1, temp2, temp3);
CrossSpectra(H, E, NULL, waveData.Qhy, temp1, temp2, temp3);

arm_rfft_fast_f32(&arm_rfft_instance, H, waveData.heave, 1); /*
   Inverse FFT */

a1 = waveData.a1b1;
b1 = waveData.a1b1 + FREQ_LENGTH;
a2 = waveData.a2b2;
b2 = waveData.a2b2 + FREQ_LENGTH;

CalcDirSpec(waveData.Chh, waveData.Cxx, waveData.Cyy, waveData.Cxy,
    waveData.Qhx, waveData.Qhy, a1, b1, a2, b2, temp1, temp2, temp3);

wave_CalcParams(temp1, temp2); /* Calculating wave-parameters from
   wave-spectrum Chh */

__asm volatile( "" ::: "memory" );
osSemaphoreRelease(waveDataBinarySemHandle);
osSemaphoreAcquire(transferDataBinarySemHandle, osWaitForever);
__asm volatile( "" ::: "memory" );

txMsg.Data[0] = CMD_NEW_DATA;
txMsg.Len = 1;

uint32_t tick = GetTick();
memcpy(txMsg.Data + txMsg.Len, (void*) &tick, 4);
txMsg.Len += 4;
```

```
  memcpy(txMsg.Data + txMsg.Len, &waveData.timeStamp, 4);
  txMsg.Len += 4;

  LPUART1_SendMsg();
}
```

A.7. AHRS-observeren fra Hua et al. (2014) i python-kode

```
import numpy as np
import quaternion as quat
from numpy.linalg import norm

class HuaMahonyAHRS():

  def __init__(self, rate, k1, k2, k3, k4, kb, delta):
    self.rate = rate
    self.samplePeriod = 1 / rate
    self.k1 = k1
    self.k2 = k2
    self.k3 = k3
    self.k4 = k4
    self.kb = kb
    self.delta = delta
    self.q = quat.quaternion(1, 0, 0, 0)
    self.bias = np.array([0., 0., 0.])

  def update(self, gyro, acc, mag):

    q = self.q

    # Normalise accelerometer measurement
    acc /= norm(acc)

    mag_proj = (np.sum(acc ** 2) * np.identity(3) -
                np.outer(acc, acc)).dot(mag)
    mag_proj /= norm(mag_proj)

    # Estimated direction of gravity and magnetic field
    v = -np.array([
      2 * (q.x * q.z - q.w * q.y),
      2 * (q.w * q.x + q.y * q.z),
      q.w ** 2 - q.x ** 2 - q.y ** 2 + q.z ** 2])

    w = np.array([
      2 * (0.5 - q.y ** 2 - q.z ** 2),
      2 * (q.x * q.y - q.w * q.z),
      2 * (q.w * q.y + q.x * q.z)])

    e_acc = np.cross(acc, v)
```

```
    e_mag = np.cross(mag_proj, w)

    # Error
    e = self.k1 * e_acc + self.k2 * np.outer(v, v).dot(e_mag)

    if self.k3 > 0 or self.k4 > 0:
      e_bias = - self.k3 * e_acc - self.k4 * e_mag

      if self.kb == 0 or norm(self.bias) <= self.delta:
        self.bias += e_bias * self.samplePeriod
      else:
        self.bias += (-self.kb * self.bias * (1 - self.delta / norm(
            self.bias)) + e_bias) * self.samplePeriod

