Norges miljø- og biovitenskapelige universitet

Masteroppgave 2017 30 stp
Fakultet for realfag og teknologi

Kombinasjon av romlige data og mobilsensorer for bestemmelse av menneskelig aktivitet

Combining geospatial data and mobile sensors for evaluation of human activity

Eirik Foslie Aabøe
Geomatikk

Sammendrag

Helt siden antikkens Hellas har menneskelig aktivitet og bevegelser vært et tema for forskning. Utviklingen har tatt store steg og metoden har endret seg, fra forskning på menneskets fysiologiske natur til informasjonsalderens bruk av teknologi. Det siste årtusenet har teknologien vokst med stor fart. Datamaskiner, sensorer og programvare har blitt kraftigere, billigere og mer tilgjengelig.

I denne oppgaven benyttes informasjonsteknologi til å analysere menneskelige bevegelsesmønstre. Oppgaven er todelt; del 1 omhandler utviklingen av en applikasjon for å loggføre akselerometer- og geografiske data ved mobiltelefon. I del 2 analyseres innsamlede data.

I analysedelen klassifiseres dataene automatisk til klassene: stasjonær, gange, løping, sykling, kjøring og kollektivtransport. Akselerometerdata, geografiske data og kartdata kombineres ved ulike algoritmer. Egenskaper som fart, utslag til akselerometeret og plassering i miljøet gir gode indikasjoner for hva en bruker foretar seg.

Kombinasjon av geolocation-, akselerometer- og kartdata har vist seg å være overraskende effektivt og gitt resultater over forventning.

Abstract

Human activity and motion has been a subject of research since the ancient Greece. The development has taken great steps and the methods have changed from research of human physiological nature to the use of technology in the age of information. During the last century, techonology has grown rapidly. Computers, sensors and software have become powerful, cheap and available.

In this thesis, information technology is used to analyze human activity patterns. The thesis is separated in two parts; part 1 focuses on development of an application used to collect accelerometer- and trajectorydata with cellphone. In part 2, the collected data is analyzed.

In the phase of the analysis, data is automaticly classified into the following classes: stationary, walking, running, cycling, driving and public transport. Accelerometer-, trajectory- and geospatial data is combined by different algorithms. Characteristics like speed, accelerometer values and location in the enviroment provides good indications of what a user is doing.

Combination of geolocation-, accelerometer- and geospatial data has proven to be surprisingly efficient and yielded results beyond expectations.

Forord

Etter endt studietid representer denne oppgaven finaleopptredenen min som student i geomatikk ved Norges miljø- og biovitenskapelige Universitet (NMBU). Oppgaven er skrevet ved fakultet for realfag og teknologi og ferdigstiller min mastergrad. Graden er gjennomført i løpet av våren 2017 og utgjør 30 studiepoeng.

Takk til Håvard Tveite for veiledning av oppgaven.

Takk til Atle Frenvik Sveen og Alexander Nossum fra Norkart AS for diskusjon og avgjørelse av oppgave og støtte til teknologisk infrastruktur.

En stor takk til Joakim Tveit Husefest og Stian Rostad som startet student-prosjektet Road2zero med en knall idé som la grunnlaget for masteroppgaven. Mange arbeidstimer, godt mot og lange netter førte dessverre ikke til kommersiell suksess, men et uvurderlig vennskap er ei heller å forakte. En takk går også til resten av mine venner og medstudenter på geomatikkprogrammet. Godt humør, lange matpauser og mye tullball har gjort godt i slitsomme perioder. Et bedre studentmiljø skal du lete lenge etter.

En spesiell takk til min familie for støtte og motivasjon i løpet av mine år som student. Ved å skrive disse ordene antar jeg at oppdragelsen har vært vellykket. Takk også for korrekturlesing av oppgaven.

Sist men ikke minst ønsker jeg å takke Atosa for å ha opprettholdt min mentale helse i løpet av denne perioden. Jeg elsker deg!

Innhold

Figurer

Tabeller

Programkode og algoritmer

Kapittel 1

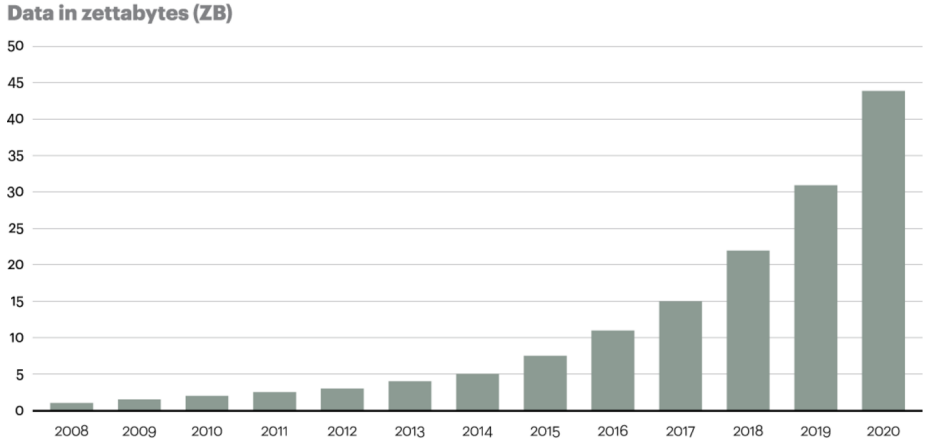
Introduksjon

1.1 Bakgrunn

Smarttelefonenes innmarsj i 2008 har ført med seg en stor verdi av personlige data. Personlige data blir generert fra telefonene, og kan brukes til ulike formål som identifikasjon og anbefalingssystemer. Dagens telefoner med kraftige prosessorer, mye minne, høy lagringskapasitet og kraftige batterier, medfører at komplekse applikasjoner kan kjøres på enhetene. Sensorer har blitt billige, nøyaktige og små, og blir integrert i telefoner [Mafrur et al., 2015]. Dette er sensorer som GNSS-mottaker (globalt navigasjonssatellittsystem), akselerometer, gyroskop, termometer, magnetometer og barometer. Sensorene har flere ulike bruksområder. Ved enkle oppgaver som når en telefon vippes fra stående til liggende posisjon, vil også visningen på skjermen byttes fra stående til liggende. Slik funksjonalitet krever kun et akselerometer. Stedsbaserte tjenester (LBS - Location based services) er mulig med smarttelefonenes posisjonsbestemmelse. Slike tjenester kan benyttes til å blant annet estimere trafikk, gi gode veibeskrivelser og deling av posisjon.

Enorme datamengder blir generert hver dag av mannen i gata, uten at han vet det selv. Store aktører som Google, Apple og Facebook fyller opp datasentere med informasjon og vet mer om deg enn du tror. Datasett som er store og komplekse defineres ofte som "Big data" (stordata). Dataene blir generert med smarttelefoner, sosiale medier, nettlesere, IoT (Internet of things), biler og andre "smarte gjenstander" tilkoblet internett.

Figur 1.1 viser at endringstakten på datamengde er +40% årlig, der datamengden vil utgjøre 45 zettabytes innen 2020. Hver dag blir 2,5 kvintillioner (10^{18}) byte generert, og 90% av verdens data er blitt produsert de siste 2 årene. Wal-Mart i USA er et eksempel på hvordan enorme mengder data blir til. Hver time blir en million kunde-transaksjoner registrert og lagret i databaser. Dataene estimeres til å være 2.5 petabytes i timen. Forskningen og tallene kommer fra en undersøkelse i 2013 [Hagen et al., 2013]. Med stor sikkerhet kan det anslås at veksten siden den tid ikke har vært mindre enn deres estimater.



Figur 1.1: Den globale datamengden vokser med 40% årlig og vil nå 45 zettabytes innen 2020 [Hagen et al., 2013]

Big data har drevet teknologien videre. For å nyttiggjøre seg dataene kreves det teknologi som kan behandle dem. De siste årene har maskinlæring blitt benyttet som teknologiplattform for å levere løsninger ut fra de store datamengdene. Resultatene fra teknologien benyttes for å ytterligere drive applikasjoner som IoT, smarte hus, smarte byer, selvkjørende biler osv til bedre løsninger.

Posisjonsdata utgjør en viktig og stor del av personlig genererte data. Informasjon om hvor brukere er og hva de gjør der kan gi akademiske-, samfunns- og markedsverdier. Dette er hva denne studien prøver å bidra med noen svar til. Masteroppgavens idé kommer fra et studentprosjekt kalt Road2zero. Her var planen å lage en app for automatisk identifisering av transportmidler. Basert på identifiseringen skulle det gis en viss poengsum for klimavennlighet.

Masteroppgaven ser på muligheter ved automatisk identifisering av bevegelsesmønstre, med hjelp av posisjon-, sensor- og romlige data.

1.2 Tidligere arbeid

Det finnes flere studier og produkter basert på posisjon- og sensordata. Apper som Human, Moves, LifeLog, Apple Health, Nike Fuel, Google Maps, Waze med flere leverer forskjellige former produkter basert på bevegelsesmønstre. Disse leverer statistikker som blant annet antall skritt, distanse, sykkeldistanse, løpedistanse, transport-type, forskjellige helse parametere, veibeskrivelser, trafikk. Teknologien som ligger bak appene er proprietære der kildekode er ikke delt. Av løsningen til noen av tjenestene kommer det klart fram at akselerometeret spiller en større rolle. Andre tjenester, som produktene Google leverer, baserer seg i større grad på posisjon. Google gjenkjenner hvor du er og hva du skulle ønske å gjøre der. Basert på slike data kan Google levere anbefalingssystemer, stedsbaserte tjenester og reklame rettet mot individuelle brukere. Samtidig ser Ruter på muligheter innenfor automatiske billeteringssystem, der brukere av kollektivtrafikk skal betale billetter automatisk basert på deres transport. Q-Free har forsket på mulighetene for å bytte ut bomringer med et system der bilister betaler for brukt vei, basert på posisjonsdata[1].

Innen akademiske kretser forskes det også på automatisk identifikasjon av ulike former for bevegelsesmønstre og aktivitet. Innen fagfelt som folkehelse, livskvalitet, epidemiologi, miljø, sport og sensorteknologi foreligger det omfattende studier innen bevegelse, aktivitet og trender.

[1] `https://www.digi.no/artikler/tryggere-bompenger-med-ny-algoritme/197839`

1.3 Problemstilling

Del 1:
Utvikle en applikasjon for innhenting av sensordata fra mobiltelefon.

Del 2:
Utvikle en modul for klassifisering og analyse av menneskelig aktivitet ved kombinasjon av romlige data og mobilsensorer.

Applikasjonen utviklet med React Native og Python-modulen utviklet for analyse, samt innsamlet data, finnes på `https://github.com/eirikaa/Human-Activity-Evaluation`

Kapittel 2

Teori og teknologi

Bevegelsesmønstre

Analyse av menneskelig aktivitet og bevegelse har røtter tilbake i antikkens Hellas, der Aristoteles utviklet en modell av det menneskelige bevegelsesapparatet ved krefter og gravitasjonssenter. Menneskelig bevegelse ifølge [Hamill and Knutzen, 2003], innebærer en endring av en persons plassering, posisjon eller forhold til et punkt i miljøet. [Brooke and Whiting, 1973] utvidet definisjonen ved å si at bevegelse ikke er uniform, men at den har mange sjikt. Det kreves mange synsvinkler for å analysere og identifisere den. Menneskelig bevegelse kan ikke begrenses til det rent fysiske. Bevegelseskonseptet krever at man gjenkjenner den totale konteksten som menneskelig bevegelse er en del av; sosiologisk, miljømessig, psykologisk, mekanisk, fysiologisk og anatomisk er alle faktorer som må bedømmes [Godfrey et al., 2008].