  # Apply feedback terms
    gyro += e - self.bias

    # Compute rate of change of quaternion
    qDot = 0.5 * q * quat.quaternion(0, gyro[0], gyro[1], gyro[2])

    # Integrate to yield quaternion
    q += qDot * self.samplePeriod

    self.q = q / q.abs()
```

A.8. Beregning av bølgeparametre i Python

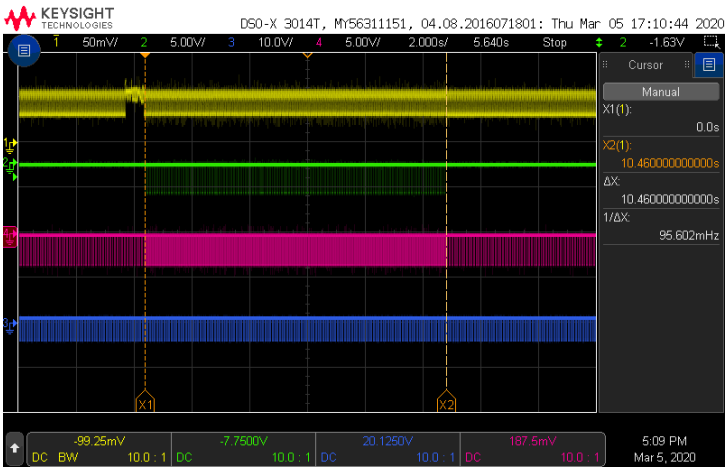
```
import numpy as np


class WaveCalc:
  data: WaveData

  def __init__(self, az_ned, north, east, rate, t_hp):
    self.f_hp = 1.0 / t_hp
    self.az = az_ned - np.mean(az_ned)
    self.north = north
    self.east = east
    self.length = len(az_ned)
    self.rate = rate
    self.scale = 2 / (self.length * self.rate)
    self.data = WaveData()
    self.data.freq = np.fft.rfftfreq(self.length, 1. / self.rate)
    self.deltaFreq = self.rate / self.length

    self._calc()

  def _calc(self):

    freq_ = self.data.freq.copy()
    freq_[0] = 1  # Avoiding division by zero
    self.Az = np.fft.rfft(self.az)
```

```
self.Heave = self.Az / (2 * pi * freq_) ** 2

n = math.ceil(self.f_hp * self.length / self.rate)
if n == 0:
  n = 1
self.Heave[0:n] = 0.0
self.data.Chh = abs(self.Heave ** 2) * self.scale
self.data.Chh[-1] /= 2
self.data.heave = np.fft.irfft(self.Heave)

self.North = np.fft.rfft(self.north)
self.North[0:n] = 0.0
self.North[-1] /= np.sqrt(2)
self.East = np.fft.rfft(self.east)
self.East[0:n] = 0.0
self.East[-1] /= np.sqrt(2)

Qhn = - (self.Heave.conj() * self.North).imag * self.scale
Qhe = - (self.Heave.conj() * self.East).imag * self.scale
Cne = (self.North.conj() * self.East).real * self.scale

Chn = - (self.Heave.conj() * self.North).real * self.scale
Che = - (self.Heave.conj() * self.East).real * self.scale

Cnn = abs(self.North ** 2) * self.scale
Cee = abs(self.East ** 2) * self.scale

m = len(Cnn)
self.data.a1 = np.empty(m, dtype=float)
self.data.b1 = np.empty(m, dtype=float)
self.data.a2 = np.empty(m, dtype=float)
self.data.b2 = np.empty(m, dtype=float)

for i in range(0, m):
  d = np.sqrt(Chh[i] * (Cnn[i] + Cee[i]))
  if d != 0.:
    self.data.a1[i] = Qhn[i] / d
    self.data.b1[i] = Qhe[i] / d
  else:
    self.data.a1[i] = 0.
    self.data.b1[i] = 0.
  d = Cnn[i] + Cee[i]
  if d != 0.0:
    self.data.a2[i] = (Cnn[i] - Cee[i]) / d
    self.data.b2[i] = 2 * Cne[i] / d
  else:
    self.data.a2[i] = 0.
    self.data.b2[i] = 0.
self.data.m0 = sum(self.data.Chh) * self.deltaFreq
self.data.m1 = self.data.Chh.dot(self.data.freq) * self.deltaFreq
freq2 = self.data.freq ** 2
self.data.m2 = self.data.Chh.dot(freq2) * self.deltaFreq
```