Bevegelsesmønstre vil i denne studien defineres kun til forflytning i miljøet. Denne bevegelsen kan være så mangt, du kan bevege deg ved fysisk aktivitet som gange, sykling og løping, ved transport med kjøretøy eller kollektivtransport. Oppgaven tar for seg analyse og klassifisering av bevegelsesmønstre ved hjelp av informasjonsteknologi.

For å analysere datasettene som produseres av applikasjonen som beskrives i kapittel 3 kreves det definisjon på hva som skal utforskes. Å kunne beskrive typene bevegelse i henhold til de kjente dataene som eksisterer er avgjørende for arbeidet med analysen. Følgende klasser av bevegelsesmønstre benyttes i denne studien:

- Stasjonær
- Gange
- Løping
- Sykling
- Bilkjøring
- Kollektivtransport

Egenskaper til bevegelser

Forskjellige typer bevegelse har både ulike og lignende egenskaper. Egenskapene må beskrives og kvantifiseres for å tallfeste parametere som kan gi grunnlag for analyse.

Ved bevegelser uten transport inngår gange, løping og sykling. Gange kjennetegnes av lav fart, som for de fleste mennesker er lavere enn 6 km/t. Utslaget til akselerometer vil hovedsakelig ligge på et middels nivå som bestemmes til aktivitet. Akselerometerets funksjon, egenskaper og avledninger vil forklares løpet av oppgaven.

Løping beskrives ved middels fart, som generalisert ligger mellom 8 og 16 km/t. Akselerometeretutslaget vil være høyt og klassifiseres som høy aktivitet.

Sykling gjenkjennes ved gjennomsnittlig høy fart, med stor variasjon. Fart ligger mellom 6 og 50 km/t og akselerometerutslaget varier fra lavt til høyt utslag. Variasjonen forekommer ved at nedoverbakker og oppoverbakker skiller mye. Akselerometer vil ikke registrere mye akselerasjon i nedoverbakker der det ikke tråkkes ved sykling.

For bevegelser uten transport varier fart, basert på ulik fysisk form og ulikt formål for aktivitet. Fart og akselerometerutslag generaliseres kraftig i

beskrivelsene av bevegelse. Beskrivelsene av fart er også relativ med hensyn til transportform. Høy fart ved sykling er ikke det samme som høy fart ved bilkjøring.

Bilkjøring og kollektivtransport inngår i bevegelser med transportmidler. Bilkjøring kjennetegnes ved potensiell fart fra 10 til 140 km/t og et lavt akselerometerutslag.

Buss, t-bane, tog og trikk er en del av kollektivtransport. Kollektivtransport kjennetegnes av lignende egenskaper som bil, med varierende fart og lavt akselerometerutslag. Egenskaper som repeterende stopp ved bussholdeplasser og at en reise starter ved en bussholdeplass og avsluttes ved en bussholdeplass er særegent for bussturer. T-bane, trikk og tog har samme egenskap som repeterende stopp ved holdeplasser, men også unik geografisk plassering. Disse transportmidlene benytter ikke veien, som bil og buss, men har egne avgrensete områder for sine bruksområder.

Stasjonære bevegelser vil bety at mennesket er i ro, uten fart.

Utviklingsmiljø

For å utvikle programvare kreves det et utviklingsmiljø/infrastruktur. Utviklingsmiljøet består av "backend" og "frontend". Delen av programvaren som ligger nærmest databasen der dataene er lagret eller skal lagres er backend. Her foregår bakgrunnsprossene som fungerer som en motor i programvaren. Tunge beregningsoperasjoner og databasehåndtering som brukeren av systemet ikke trenger å forholde seg til foregår normalt backend på en server [Granevang, 2017a]. For å samhandle med backend i programvaren kreves det også en frontend. Dette er delen av programvaren som ligger nærmest brukeren. Frontenddelen er det brukeren visuelt kan se på sin klient, og styrer hva som skjer når brukeren samhandler med elementene i applikasjonen. Backend og frontend kommuniserer via et API (Application Programming Interface) [Granevang, 2017b].

2.1 Server

En server benyttes for å tilby tjenester til andre datamaskiner (klienter) i et nettverk. Klienter kan være datamaskiner, telefoner eller andre objekter som har internett tilgjengelig [Liseter, 2017]. I dag finnes det mange muligheter for å sette opp servere som kan tilby tjenester. Egne lokale servere eller servere i "skyen" kan benyttes. Digital Ocean[1] er en amerikansk sky-infrastrukturleverandør med datasentere i hele verden. Hos Digital Ocean kan en kjøpe plass på deres servere og få tildelt en "boks" (droplet), som er en virtuell privat server (VPS). Boksen blir installert med en valgfri Linux-distribusjon, CPU, størrelse på harddisk og overføringshastighet etter behov. Boksen har en bestemt IP-adresse som benyttes ved webløsninger. Slike skytjenester er nyttige for å publisere og utvikle applikasjoner billig for private brukere og mindre bedrifter.

2.2 API

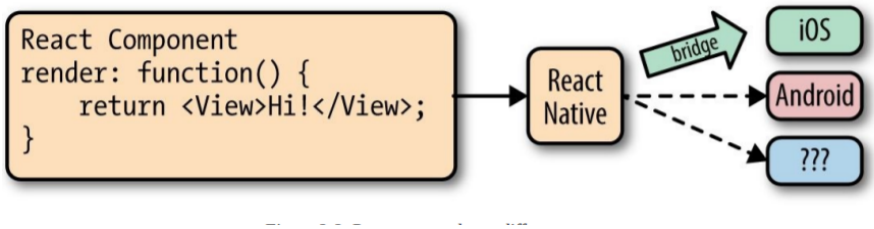
For å utvikle en applikasjon som inneholder backend og frontend kreves det et system for å levere data sømløst mellom database og klient basert på spørringer eller forespørsler. For å gjøre dette kan et API etableres. Dette er systemet som binder backend og frontend sammen. Et API kan etableres med de fleste programmeringsspråk. I Python finnes det flere biblioteker som kan benyttes for å etablere et API, som Django, Bottle, Falcon, Jinja, Flask.

"Flask stands out from other frameworks because it lets developers take the driver's seat and have full creative control of their applications" [Grinberg, 2014]. I andre rammeverk må utvikleren ofte "kjempe" med rammeverket, der det er standard maler for å løse problemer. Dette er ikke tilfellet i Flask[2] der utvikleren har stor frihet til å skreddersy applikasjonen selv. Det er et enkelt web-mikrorammeverk basert på Jinja2[3] og Werkzeug[4] som kan benyttes i store og kraftige applikasjoner [Grinberg, 2014]. I 2016 ble Flask det mest populære Python-rammeverket for web-utvikling ifølge GitHub sine statistikker [GitHub, 2017]. Flask er lett å komme i gang med og gir utvikleren kontroll over applikasjonen. Derfor ble dette rammeverket benyttet i oppgaven.

[1] Digital Ocean `www.digitalocean.com`
[2] Flask `http://flask.pocoo.org/`
[3] Jinja2 `http://jinja.pocoo.org/docs/2.9/`
[4] Werkzeug `http://werkzeug.pocoo.org/`

2.3 Klient

For å bygge appen har rammeverket React Native blitt benyttet. React Native er utviklet av Facebook og bygger på React som er et tilsvarende rammeverk rettet mot web. Målet med React Native er å muliggjøre utvikling av "native" mobile applikasjoner med Javascript kryssplattform, rettet mot både Android og IOS [Eisenman, 2016]. Innen utvikling betyr native at dette er maskinvarens morsmål innenfor programmeringsplattformen. I utgangspunktet er native programmeringsspråk for iPhone's operativsystem (IOS), Swift eller Objective C. For Android er dette Java. I React Native brukes disse språkene for komponentene i appen, mens Javascript benyttes til funksjonaliteten og designet. Det finnes mange standardkomponenter som følger med rammeverket[5]. Tiltross for ungt rammeverk er det etablert et stort og stadig økende miljø der flere komponenter ligger på Github og er klare for bruk. Dette gir altså mulighet til å benytte både native språk, og Javascript for å utvikle applikasjoner. Web-utviklere får en enklere overgang til apputvikling der kjent teknologi kan utnyttes.



Figur 2.1: React Native kan rendre til ulike operativsystem [Eisenman, 2016]

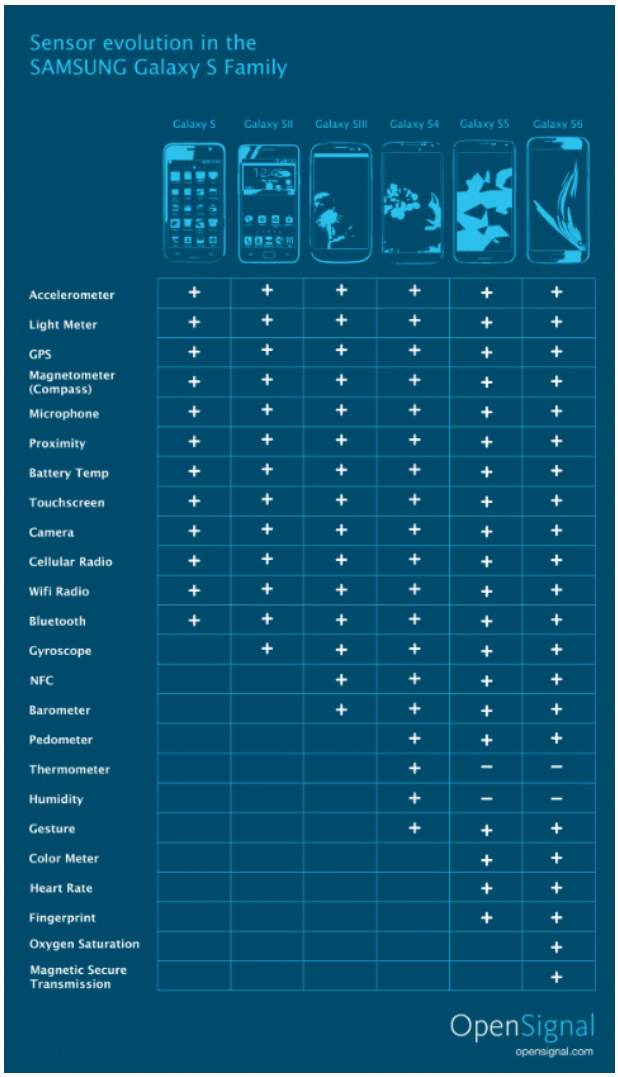
Javascript-versjonen som React og React Native bruker er ES6 (ECMA-Script2015) med JSX. ES6 er den nyeste versjonen i en Javascript-verden som beveger seg veldig raskt i alle mulig forskjellig retninger. JSX er "det nye XHTML", dette fungerer på en tilsvarende måte som HTML, bare at det er integrert i Javascriptet [Bevaqua, 2017].

[5]React Native dokumentasjon `https://facebook.github.io/react-native/docs/getting-started.html`

2.4 Mobilsensorer

Som nevnt blir sensorer billigere, kraftigere og mindre. Moderne smarte mobiltelefoner har mange integrerte sensorer som støtter opp om funksjonaliteten til telefonene. Figur 2.2 viser at utviklingen til Samsungs generasjoner av Galaxy-modellen har økt kraftig. Figuren viser utviklingen fra Samsung Galaxy S utgitt i 2010 til Samsung Galaxy S6 som ble utgitt i 2015. I løpet av tidsperioden har antallet sensorer økt, samtidig som de eksisterende har blitt forbedret med kraftigere ytelse og bedre nøyaktighet. Siden 2015 har det kommet 2 nye telefoner i serien (Galaxy S7 og Galaxy S8) med nye integrerte sensorer.

Akselerometeret som vektlegges i oppgaven er en av grunnbrikkene til funksjonaliteten i moderne telefoner. Denne sensoren har vært essensiell siden den første generasjonen av smarte telefoner kom på markedet. Det samme gjelder GNSS-mottaker, gyrometer, berøringsskjerm med mer. Nye sensorer kommer stadig til, som pulsmåler, barometer, pedometer, fingeravtrykk og mange mange fler.



Figur 2.2: Sensorutvikling i Samsung Galaxy serien [6]

Med React Native finnes det komponenter for å utnytte noen av telefonens mange sensorer, både for Android og IOS. React Native Sensor Manager[7] er et bibliotek der sensorene i Android-telefonene kan brukes i applikasjoner. Biblioteket har støtte for akselerometer, gyroskop, magnetometer, orientering, skritteller, termometer, lyssensor og nærhetssensor. For IOS finnes biblioteket

[6]Sensorutvikling i Samsung Galaxy serien `https://opensignal.com/blog/2016/02/19/sensing-samsung-the-evolution-of-sensors-in-the-galaxy-s-series/`

[7]React Native Sensor Manager `https://github.com/kprimice/react-native-sensor-manager`

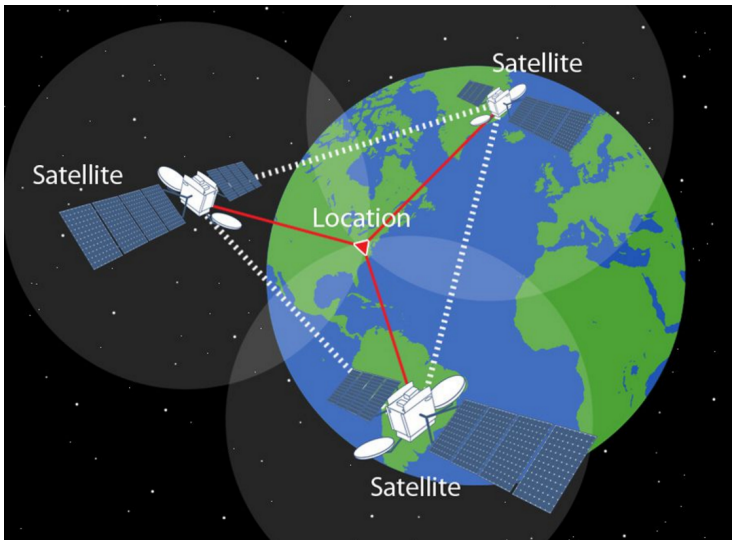
React Native Motion Manager[8]. Dette biblioteket støtter ikke like mange sensorer. Akselerometer, gyroskop og magnetometer er sensorene som er tilgjengelig med denne komponenten. Som nevnt er React Native utviklet for å fungere kryssplattform, men i noen sammenhenger er logikken og innholdet i de forskjellige operativsystemene så ulikt at det kreves individuelle komponenter.

2.5 Globale Navigasjonssatellittsystemer

Globale navigasjonssatellittsystemer (GNSS) er fellesbetegnelsen for satellittbaserte navigasjonssystemer [Forssell, 2017]. De store GNSS-systemene er det amerikanske systemet GPS (Globalt posisjonerings system), det russiske systemet GLONASS, kinesiske Beidou og GALILEO som er et prosjekt fra den europeiske union (EU) og European Space Agency (ESA). Før GALILEO kom på banen var systemene kun styrt av militære organisasjoner.

Formålet med disse systemene er å levere et posisjoneringsystem. Systemene er bygget ulikt, men med identiske formål. GNSS er et enveissystem, grunnet de militære bruksområdene, der deling av mottaker-posisjon ikke er ønskelig. Dette betyr at signalene går en vei, fra satellitt til mottaker. Satellittenes posisjoner må være kjent for at det skal være mulig å bestemme en posisjon på grunnlag av målinger. Ved avstandsmålinger til minimum 3 satellitter kan en bestemme posisjon ved triangulering illustrert ved figur 2.3. Man ønsker gjerne sikt til 4 satellitter for å kunne estimere klokkefeil, som er essensielt for nøyaktig posisjonering [Hofmann-Wellenhof et al., 2008].

[8]React Native Motion Manager `https://github.com/pwmckenna/react-native-motion-manager`

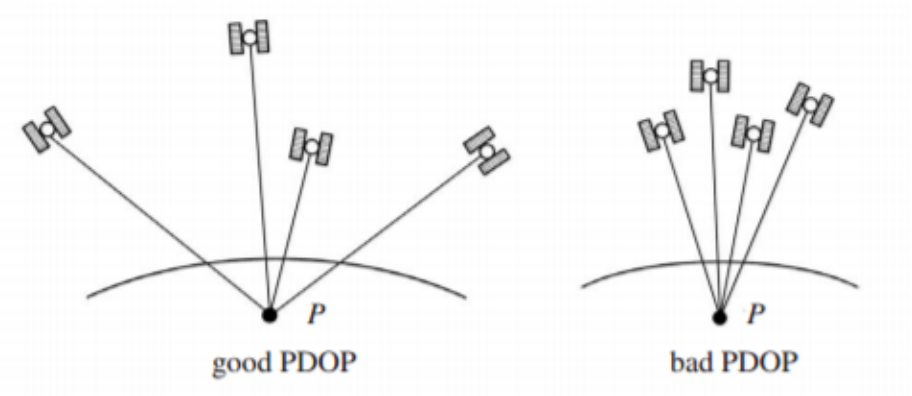


Figur 2.3: Posisjonsbestemmelse ved GNSS [9]

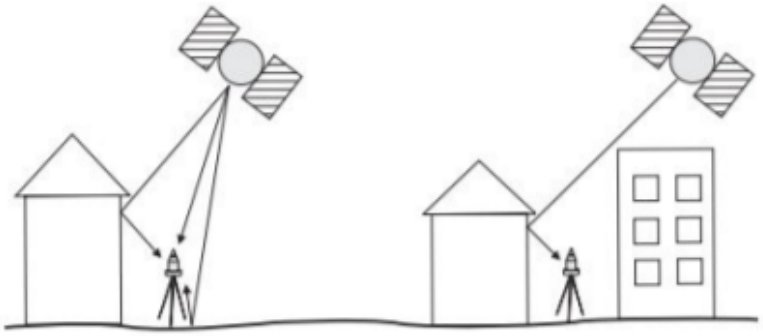
Det finnes mange forskjellig typer GNSS-mottakere. De mest sofistikerte systemene måler på faser og koder fra GNSS-signalene og kan oppnå centimeters nøyaktighet. Vanlige håndholdte GNSS-mottakere måler på koder og har typisk en nøyaktighet på 3 til 5 meter i grunnriss og 5 til 10 meter i høyde ved gode forhold. Dette gjelder for GNSS-mottakere integrert i mobiltelefoner.

[9]Posisjonsbestemmelse ved GNSS `https://www.nationalgeographic.org/topics/gps/`

Flere potensielle feilkilder er knyttet til målinger med GNSS. Signalene er svake og skal reise langt, fra satellitt til mottaker (20200 km) der blant annet atmosfæren forstyrrer signalet. Det kreves en god satellittgeometri som illustrert i figur 2.4 for best mulig presisjon. Den kritiske faktoren i oppgaven og ved håndholdte mottakere som ikke skal ha bedre presisjon enn 3 til 5 meter er "multipath" vist i figur 2.5. Det er kjent at GNSS signalene ikke kan reise gjennom tykke objekter som vegger, trær og lignende, dette betyr at signalene er nesten ubrukelige innendørs. Når signalene treffer faste objekter, kan signalene reflekteres før de ankommer mottaker. I tettbygde områder eller skog kan multipath-effektene gi et utslag, samt sikt til færre satellitter som fører til dårligere satellittgeometri og dårligere presisjon.



Figur 2.4: PDOP geometri [Seeber, 2003]



Figur 2.5: Multipath [Seeber, 2003]

2.6 Geolocation

For mobile applikasjoner er det ofte essensielt å vite en brukers posisjon. For å finne brukerens posisjon benyttes geolocation-API. Geolocation er et web-API for posisjonsbestemmelse i nettlesere og telefoner. Den samme teorien som posisjonering med GNSS benyttes, men flere typer målinger integreres i posisjonsbestemmelsen. Telefonene kan stort sett motta signaler fra GPS og i noen tilfeller GLONASS. Bortsett fra GNSS benytter API'et informasjon fra offentlige IP adresser, mobilmaster, WiFi, MAC ID med mer [Gup, 2017].

React Native har støtte for geolocation[10] som bygger på MDN (Mozilla developer network) web-geolocation-API'et[11]. Ved å benytte geolocation kan en motta informasjon som posisjon (lengde og breddegrad), høyde over havet, nøyaktighet til posisjon, nøyaktighet til høyde, retning, fart og et tidstempel.

API'et har metodene: getCurrentPosition, watchPosition, clearWatch og stopObserving. GetCurrentPosition og watchPosition er metodene som leverer posisjoner. Den viktigste forskjellen er at getCurrentPosition leverer én posisjon ved forespørsel, mens watchPosition gir kontinuerlige posisjonsoppdateringer.

2.7 Akselerasjon

Fysikken beskriver akselerasjon som endring av hastighet til et legeme med hensyn til tid, beskrevet ved formel 2.1. Newtons andre lov 2.2 beskriver akselerasjon som nettoresultat av alle krefter som virker på et legeme. SI-enheten for akselerasjon er meter per sekund i andre (m/s^2) [Tipler and Mosca, 2007].

$$a_x = \frac{dv_x}{dt} = \frac{d^2x}{dt^2} \tag{2.1}$$

$$\vec{F} = m \times \vec{a} \tag{2.2}$$

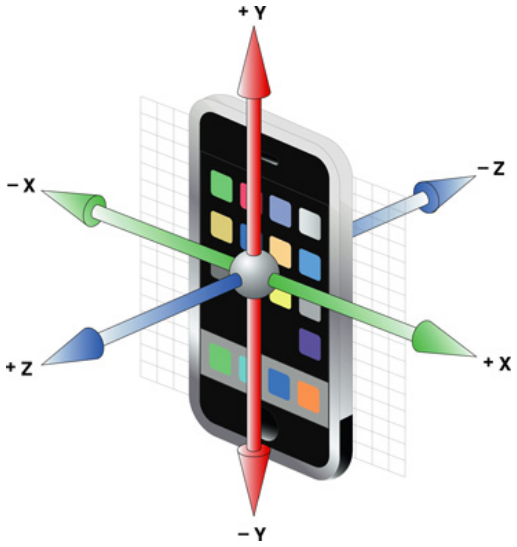
2.7.1 Akselerometer

Et akselerometer er en sensor som måler akselerasjon og krefter indusert av tyngdekraft [Wikipedia, 2017a]. Akselerometeret måler akselerasjon og

[10] React Native dokumentasjon `https://facebook.github.io/ve/docs/geolocation.html`

[11] MDN Geolocataion dokumentasjon `https://developer.mozilla.org/en-US/docs/Web/API/Geolocation`

tyngdekraften den gjennomgår, dette måles typisk i m/s^2 eller g-krefter. Et akselerometer i fritt fall vil derav ikke måle noen akselerasjon. Akselerasjon fra mobile sensorer måler vektorer i 3 akser ($\vec{X}$, $\vec{Y}$, $\vec{Z}$), illustrert ved figur 2.6.