```
self.data.hm0 = 4 * np.sqrt(self.data.m0)
self.data.tm01 = self.data.am0 / self.data.am1
self.data.tm02 = np.sqrt(self.data.am0 / self.data.am2)
```

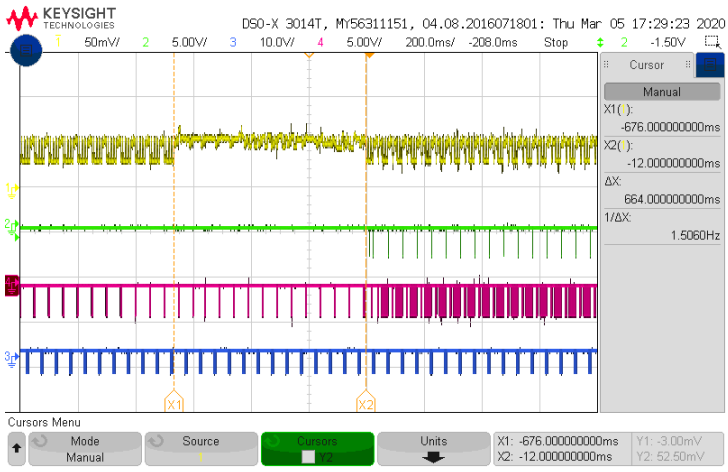
B. Test av bølgesensor

B.1. Test av MCU



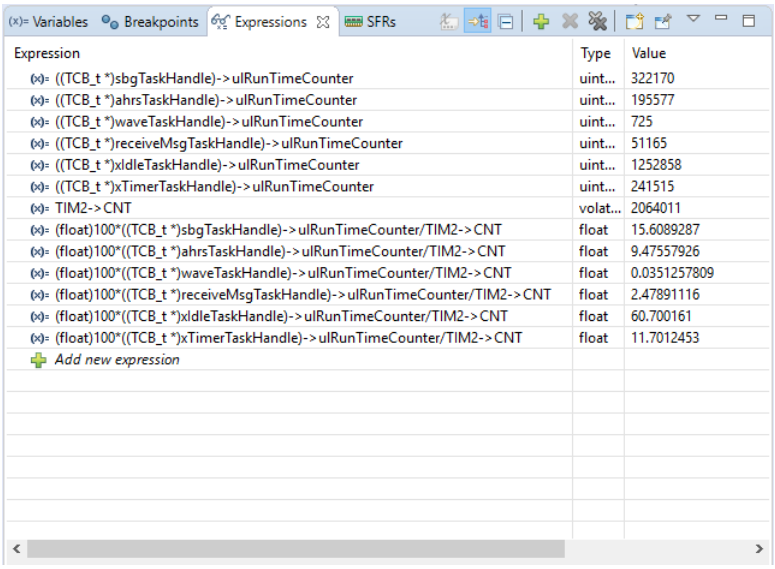
IDD DL_RX DL_TX SBG_RX

Figur B.1.: Overføringstid med -O0

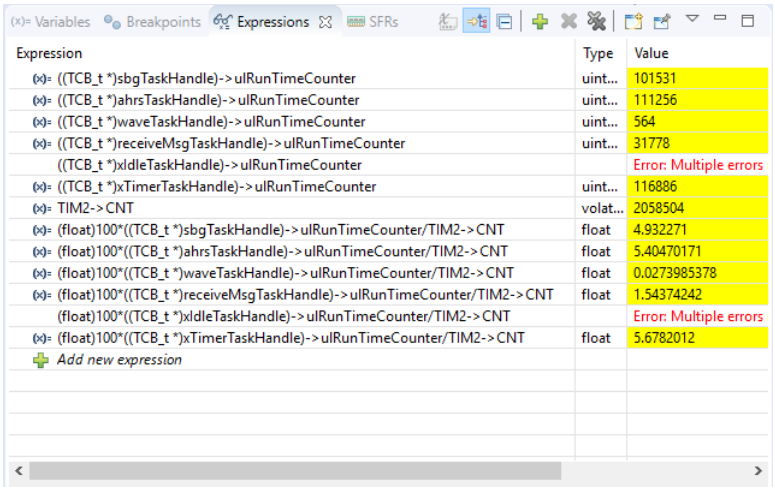


IDD DL_RX DL_TX SBG_RX

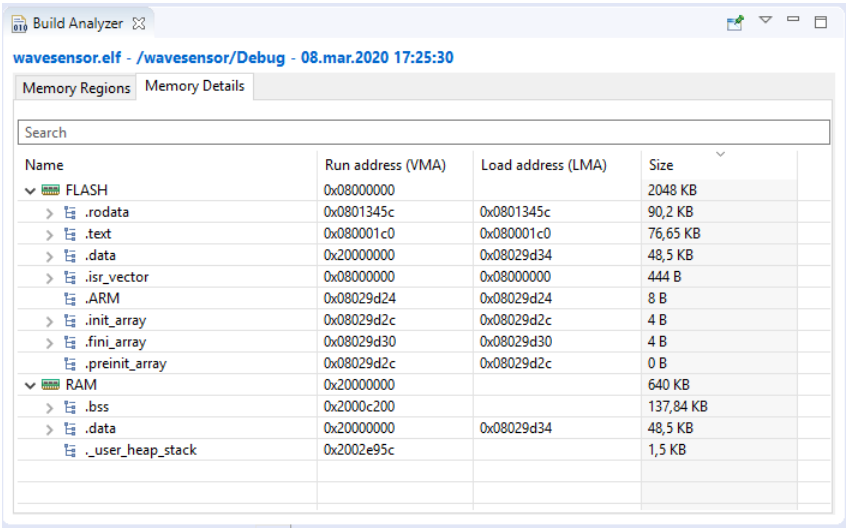
Figur B.2.: Bølgeberegningstid med -O0



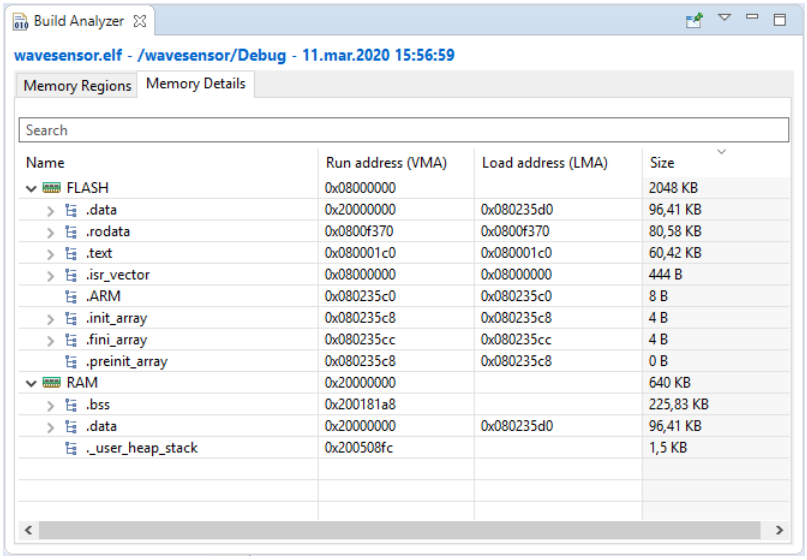
Figur B.3.: Kjøretid for tråder, over to perioder (34 min), med -O0 kompilering, i millisekund og prosent



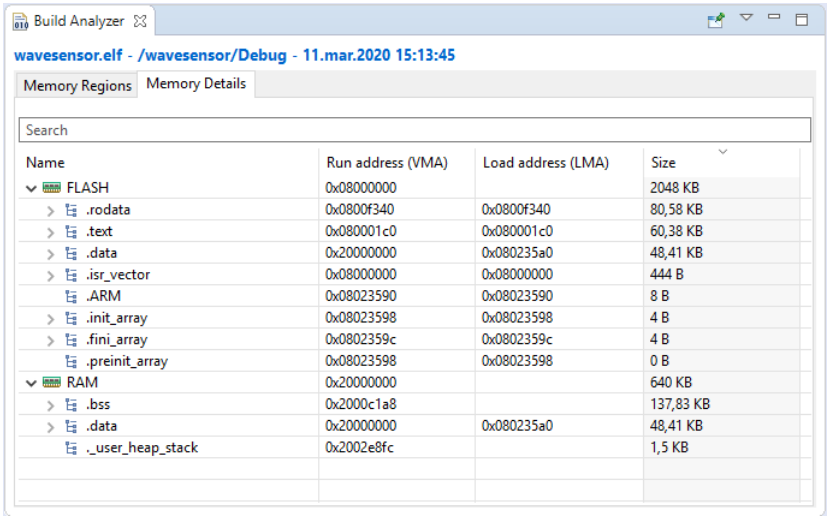
Figur B.4.: Kjøretid for tråder, over to perioder (34 min), med -O3 kompilering, i millisekund og prosent



Figur B.5.: Ram og flash-forbruk, 2048 samples med -O0



Figur B.6.: Ram og flash-forbruk, 4096 samples med -O3 kompilering



Figur B.7.: Ram og flash-forbruk, 2048 samples med -O3 kompilering

B.2. Test på rådata

Akselerometerbias	H_{m0}		T_{m01}		T_{m02}	
acc x-bias = 0.05 m/s^2	0.1 cm	0.0 %	3 ms	0.0 %	3 ms	0.0 %
acc x-bias = 0.15 m/s^2	0.3 cm	0.1 %	8 ms	0.1 %	10 ms	0.1 %
acc x-bias = 0.3 m/s^2	0.6 cm	0.2 %	15 ms	0.2 %	19 ms	0.3 %
acc x-bias = 1 m/s^2	1.6 cm	0.6 %	48 ms	0.6 %	63 ms	0.9 %
acc x-bias = 2 m/s^2	3.3 cm	1.4 %	88 ms	1.0 %	122 ms	1.7 %