Figur 2.6: Akselerometerets akser [12]

Lineær akselerasjon

Lineære/relativ akselerasjon gir de samme tre vektorene som representerer akselerasjonen langs hver akse. Forskjellen med lineær akselerasjon er at gravitasjon blir ekskludert. For å ekskludere gravitasjonen kan man bruke filtreringsteknikker som lavpassfilter eller høypassfilter [Android-Developers, 2017].

$$lineær\ akselerasjon = akselerasjon - akselerasjon\ fra\ gravitasjon \quad (2.3)$$

I oppgaven er det hensiktsmessig å benytte normen til vektorene: $\vec{X}$, $\vec{Y}$ og $\vec{Z}$ som et alternativ til filtreringsteknikker. Aksenes akselerasjon kan sees bort fra, vi ser på det totale utslaget i akselerasjon. Når telefonen er i ro vil utslaget i akselerasjon tilsvare tyngdekraften $g = 9.81m/s^2$.

[12] Akselerometerets akser `https://www.packtpub.com/books/content/cocos2d-iphone-handling-accelerometer-input-and-detecting-collisions`

$$|X\vec{Y}Z| = \sqrt{\vec{X}^2 + \vec{Y}^2 + \vec{Z}^2} \tag{2.4}$$

Ved å subtrahere g fra normen til vektorene måler vi relative akselerasjoner som ikke er påvirket av tyngdekraften.

$$\begin{aligned} g &= 9.81 m/s^2 \\ |X\vec{Y}Z| &= \sqrt{\vec{X}^2 + \vec{Y}^2 + \vec{Z}^2} - g \end{aligned} \tag{2.5}$$

Akselerometeret har mange bruksområder. Det mest åpenbare er at det brukes innen gravimetri for å bestemme jordas gravitasjonsfelt. Innen navigasjon brukes det for nøyaktig posisjonsbestemmelse som et tillegg til GNSS-målinger, der det inngår i INS (Inertial Navigation System). Akselerasjon er essensielt innen nyere Virtual Reality-teknologier, det brukes som sensor for å måle jordskjelv og akselerometer bidrar til å løse ut en airbag ved bilkollisjon. Akselerometeret er en nyttig sensor innen mange fagområder.

2.8 Database

Som med servere, er det mange muligheter når det gjelder valg av databaser. Det finnes færre når det gjelder romlige databaser som er gratis, men fortsatt mange gode muligheter. Romlige databaser skiller seg fra klassiske relasjonsdatabaser ved at de trenger støtte for abstrakte datatyper. Dette er datatyper som inngår i et GIS der vi jobber med geografiske objekter. De geografiske objektene har tilhørende beskrivende data (egenskapsdata) knyttet til seg med vanlig databaselogikk. Romlige databaser benytter SQL til å gjøre romlige spørringer mot innhold i databasen [Rigaux et al., 2001].

PostgreSQL med utvidelsen PostGIS er utnyttet i oppgaven. PostGIS ble valgt basert på tidligere erfaringer og er åpen kildekode-programvare som har lagt til støtte for geografiske objekter i PostgreSQL sin ORDBMS (objekt-relasjonell databasehåndteringssystem). Til forskjell fra relasjonsbaserte databasehåndteringssystem støtter ORDBMS objekter og klasser.

PostGIS følger "Simple Features" for SQL som er en standard fra både OGC (Open Geospatial Consortium) og ISO 19125. Standarden spesifiserer en lagring- og tilgangmodell for geografiske objekter (punkt, linje, polygon, multipolygon, multilinje osv) [Wikipedia, 2017c].

2.9 Rom-tid GIS

Romlig data er data som har en geometri og vanligvis noen egenskaper som følger med geometrien. Disse dataene går fra 0 til 3 dimensjoner, der punkter er 0-dimensjonale, linjer er 1-dimensjonale og flater er 2-dimensjonale. Flater som inneholder høyder vil bestå av 3 dimensjoner. Dette er hva som inngår i den konvensjonelle vektor-GIS modellen. Ved å legge til en fjerde dimensjon som representerer tid åpnes mange muligheter for å gjøre komplekse analyser. Når vi har geografiske data med en tidsdimensjon er dette romlige temporale data. Med romlige temporale data håndteres og modelleres objekter/fenomen som har ulik status ved ulike tidspunkt. Videre kan en finne ut sammenhenger mellom objekter/fenomen ved å utnytte den temporale informasjonen i analyser. Ved å utnytte temporal informasjon kan sannsynlige hendelser for fremtiden predikeres.

I oppgaven benyttes det temporale data over tidsperioder der data har blitt innsamlet.

2.10 Maskinlæring

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measued by P, improves with experience E" [Mitchell, 1997].

Maskinlæring er en gren av kunstig intelligens (AI - Artificial Intelligence). Hovedfokuset innen maskinlæring baserer seg på automatiserte prosesser som maskiner kan gi bedre svar på enn mennesker. Ved maskinlæring kan en lære maskiner å gjenkjenne komplekse mønstre og gjøre intelligente beslutninger basert på data [Wikipedia, 2017b]. Algoritmene benytter treningsdata for å lære parametere innen applikasjonen som utøves. Maskinlæringsteknikkene benytter nettverk for datastrukturer med tilhørende algoritmer. Nevrale nettverk inngår ofte i kunstig intelligens, disse nettverkene er inspirert av måten nerveceller i en hjerne er organisert. Nettverket vil bestå av dataelementer (nevroner) som mottar og sender parametere til hverandre [Dvergsdal, 2017].

Kapittel 3

Applikasjon

3.1 Backend

3.1.1 Digital Ocean

For å etablere en fungerende applikasjon som kan lagre data som logges, kreves en server. På serveren settes det opp et API for å kommunisere med frontend i applikasjonen. En boks fra Digital Ocean's løsning ble benyttet som server i applikasjon. Serveren ble satt opp med Ubuntu 14.04 (Trusty) distribusjon. Denne serveren kan brukes til mange formål, og i oppgavens sammenheng er den benyttet i henhold til et API der data lagres automatisk. API'et bygger på Flask og apache2 web-server[1].

3.1.2 Flask

Flask-rammeverket kan kjøres fra en Digital Ocean boks. For at web-applikasjonen skal kunne kjøres i bakgrunnen må den settes opp med en web-server. Ved å følge oppskriften fra Digital Ocean[2] settes det opp en fungerende web-server, bygget på Apache som kjører Flask-applikasjonen.

Hensikten med applikasjonen er å logge data. Akselerometerdata og geodata blir skrevet til separate CSV (comma separated values) filer. Dataene blir sendt til Flask-applikasjonen via GET-forespørsler. En GET-forespørsel

[1] Apache `https://httpd.apache.org/`

[2] Sette opp en flask applikasjon på Ubuntu server `https://www.digitalocean.com/community/tutorials/how-to-deploy-a-flask-application-on-an-ubuntu-vps`

forespør data fra en spesifisert ressurs. API'et er åpent på `http://138.68.86.63/`. Med `http://138.68.86.63/storeGNSS` lagres geolocation-data, med parameterne $lat, lon, speed, accuracy, altitude, heading, time, activity$. For å lagre akselerometerdata brukes `http://138.68.86.63/storeAccelerometer` med parameterne $x, y, z, activity$. En vanlig forespørsel for geodata kalles slik: `http://138.68.86.63/storeGNSS?lat=60&lon=10&speed=2&altitude=120&activity=Aktivitet`.

Dataene blir lagret med følgende struktur, akselerometerdata 3.1 og geodata 3.2.

Akselerometerdata

Tabell 3.1: Akselerometerdata

ID,	X,	Y,	Z,	XYZ,	Time,	Activity
1,	-2.7150247097,	4.80695867538,	8.52275943756,	0.344587106903,	1487412913.21,	Aktivitet
2,	-3.3183636665,	4.72555541992,	8.78133296967,	0.699720270981,	1487412913.27,	Aktivitet
3,	-2.9305028915,	4.76865100861,	8.17320632935,	0.096017430898,	1487412913.33,	Ro

XYZ kolonnen representeres av:

$$\begin{aligned} g &= 9.81 m/s^2 \\ |X\vec{Y}Z| &= \sqrt{\vec{x}^2 + \vec{y}^2 + \vec{z}^2} - g \end{aligned} \tag{3.1}$$

Geodata

Tabell 3.2: Geodata

ID,	LAT,	LON,	SPEED,	ACCURACY,	ALTITUDE,	HEADING,	TIME,	ACTIVITY
1,	59.8818309,	10.8292374,	0.2700000107,	13.0,	152.0,	95.30000305,	1487369292.1,	Aktivitet
2,	59.8818519,	10.8290795,	0.9599999785,	13.0,	152.0,	95.30000305,	1487369294.1,	Aktivitet
3,	59.8818381,	10.8291693,	0.0,	11.0,	154.0,	0.0,	1487369297.13,	Ro

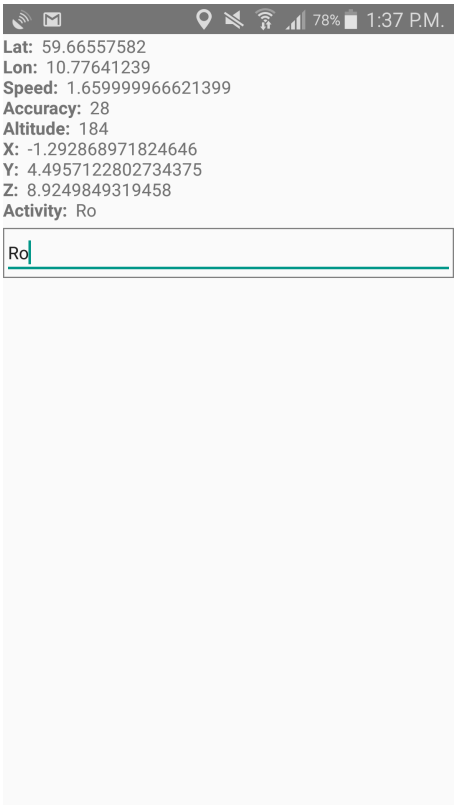
Parameterne API'et tar i mot kommer fra appen som blir forklart i avsnitt 3.2.1.

3.2 Frontend

3.2.1 React Native

En enkel app ble utviklet for datainnsamling med React Native-rammeverket. De essensielle komponentene i applikasjonen er det innebygde Geolocation-API'et, avsnitt 2.6 og sensor biblioteket React Native Sensor Manager, avsnitt 2.4.