Tabell B.1.: Avvik i bølgeparametrene for ulike akselerometerbiaser på x-aksen

Akselerometerbias	Rull		Stamp		Gir		Bølgehøyde	
acc x-bias = 0.05 m/s^2	0.02°	0.3 %	0.29°	9.3 %	0.59°	21.8 %	0.1 cm	0.2 %
acc x-bias = 0.15 m/s^2	0.05°	0.9 %	0.88°	27.9 %	1.70°	63.3 %	0.4 cm	0.7 %
acc x-bias = 0.3 m/s^2	0.10°	1.8 %	1.75°	55.7 %	3.26°	120.9 %	0.9 cm	1.3 %
acc x-bias = 1 m/s^2	0.33°	5.9 %	5.83°	185.2 %	8.98°	331.1 %	2.9 cm	4.4 %
acc x-bias = 2 m/s^2	0.66°	11.9 %	11.56°	366.8 %	14.23°	522.6 %	5.9 cm	8.8 %

Tabell B.2.: Avvik i orientering og bølgehøyde, for ulike akselerometerbiaser på x-aksen

Akselerometerbias	Rull		Stamp		Gir		Bølgehøyde	
acc z-bias = 0.05 m/s^2	0.01°	0.1 %	0.00°	0.1 %	0.02°	0.9 %	0.1 cm	0.2 %
acc z-bias = -0.05 m/s^2	0.01°	0.1 %	0.00°	0.1 %	0.02°	0.9 %	0.1 cm	0.2 %
acc z-bias = 0.3 m/s^2	0.04°	0.7 %	0.02°	0.7 %	0.12°	5.6 %	0.7 cm	1.0 %
acc z-bias = -0.3 m/s^2	0.03°	0.7 %	0.02°	0.6 %	0.11°	5.2 %	0.7 cm	1.0 %
acc z-bias = 1 m/s^2	0.13°	2.7 %	0.08°	2.4 %	0.45°	20.5 %	2.4 cm	3.5 %
acc z-bias = -1 m/s^2	0.11°	2.2 %	0.06°	1.9 %	0.35°	15.9 %	2.3 cm	3.4 %
acc z-bias = 2 m/s^2	0.31°	6.1 %	0.19°	5.5 %	1.04°	47.2 %	4.8 cm	7.0 %

Tabell B.3.: Avvik i orientering og bølgehøyde, for ulike akselerometerbiaser på z-aksen

Akselerometerbias	H_{m0}		T_{m01}		T_{m02}	
acc z-bias = 0.05 m/s^2	0.0 cm	0.0 %	1 ms	0.0 %	1 ms	0.0 %
acc z-bias = -0.05 m/s^2	0.0 cm	0.0 %	1 ms	0.0 %	1 ms	0.0 %