Figur 3.1 viser applikasjonens design. Dataene som samles vises fram der, men i teorien trenger ikke selve applikasjonen å vise noe som helst. Derfor har også funksjon blitt vektlagt fremfor design. Tekstfeltet i applikasjonen er viktig da sannhetsverdiene for aktiviteter blir skrevet inn her.



Figur 3.1: Applikasjonen som ble utviklet med React Native

Geolocation

Kode 3.1 viser kallet til geolocation API'et. For at posisjonen skal oppdateres jevnlig benytter metoden watchPosition. Det som er viktig å merke seg er at alternativet "distanceFilter" er satt til 4 i watchPosition. Dette vil si at posisjonen skal oppdateres hver gang applikasjonen får en posisjonsoppdatering som er 4 eller flere meter fra forrige posisjon. 100 meter er "default-verdi" (dette mangler fra dokumentasjonen). Det finnes ikke alternativer til å velge et satt intervall i tidsrommet for posisjonsoppdateringer. Dette ville vært nyttig slik at dataene kunne brukt samme frekvens som akselerometeret. For analyse ville det også vært nyttig med flere oppdateringer for å se på inaktivitet og klyngeeffekter. En mulighet for å levere posisjonsoppdateringer på en bestemt frekvens er å utvikle en tidsbasert funksjon som kaller getCurrentPosition med en bestemt frekvens.

Kode 3.1: Geolocation

```
// Geolocation API
this.watchID = navigator.geolocation.watchPosition((position) =>
↪  {
    this.setState({
        // Update parameters for each geolocation call
        lat: position.coords.longitude,
        lon: position.coords.latitude,
        speed: position.coords.speed,
        accuracy: position.coords.accuracy,
        altitude: position.coords.altitude,
        heading: position.coords.heading
    });
    // Send parameters to server for storage
    this.fetchGeolocation(this.state.lat, this.state.lon,
    ↪  this.state.speed, this.state.accuracy,
    ↪  this.state.altitude, this.state.heading,
    ↪  this.state.activity);
    },
(error) => alert(JSON.stringify(error)),
{enableHighAccuracy: true, timeout: 0, maximumAge: 1000,
↪  distanceFilter:4}
);
```

React Native Sensor Manager

For å benytte de andre sensorene i telefonen benyttes React Native Sensor Manager diskutert i avsnitt 2.4. I dette biblioteket får man tilgang til akselerometer, gyroskop, magnetometer, orientering, skritteller, termometer, lyssensor og nærhetssensor. Kallet av sensorbiblioteket vises i kode 3.2. Av disse sensorene er det akselerometeret som er blitt benyttet i oppgaven. Hastigheten for nye oppdateringer har en frekvens på 3.33 Hz som tilsvarer hvert 300 millisekund for hver akse. Oppdateringfrekvensen kan settes fritt. Frekvensen ble satt høyt for å få med seg utslag fra akselerometer på alle skritt og bevegelser brukeren foretar seg. Det er uvisst om verdiene er integrert over tid, eller om verdien som leveres reflekter den umiddelbare situasjonen.

Kode 3.2: Akselerometerdata

```
import {
    DeviceEventEmitter // react-native-sensor-manager
    } from 'react-native';

//Sensor manager
import { SensorManager } from "NativeModules";

SensorManager.startAccelerometer(300); // Start the accelerometer
 ↪  with a minimum delay of 300 ms between events
DeviceEventEmitter.addListener("Accelerometer", function (data) {
    this.setState({
        // Update parameters for each accelerometer event
        x: data.x,
        y: data.y,
        z: data.z
    });
    // Send parameters to server for storage
    this.fetchAccelerometer(this.state.x, this.state.y,
     ↪  this.state.z, this.state.activity);
}.bind(this));
```

Komponentene for geodata- og akselerometerverdier benytter "fetch" for å forespørre GET-forespørselen fra Flask API'et som vises i kode 3.3. I vanlige GET-forespørsler benytter man responsen fra API'et. I dette tilfellet kreves det kun å sende data via parametere, som lagres på serveren i CSV-filer. Da benyttes URL'en, som diskutert i avsnitt 3.1.2. Fetch-metodene kalles hver gang det oppdateres verdier for geolocation eller akselerometer, og sender da nye parametere til API'et.

Kode 3.3: Fetch forespørsel

```
fetchGeolocation(lat, lon, speed, accuracy, altitude, heading,
↪  activity) {
    // URL for API at IP-address 138.68.86.63
    var URL = "http://138.68.86.63/storeGNSS?lat=" + lat +
    ↪  "&lon=" + lon + "&speed=" + speed + "&accuracy=" +
    ↪  accuracy + "&altitude=" + altitude +  "&heading=" +
    ↪  heading + "&activity=" + activity;

    fetch(URL)
        .catch((error) => {
            console.log("Error fetchGeolocation");
        })
        .done();
}

fetchAccelerometer(x, y, z, activity) {
    // / URL for API at IP-address 138.68.86.63
    var URL = "http://138.68.86.63/storeACCELEROMETER?&x=" + x +
    ↪  "&y=" + y +"&z=" + z + "&activity=" + activity;
    fetch(URL)
        .catch((error) => {
            console.log("Error fetchGeolocation");
        })
        .done();
}
```

3.3 Feltarbeid

Applikasjonen ble benyttet til å samle inn data. Dette ble gjort med Samsung Galaxy S5. Datainnsamlingen har i hovedsak fungert ved å la telefonen ligge i en lomme ved dagligdagse gjøremål. Ved endring av aktivitet, blir telefonen plukket opp og det tastes inn ny sannhetsverdi for aktivitet. Varigheten på loggingen bruker å være mellom 10 minutter og noen timer. Grunnen til at det ikke har blitt logget lenger er at datasettene blir veldig store, både med hensyn til størrelse på fil og tidsbruk i analyse.

Kapittel 4

Analyse

Datasettene som er beskrevet i kapittel 3 benyttes for analyse av bevegelsesmønster. Egenskaper til bevegelsesmønstre er beskrevet i kapittel 2 og oppsummert i tabell 4.1. De ulike egenskapene kvantifiseres til skarpe verdier, men vil i analysen kun gi en forutsetning. Det vil si at de forskjellige bevegelsesmønstrene kan bli klassifisert, selv om de ikke oppfyller alle kriterier. Analysen baseres på sannsynlig bevegelsesmønster.

Tabell 4.1: Kvantifisering av bevegelsesmønstre

Aktivitet Egenskap	Gange	Løping	Sykling	Bil	Kollektivtransport
Fart	$1-6\ km/t$	$5-16\ km/t$	$5-50\ km/t$	$5-150\ km/t$	$5-150\ km/t$
Akselerometer	*Middels*	*Høyt*	*Middels/høyt*	*Lavt*	*Lavt*
Romlig data	*Nei*	*Nei*	*Nei*	*Nei*	*Ja*
Annet	*Nei*	*Nei*	*Nei*	*Nei*	*Repetitive stopp*

4.1 Preprosessering

Innsamlingen av posisjons- og akselerometerdata genererer CSV-filer. Disse dataene kan igjen bli avledet til å gi nyttig informasjon ved analyse. Analysen starter derav med en preprosessering der avledet informasjon legges til i datagrunnlaget. For å behandle dataene har i stor grad Python biblioteket Pandas blitt utnyttet.

Pandas (Panel Data)[1] er et åpent kildekodebibliotek som leverer høy ytelse på

[1]Pandas - Python Data Analysis Library `http://pandas.pydata.org/`

enkle datastrukturer og dataanalyseverktøy for programmeringsspråket Python. I biblioteket jobber man med en dataramme (DataFrame) som kan sammenlignes med et regneark, men med mer funksjonalitet. Ved endt analyse danner datarammen egenskaptabellen til de geografiske objektene. I preposesseringen leses CSV filene inn direkte ved "data_geo = pandas.read_csv(geo_file)" med de originale dataene fra datainnsamlingen. Videre kan nye kolonner legges til ved lister av samme lengde i Python.

"ID", "HUMAN TIME", "TIME DIFF' og "TRUE ACTIVITY NUM" regnes ut og legges til i datarammene for akselerometerdata og geodata. ID er en numerisk identifikasjon til alle rader. Tid blir opprinnelig samlet inn som Unix tid [2]. Unix tid er en lineær skala som er definert som sekunder etter 00:00:00, torsdag 1. Januar 1970, UTC (Coordinated Universal Time), uten skuddsekund. HUMAN TIME representerer tid slik vi er vant til å se det i UTC med riktig tidssone, eks: 2017-03-09 12:58:07.25. Dette gjør det enklere å forholde seg til dataene. I kolonnen TIME DIFF blir tidsdifferansen fra hver oppdatering lagt til, dette korrelerer i utgangspunktet med fart i bevegelsen. TRUE ACTIVITY NUM er en numerisk representasjon av de sanne aktivitetene, der det er laget en ordbok for hva brukeren taster inn. Aktiviteten blir oversatt til å numerisk representere klassene som bestemmes.

For geodata blir fart konvertert fra m/s til km/h der kolonnen SPEED oppdateres. Vinkel mellom hvert datapunkt blir også utregnet med hensyn til HEADING. HEADING representerer retningen brukeren beveger seg, mens ANGLE som legges til i datarammen representerer vinkel mellom datapunkter.

$$angle = heading_i - heading_{i-1} \tag{4.1}$$

For akselerometerdata blir kolonnene magNoG og mag lagt til i datarammen. Under datainnsamlingen blir XYZ kolonnen generert automatisk, der normen regnes ut fra vektorene i akselerometeret som forklart i avsnitt 2.7.1. Kolonnen magNoG gir et lignende resultat, men i stedet for å trekke fra $g = 9.81\ m/s^2$, trekkes den gjennomsnittlige normen fra. Dette kommer av usikkerhet rundt nøyaktigheten til akselerometeret, der $g = 9.81$ ikke nødvendigvis er helt korrekt. Fra denne utregningen legges også kolonnen mag til i datarammen.

$$\begin{aligned} mag &= \Sigma(\sqrt{x^2 + y^2 + z^2}) \\ mean &= mag/N \\ magNoG &= mag - mean \end{aligned} \tag{4.2}$$

[2] Unix tid `https://en.wikipedia.org/wiki/Unix_time`

Ved endt preprosessering vil geodata være ordnet som tabellene 4.2, 4.3 og akselerometer dataene som tabellene 4.4,i 4.5

Tabell 4.2: Oppdatert geodata del 1

ID	LAT	LON	ACCURACY	SPEED	ALTITUDE	HEADING	TIME
0	59.909778	10.806075	8.0	36.648001	136.0	146.399994	1.489061e+09
1	59.909710	10.806119	6.0	31.716002	125.0	149.500000	1.489061e+09
2	59.909613	10.806284	8.0	43.848001	138.0	130.800003	1.489061e+09
3	59.909565	10.806458	9.0	41.327998	136.0	124.400002	1.489061e+09
4	59.909523	10.806610	9.0	41.004001	134.0	119.900002	1.489061e+09
5	59.909479	10.806822	8.0	43.163999	137.0	114.900002	1.489061e+09
6	59.909422	10.807040	7.0	42.804001	134.0	110.199997	1.489061e+09
7	59.909396	10.807247	8.0	39.060001	134.0	105.800003	1.489061e+09
8	59.909381	10.807455	7.0	41.688000	138.0	100.699997	1.489061e+09
9	59.909363	10.807690	7.0	44.351999	138.0	102.599998	1.489061e+09
10	59.909342	10.807922	7.0	44.495999	140.0	97.500000	1.489061e+09