acc z-bias = 0.3 m/s^2	0.2 cm	0.1 %	5 ms	0.1 %	3 ms	0.0 %
acc z-bias = -0.3 m/s^2	0.2 cm	0.1 %	5 ms	0.1 %	4 ms	0.0 %
acc z-bias = 1 m/s^2	0.6 cm	0.2 %	16 ms	0.2 %	12 ms	0.2 %
acc z-bias = -1 m/s^2	0.7 cm	0.3 %	17 ms	0.2 %	13 ms	0.2 %
acc z-bias = 2 m/s^2	1.1 cm	0.4 %	38 ms	0.4 %	27 ms	0.4 %

Tabell B.4.: Avvik i bølgeparametrene for ulike akselerometerbiaser på z-aksen

Bibliografi

[1] ARM. CMSIS-RTOS2. http://www.keil.com/pack/doc/CMSIS/RTOS2/html/index.html. Version 2.1.3.

[2] ARM. CMSIS DSP Software Library. http://www.keil.com/pack/doc/CMSIS/DSP/html/index.html, March 2019. Version 1.6.0.

[3] Astrid H Brodtkorb, Ulrik D Nielsen, and Asgeir J Sørensen. Online wave estimation using vessel motion measurements. *IFAC-PapersOnLine*, 51(29):244–249, 2018.

[4] Torleiv H Bryne, Thor I Fossen, and Tor A Johansen. A virtual vertical reference concept for gnss/ins applications at the sea surface. *IFAC-PapersOnLine*, 48(16):127–133, 2015.

[5] Sara Cheng. IMU Project. https://github.com/sarabear16/IMU_Project, 2017. GitHub repository.

[6] Datawell. Monitoring Software W@ves21. http://www.datawell.nl/Portals/0/Documents/Brochures/datawell_brochure_waves21_b-30-01.pdf, 2012.

[7] Marshall D Earle. Nondirectional and directional wave data analysis procedures. *NDBC Tech. Doc*, 96(002), 1996.