Tabell 4.3: Oppdatert geodata del 2

ACTIVITY	HUMAN TIME	TIME DIFF	ANGLE	TRUE ACTIVITY NUM
t-bane	2017-03-09 12:58:07.250000	1.89	3.100006	6
t-bane	2017-03-09 12:58:09.140000	1.03	18.699997	6
t-bane	2017-03-09 12:58:10.170000	1.09	6.400002	6
t-bane	2017-03-09 12:58:11.260000	0.92	4.500000	6
t-bane	2017-03-09 12:58:12.180000	0.96	5.000000	6
t-bane	2017-03-09 12:58:13.140000	1.01	4.700005	6
t-bane	2017-03-09 12:58:14.150000	1.13	4.399994	6
t-bane	2017-03-09 12:58:15.280000	0.97	5.100006	6
t-bane	2017-03-09 12:58:16.250000	0.96	1.900002	6
t-bane	2017-03-09 12:58:17.210000	0.99	5.099998	6
t-bane	2017-03-09 12:58:18.200000	0.93	1.900002	6

Tabell 4.4: Oppdatert akselerometerdata del 1

ID	X	Y	Z	XYZ	Time
30	3.864242	8.829217	-7.403470	2.343140	1.489060e+09
31	2.868254	5.635352	-4.913500	-1.802094	1.489060e+09
32	2.954445	-0.244807	-5.535993	-3.530199	1.489060e+09
33	3.112462	3.949834	-10.846333	2.145398	1.489060e+09
34	6.560112	4.184466	-6.704363	0.460994	1.489060e+09
35	5.104438	5.252280	-6.876746	0.236460	1.489060e+09
36	5.770026	3.878008	-8.145673	0.899068	1.489060e+09
37	5.937620	3.332130	-5.904700	-0.797569	1.489060e+09
38	5.789180	3.921104	-6.867169	-0.009609	1.489060e+09
39	6.153099	1.938705	-6.493673	-0.656477	1.489060e+09
40	8.226477	2.494160	-4.310162	-0.193699	1.489060e+09

Tabell 4.5: Oppdatert akselerometerdata del 2

Activity	HUMAN TIME	TIME DIFF	magNoG	mag	TRUE ACTIVITY NUM
t-bane	2017-03-09 12:51:13.070	0.34	2.250218	12.153140	6
t-bane	2017-03-09 12:51:13.410	0.23	-1.895015	8.007906	6
t-bane	2017-03-09 12:51:13.640	0.39	-3.623121	6.279801	6
t-bane	2017-03-09 12:51:14.030	0.25	2.052477	11.955398	6
t-bane	2017-03-09 12:51:14.280	0.39	0.368073	10.270994	6
t-bane	2017-03-09 12:51:14.670	0.25	0.143539	10.046460	6
t-bane	2017-03-09 12:51:14.920	0.31	0.806147	10.709068	6
t-bane	2017-03-09 12:51:15.230	0.25	-0.890491	9.012431	6
t-bane	2017-03-09 12:51:15.480	0.34	-0.102530	9.800391	6
t-bane	2017-03-09 12:51:15.820	0.34	-0.749398	9.153523	6
t-bane	2017-03-09 12:51:16.160	0.20	-0.286620	9.616301	6

Pandas har også en utvidelse kalt GeoPandas[3], som retter seg mot romlige data. GeoPandas bidrar til enkel og kraftig prosessering av romlige data i Python. Datatypene fra Pandas utvides til å støtte romlige operasjoner på geometriske typer. De geometriske operasjonene utføres av Shapely[4]. Videre avhenger GeoPandas av Fiona[5] for å lese romlige datafiler. Descartes[6] og matplotlib[7] benyttes for å plotte kart.

Datarammen til Pandas som ble diskutert tidligere blir utvidet til Geodataramme (GeoDataFrame), som gir datasettet romlige egenskaper. I oppgaven benyttes GeoPandas hovedsakelig til å skrive ferdig analysert datasett til en romlig fil, der GeoJSON blir benyttet.

4.2 Akselerometerdata

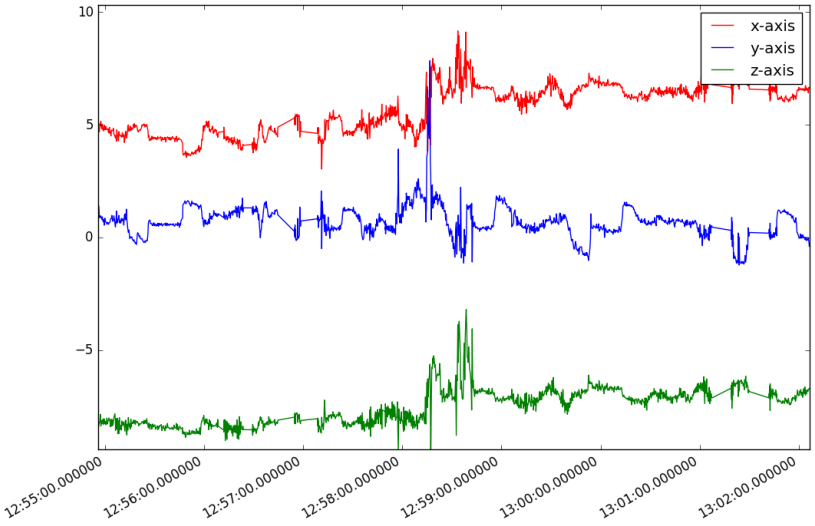
Figur 4.1 viser et plott fra alle aksene i akselerometeret. Plottet viser en tydelig korrelasjon mellom aksene, de oppfører seg likt til tross for forskjellig verdier. Plottet viser verdier fra en busstur, som skal tilsvare lav akselerasjon. Grunnen til at Z-aksen er negativ henger sammen med at mobilen har ligget opp ned i lommen. Noe akselerasjon oppleves, men ikke mye.

I figur 4.2 brukes normen til vektorene fra $\vec{X}$, $\vec{Y}$, $\vec{Z}$. Plottet viser små akselerasjoner, som i figur 4.1. Den gjennomsnittlige normen ligger som forventet rundt $g = 9.81\, m/s^2$.

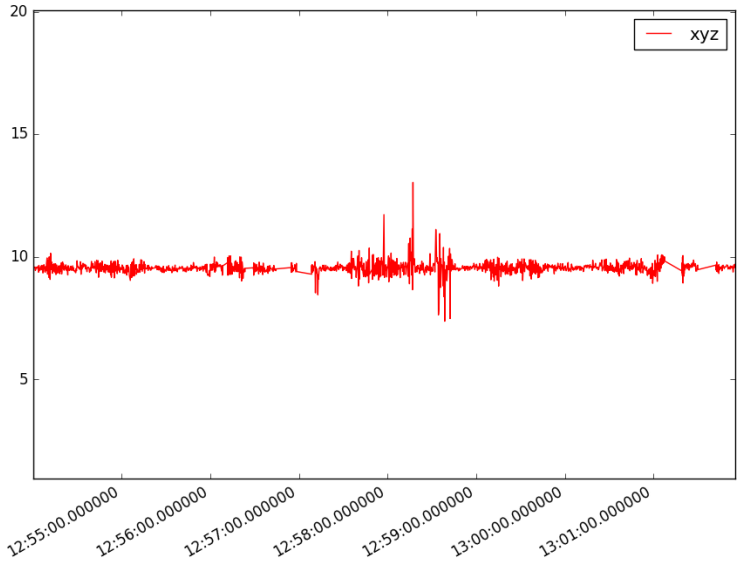
I figur 4.3 er gravitasjonskraften subtrahert fra normen til vektorene $\vec{X}$, $\vec{Y}$, $\vec{Z}$, dette gir oss relative akselerasjoner. Den gjennomsnittlige akselerasjonen ligger nå rundt $0\, m/s^2$.

Figurene representerer de samme dataene med ulike avledninger.

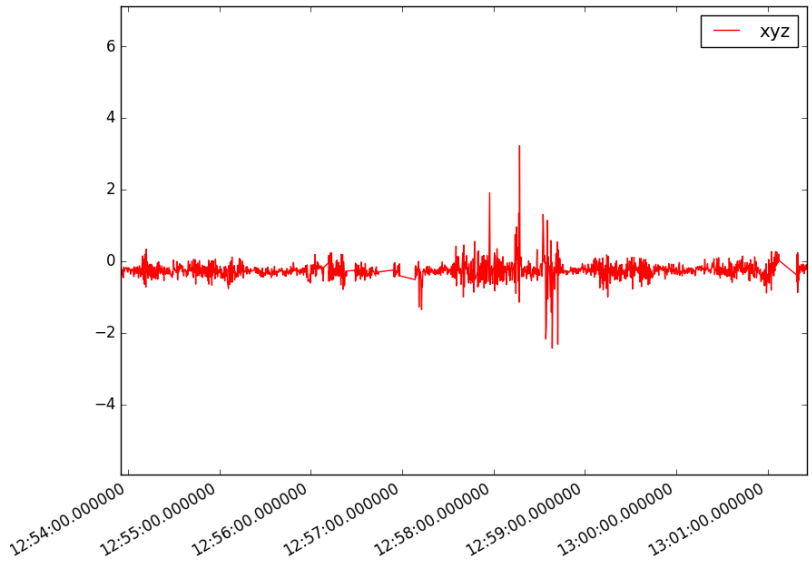
[3]GeoPandas `http://geopandas.org/index.html`
[4]Shapely `http://toblerity.org/shapely/`
[5]Fiona `http://toblerity.org/fiona/`
[6]Descartes `https://pypi.python.org/pypi/descartes`
[7]Matplotlib `http://matplotlib.org/`



Figur 4.1: Akselerometerets 3 akser. Ikke aktivitet



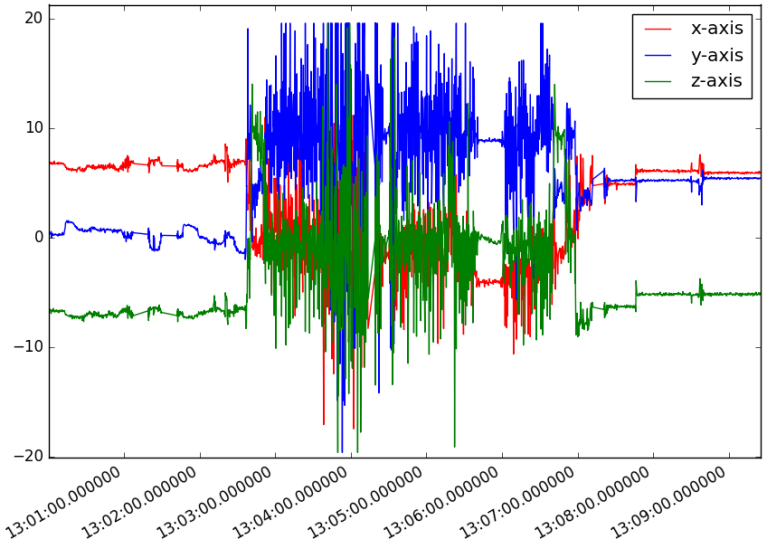
Figur 4.2: Normen til akselerometerets akser. Ikke aktivitet