[8] Thor I Fossen. *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons, 2011.

[9] FreeRTOS. Memory barriers in FreeRTOS. https://www.freertos.org/FreeRTOS_Support_Forum_Archive/February_2017/freertos_Memory_barriers_in_FreeRTOS_8049c367j.html, 2017.

[10] J-M Godhaven. Adaptive tuning of heave filter in motion sensor. In *IEEE Oceanic Engineering Society. OCEANS'98. Conference Proceedings (Cat. No. 98CH36259)*, volume 1, pages 174–178. IEEE, 1998.

[11] Thomas Haslwanter. scikit-kinematics. https://github.com/thomas-haslwanter/scikit-kinematics, 2019. GitHub repository.

[12] Minh-Duc Hua, Guillaume Ducard, Tarek Hamel, Robert Mahony, and Konrad Rudin. Implementation of a nonlinear attitude estimator for aerial robotic vehicles. *IEEE Transactions on Control Systems Technology*, 22(1):201–213, 2013. https://ieeexplore.ieee.org/document/6516072.

[13] Felix Kelberlau, Vegar Neshaug, Lasse Lønseth, Tania Bracchi, and Jakob Mann. Taking the motion out of floating lidar: Turbulence intensity estimates with a continuous-wave wind lidar. *Remote Sensing*, 12(5):898, 2020.

[14] Sebastian Küchler, Johannes Karl Eberharter, Karl Langer, Klaus Schneider, and Oliver Sawodny. Heave motion estimation of a vessel using acceleration measurements. *IFAC Proceedings Volumes*, 44(1):14742–14747, 2011.

[15] Michael S Longuet-Higgins. Observations of the directional spectrum of sea waves using the motions of a floating buoy. *Ocean wave spectra*, 1961.

[16] Asle Lygre and Harald E Krogstad. Maximum entropy estimation of the directional distribution in ocean wave spectra. *Journal of Physical Oceanography*, 16(12):2052–2060, 1986.

[17] Sebastian Madgwick. An efficient orientation filter for inertial and inertial/magnetic sensor arrays. *Report x-io and University of Bristol (UK)*, 25:113–118, 2010.

[18] Robert Mahony, Tarek Hamel, and Jean-Michel Pflimlin. Nonlinear complementary filters on the special orthogonal group. *IEEE Transactions on automatic control*, 53(5):1203–1218, 2008.

[19] Tilen Majerle. STM32 UART DMA RX/TX. https://github.com/MaJerle/stm32-usart-uart-dma-rx-tx, 2020. GitHub repository.

[20] Nortek. The Comprehensive Manual. https://www.nortekgroup.com/assets/documents/ComprehensiveManual_Oct2017_compressed.pdf, 2017.

[21] Markus Richter, Klaus Schneider, Dominik Walser, and Oliver Sawodny. Real-time heave motion estimation using adaptive filtering techniques. *IFAC Proceedings Volumes*, 47(3):10119–10125, 2014.

[22] STMicroelectronics. STM32Cube. https://www.st.com/en/embedded-software/stm32cubel4.html.

[23] STMicroelectronics. X-CUBE-MEMS1. https://www.st.com/en/embedded-software/x-cube-mems1.html. Version 7.0.0.

[24] STMicroelectronics. STM32L4Rxxx and STM32L4Sxxx reference manual, December 2019. RM0432 Rev 6.