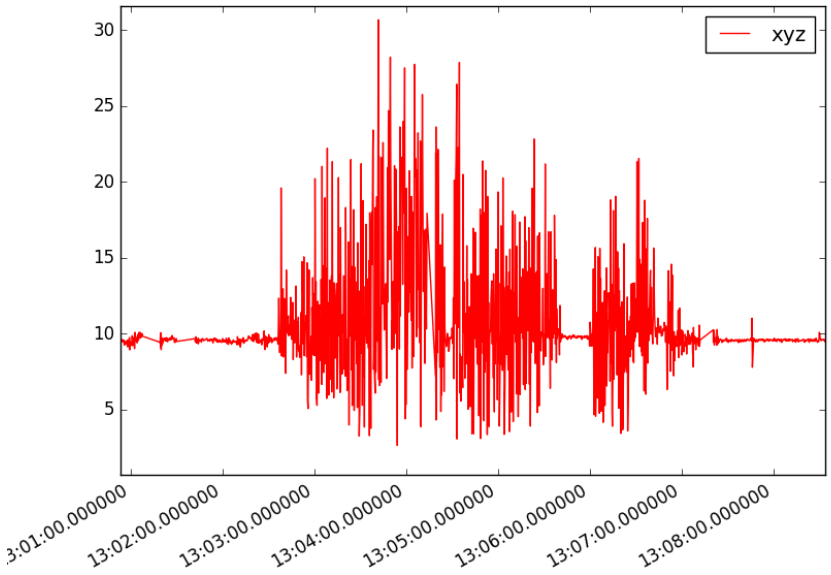
Figur 4.3: Normen til akselerometerets akser, korrigert for tyngdekraft. Ikke aktivitet

Figur 4.4 viser en del av plottet som innebar aktivitet som gange/løping. Dette gir store utslag i målt akselerasjon. I figur 4.5 vises normen til samme del av plottet, der utslagene kommer fra den gjennomsnittlige verdien på 9.81 m/s^2. Figur 4.6 viser igjen det samme, der gjennomsnittlig gravitasjonskraft er trukket fra.

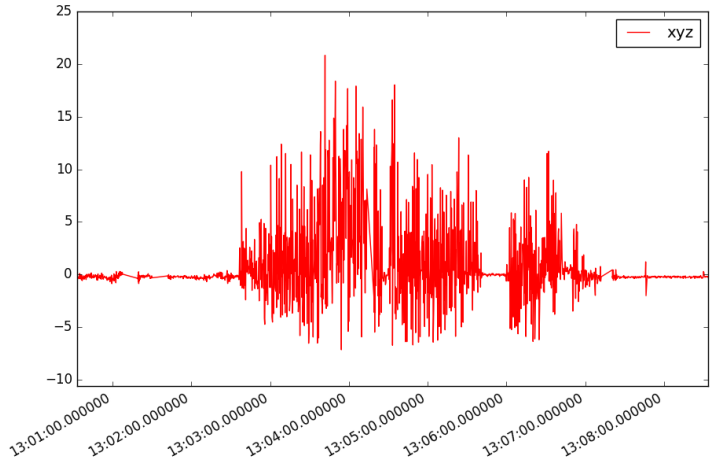
Slike data vil danne grunnlag for bestemmelse av bevegelsesmønstre, der vi kan se på ulike former for aktivitet.



Figur 4.4: Akselerometerets 3 akser ved aktivitet



Figur 4.5: Normen til akselerometerets akser ved aktivitet



Figur 4.6: Normen til akselerometerets akser ved aktivitet, korrigert for tyngdekraft

4.2.1 Kalibrering

Ved å la mobiltelefonen være i vertikal, horisontal og liggende posisjon i en kort tidsperiode ble nøyaktighet akselerometerets undersøkt. Gjennomsnittlig verdi av akselerometerutslaget korrigert for tyngdekraft skal ideelt sett bli 0. Tabell 4.6 viser resultatet for hver akse. X og Y-aksen resulterer i verdier lavere enn 0.1 m/s^2 . Z-aksen har en verdi på 0.4 m/s^2, noe som ikke er ideelt. Hvorfor Z-aksen har en så høy verdi er uvisst. Det kan være grunnet støy i målingen eller dårlig nøyaktighet i akselerometeret. Noe svak nøyaktighet ved akselerometeret tolereres fordi målingene relativt til hverandre gir gode resultater.

Tabell 4.6: Kalibrering

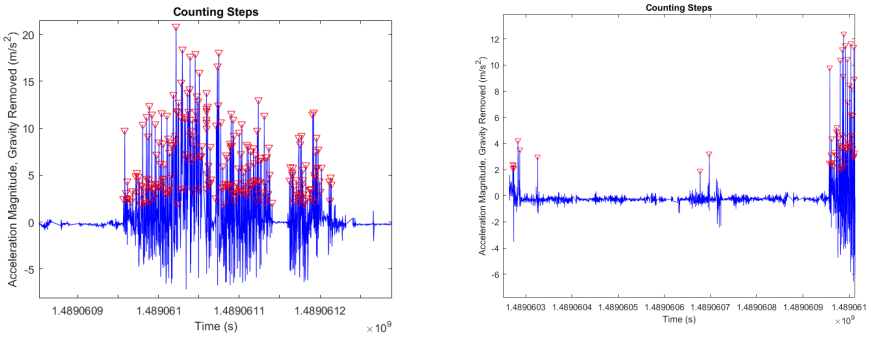
X	Y	Z
-0.048	-0.06	0.407

Kode 4.1: Kalibrering

```
def calibration(self, xyz):
    """
    :param xyz: Norm of accelerometer vectors corrected for
↪   gravitational effects
    :return: average of xyz
    """
    return sum(xyz)/len(xyz)
```

4.2.2 Skritteller

Med metoden "findpeaks" i Matlab, kan en bestemme topper i slike plott akselerometerverdiene generer. Disse toppene skal ideelt sett representere skritt slik det er illustrert i figur 4.7. Dette er kun et estimat, som treffer ganske godt. Ved 500 ekte skritt ligger resultatet på ±50 skritt. Det finnes også muligheter for å gjøre slike analyser i Python.



(a) Aktivitet

(b) Ikke aktivitet

Figur 4.7: Skritteller

4.2.3 Differensieringsalgoritmer

For å gi en mening til akselerometerdataene i analysen må data fra akselerometeret korreleres til geodataene og det må gis et uttrykk for hva dataene faktisk viser. Det ble utviklet en rekke ulike metoder for å analysere akselerometerdataene, men metodene har i bunn og grunn samme logikk. Metodene baserer seg på å se segmenter av dataene, der "diff_range" representerer hvor store segmenter som tas til betraktning. Segmentene er satt som standard til 10, som vil si at det sees på 10 datapunkter fra akselerometeret av gangen. Alle metodene returnerer listen "diff_xyz", som er resultatet av utregningene.

Algoritme 4.2 ser på differansen av maksimumutslaget og minimumutslaget i løpet av segmentet. Disse resultatene legges i listen "diff_xyz". En terskelverdi kan videre gi en bestemmelse om hva som er aktivitet, hard aktivitet og ikke aktivitet. Dette blir diskutert i avsnitt 4.2.4.

Algoritme 4.3 ser på differansen mellom maksimumsutslag og gjennomsnittlig utslag. Videre beregner algoritme 4.4 summen av det totale utslaget til akselerometeret. Algoritme 4.5 legger sammen summen av varians i segmentene,

og returnerer samtidig standardavviket.

Denne praksisen bygger på samme teori som [Mafrur et al., 2015] benytter i sin studie, der summen av varians benyttes for å bestemme aktivitetsnivå. I deres studie bestemmes de samme aktivitetsnivåene som i denne oppgaven (ingen, lav og høy) som beskrives i avsnitt 4.2.4. Forskjellen i studiene er at [Mafrur et al., 2015] ser på på alle akselerometerverdiene separat, mens denne studien benytter segmenter av akselerometerverdier. Videre bestemmes høy aktivitet om summen av varians er høyere enn 10, aktivitet om summen av varians ligger mellom 3 og 10 og ingen aktivitet om verdien er lavere enn 3, i deres studie.

Kode 4.2: Diff_maxmin

```
def diff_maxmin(self, xyz):
    """
    :param xyz: Norm of accelerometer vectors corrected for
↪   gravitational effects
    :return diff_xyz: Differentiated list with values of max
↪   value - min value
    """
    diff_xyz = []

    for value in range(0, len(xyz)-self.diff_range,
    ↪   self.diff_range):
        diff_xyz.append(max(xyz[value:value+self.diff_range]) -
        ↪   min(xyz[value:value+self.diff_range]))
    return diff_xyz
```

Kode 4.3: Diff_avg

```
def diff_avg(self, xyz):
    """
    :param xyz: Norm of accelerometer vectors corrected for
↪   gravitational effects
    :return diff_xyz: Differentiated list with values of max
↪   value - avg value
    """
    diff_xyz = []

    for value in range(0, len(xyz) - self.diff_range,
    ↪   self.diff_range):
        diff_xyz.append(max(abs(xyz[value:value +
        ↪   self.diff_range])) - ((sum(xyz[value:value +
        ↪   self.diff_range]))/self.diff_range)
    return diff_xyz
```

Kode 4.4: Diff_sum

```
def diff_sum(self, xyz):
    """
    :param xyz: Norm of accelerometer vectors corrected for
↪   gravitational effects
    :return diff_xyz: Sum of values in segments of
↪   self.diff_range
    """
    diff_xyz = []
    sum_xyz = 0
    for value in range(len(xyz)):
        sum_xyz += xyz[value]
        if value % self.diff_range == 0 and value != 0:
            diff_xyz.append(sum_xyz)
            sum_xyz = 0

    return diff_xyz
```

Kode 4.5: Diff_sum_variance

```
def diff_sum_variance(self, xyz):
    """
    :param xyz: Norm of accelerometer vectors corrected for
↪   gravitational effects
    :return diff_xyz: Sum of variances in segments of
↪   self.diff_range
    """

    diff_xyz = []
    sum_xyz = 0
    temp_value = []
    variance = []

    for counter in range(len(xyz)):
        sum_xyz += xyz[counter]
        temp_value.append(xyz[counter])
        if counter % self.diff_range == 0 and counter != 0:
            mean = sum_xyz / self.diff_range
            variance.append(sum((mean-value)**2 for value in
            ↪  temp_value) / len(temp_value))
            diff_xyz.append(sum_xyz)
            sum_xyz = 0
            temp_value = []

    std = [math.sqrt(var) for var in variance]

    return variance
```

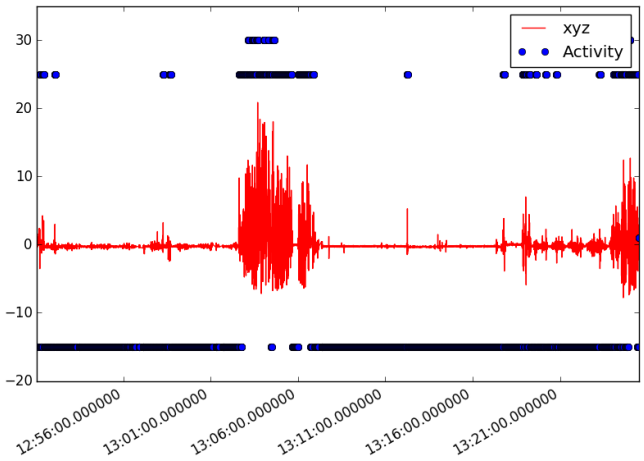
4.2.4 Klassifisering

Resultatet fra metodene i forrige avsnitt 4.2.3 er en liste "diff_xyz" der det ble gjort ulike former for mening fra akselerometerdataene. Listen leveres som input i algoritme 4.6, der tre klasser returnernes av metoden. Enten hard aktivitet, aktivitet eller liten/ingen aktivitet inngår i outputen som er liste "diff_class", i listen inngår også tidspunkt for hendelse. Klassene blir fordelt basert på terskelverdi som avhenger av algoritmevalget fra forrige avsnitt.