[25] STMicroelectronics. STM32 Nucleo-144 boards user manual, November 2019. UM2179 Rev 9.

[26] STMicroelectronics. STM32L4R5xx STM32L4R7xx STM32L4R9xx datasheet , January 2020. DS12023 Rev 5.

[27] STMicroelectronics. Description of STM32L4/L4+ HAL and low-layer drivers, February 2020. UM1884 Rev 8.

[28] SBG Systems. Ellipse 2 Micro Series Leaflet. https://www.sbg-systems.com/wp-content/uploads/Ellipse_2_Micro_Series_Leaflet.pdf, September 2018. V1.2.

[29] Malcolm John Tucker and Edward G Pitt. *Waves in ocean engineering*. Elsevier, 2001. Volume 5.

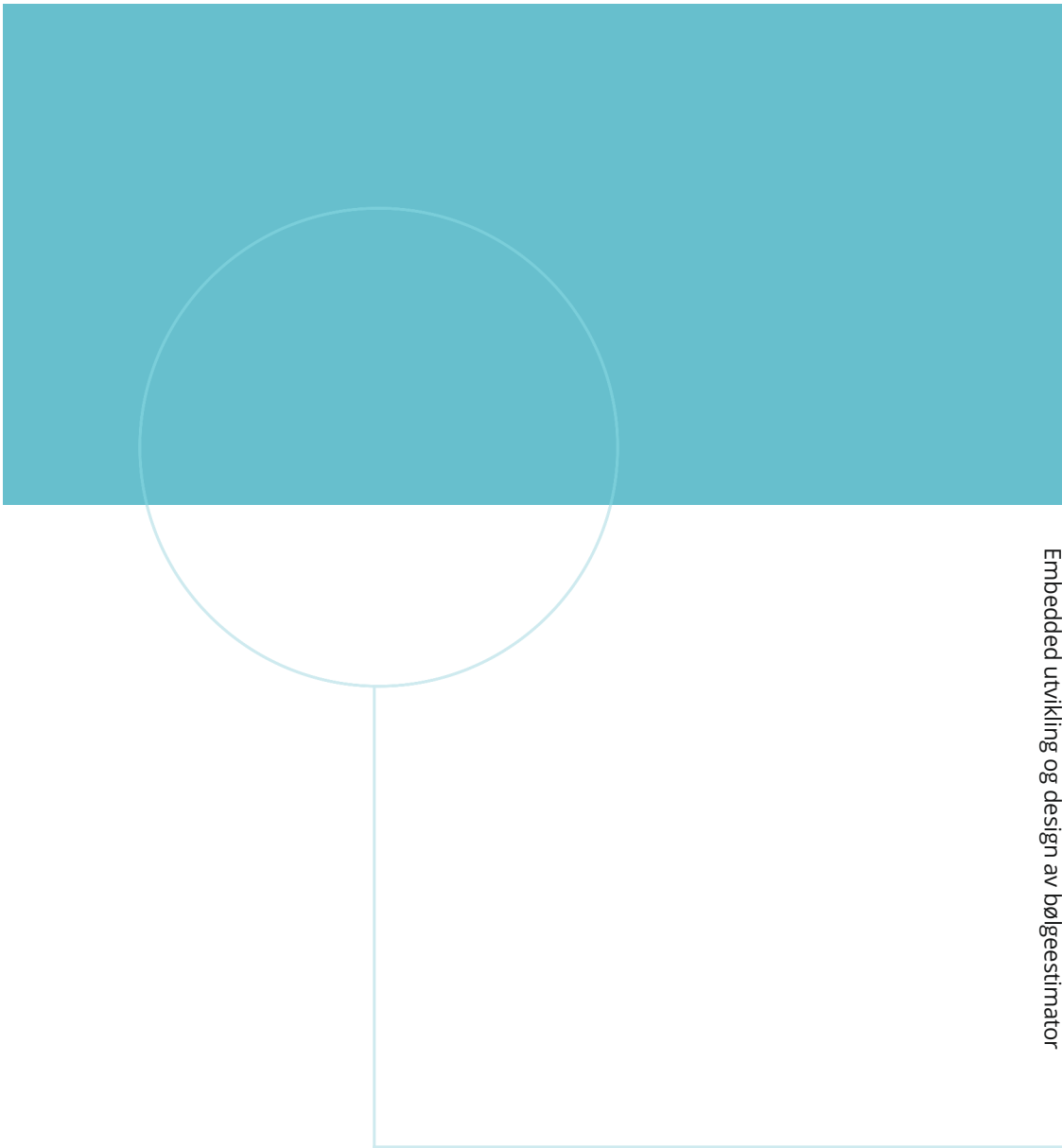
[30] MJ Tucker. The accuracy of wave measurements made with vertical accelerometers. *Deep Sea Research (1953)*, 5(2-4):185–192, 1958.

[31] Wikipedia contributors. Quaternions and spatial rotation — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation, 2019.

[32] Wikipedia contributors. Airy wave theory — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Airy_wave_theory, January 2020.

[33] Wikipedia contributors. Fourier transform — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Fourier_transform, January 2020.

[34] x-io Technologies. Open source IMU and AHRS algorithms. https://x-io.co.uk/open-source-imu-and-ahrs-algorithms/, July 2012.



Embedded utvikling og design av bølgeestimator

NTNU

Kunnskap for en bedre verden