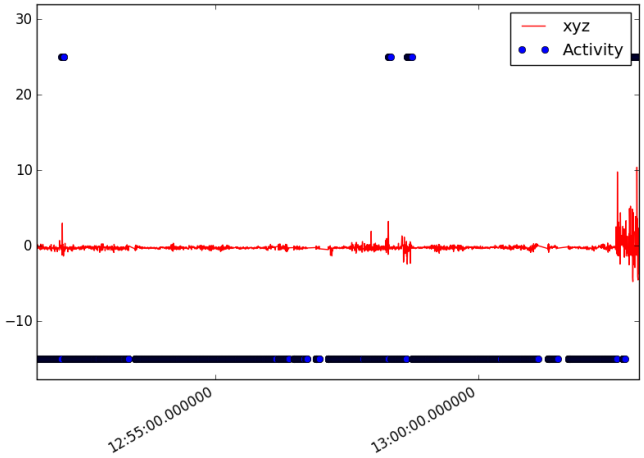
Kode 4.6: Differentiate

```
def differentiate(self, diff_xyz, xyz, time,
↪  activity_threshold=2.5, hard_activity_threshold=10):
    """
    :param diff_xyz: Lists from differentiation algorithms
    :param time: List of timesstamps from accelerometer values
    :param activity_threshold: Threshold for activity
    :param hard_activity_threshold: Threshold for hard activity
    :return diff class: Classes with parameters of activity
    """

    activity = 25
    hard_activity = 30
    low_activity = -15
    diff_class = []

    counter = -1
    for diff in diff_xyz:
        if diff >= hard_activity_threshold:
            for i in range(self.diff_range):
               diff_class.append([hard_activity, time[counter]])
               counter += 1

        elif diff >= activity_threshold:
            for i in range(self.diff_range):
                diff_class.append([activity, time[counter]])
                counter += 1
        else:
            for i in range(self.diff_range):
                diff_class.append([low_activity, time[counter]])
                counter += 1

    # Remaining values
    rest = len(time) - len(diff_class)
    for _ in range(rest):
        diff_class.append([1, 0])

    return diff_class
```

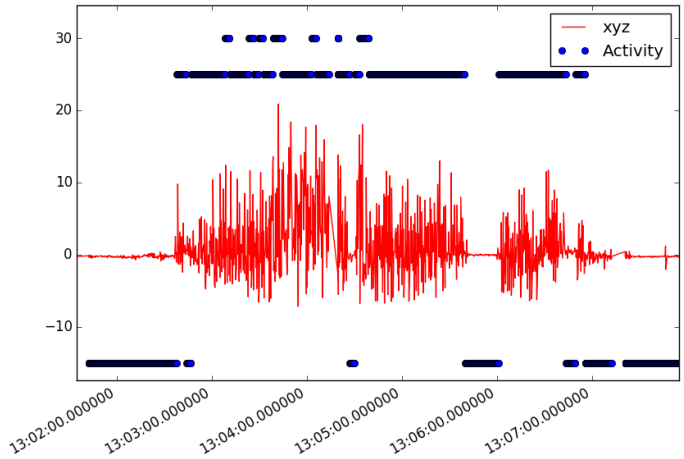
Klassifiseringen representerer nå en enkel bevegelsesanalyse, kun basert på akselerometerverdier. Dette kan beskrive noe om av hva slags type aktivitet brukeren har bedrevet. Figurene 4.8 4.9 4.10 4.11 illustrerer dette i plottet ved prikker, der verdi -15 representerer ikke aktivitet, 25 representerer aktivitet og 30 representerer høy aktivitet. I figur 4.8 4.9 4.10 er algoritme 4.3, "diff_avg()", benyttet med terskelverdi for aktivitet på 4 og hard aktivitet på 13. Figur 4.11 viser aktivitetsdelen med algoritme "diff_maxmin()" 4.2. En høyere andel blir klassifisert som hard aktivitet i siste plott på grunn av at en annen differensieringsalgoritme er brukt med like terskelverdier. Ved å se på differansen mellom minimum og maksimumutslag vil en høyere andel datapunkter bli klassifisert som hard aktivitet. Differensieringsalgoritmene viser stort sett det samme, men krever ulike terskelverdier.



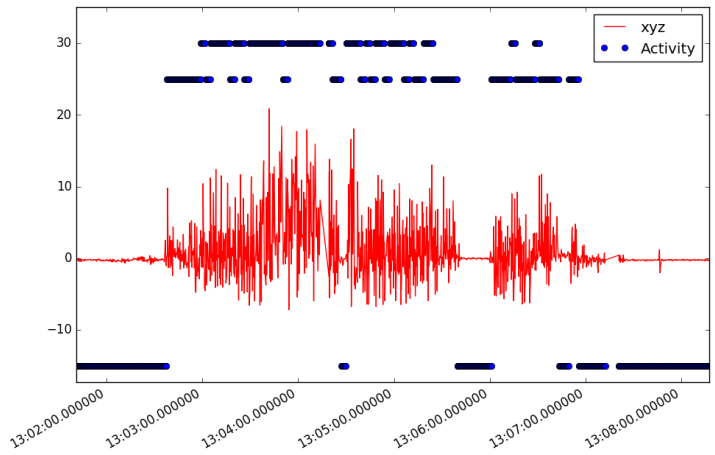
Figur 4.8: Aktivitetsanalyse med diff_avg()



Figur 4.9: Aktivitetsanalyse med diff_avg() i ro



Figur 4.10: Aktivitetsanalyse med diff_avg(), i aktivitet



Figur 4.11: Aktivitetsanalyse med diff_maxmin() i aktivitet

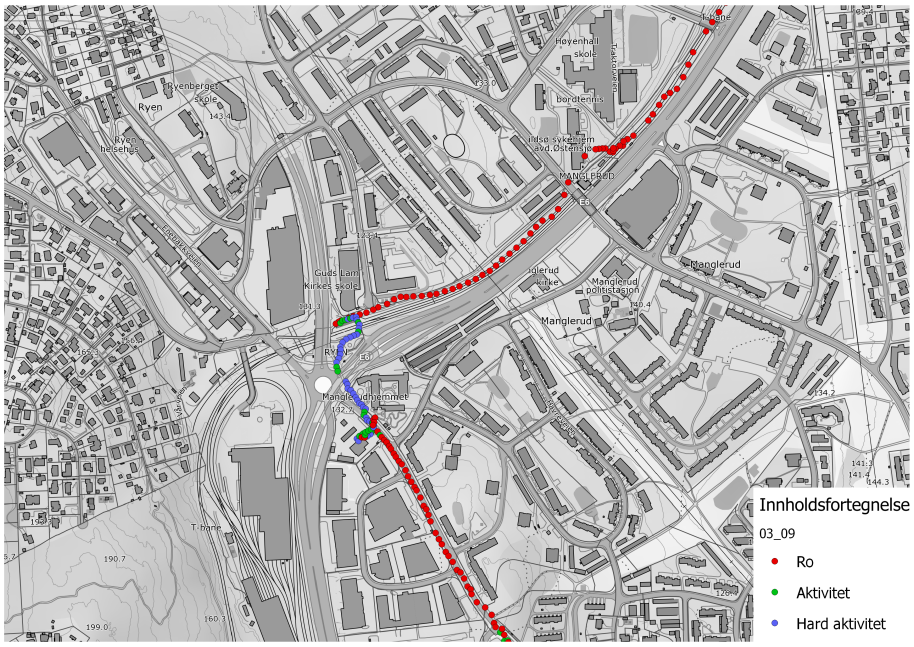
4.2.5 Interpolering av akselerometerdata

Når akselerometerdataene er tilordnet en klasse må dette samordnes med geodataene. Akselerometerdataene har en mye høyere frekvens ved datainnsamlingen enn geodataene. Derfor kreves det en interpolasjon for å tilegne klasseverdiene til posisjons dataene. Listen "diff_class" returneres fra algoritme 4.6 der tidsinformasjon og klasse er bestemt. Denne informasjonen benyttes videre ved å tilegne det til nærmeste datapunkt blant posisjonsdata i tidsrommet. Hvert datapunkt i posisjonsdataene tilegner seg flere verdier fra akselerometer-klassen fordi frekvensen for innsamling av akselerometerdata var høyere. Det er altså flere akselerometerverdier enn posisjonsdata. Av de korrelerte klasseverdiene som tilegnes posisjonsdataene blir verdien det telles flest av lagt til i datarammen for geodata. Vanligvis vil dette være et homogent utvalg, som gjør interpolasjonen akseptabel.

Kode 4.7: Interpolering av akselerometerdata

```
def classify_accelero(self, diff_class, time_geo, data_accelero):
    """
    :param diff_class: Classes with parameters of activity
    :param time_geo: Timestamps correlating with geolocation data
    :param data_accelero: Accelerometer data
    """

    diff_class_geo_time = []
    time_geo = list(time_geo)

    for i in range(len(data_accelero)):
        # Accelerometer timestamps which corresponds to nearest
        ↪  timestamp from geolocation data
        accelero_to_geo_time = min(time_geo, key=lambda x:
        ↪  abs(x-diff_class[i][1]))
        diff_class_geo_time.append([diff_class[i][0],
        ↪  accelero_to_geo_time])


    temp_diff_list = []
    diff_list = []

    for j in range(1, len(diff_class_geo_time)):
        temp_time = diff_class_geo_time[j][1]
        if temp_time == diff_class_geo_time[j - 1][1]:
            # Add diff classes which correspond with geolocation
            ↪  timestamp to list
            temp_diff_list.append(diff_class_geo_time[j][0])
        else:
            if len(temp_diff_list) > 0:
                # Add the diff class which has the highest count
                ↪  in temp_diff_list
                diff_list.append(max(set(temp_diff_list),
                ↪  key=temp_diff_list.count))
            else:
                diff_list.append(1)
            # Empty list
            temp_diff_list = []

    # Remaining values
    rem = len(self.data_geo) - len(diff_list)
    if rem > 0:
        for _ in range(rem):
            diff_list.append(1)
    elif rem < 0:
        diff_list.pop()

    # Create diff class in the dataframe for geolocation data
    self.data_geo["DIFF CLASS"] = diff_list
```

Figur 4.12 vises et kart kategorisert med hensyn til akselerometerklassene. Nord i kartet har brukeren tatt t-bane, videre har bruker gått/løpt til påfølgende busstur. I seg selv representerer akselerometerklassene godt endringene i aktivitet.



Figur 4.12: Aktivitetsanalyse

4.3 Romlige data

For å gi videre mening til de nå akselerometerklassifiserte geodataene integreres det romlige data til analysen. Ved å benytte geografisk informasjon kan en tilegne mer informasjon til datapunktene basert på hvor en bruker befinner seg. Med informasjon om hvor brukeren befinner seg kan det også benyttes som et utgangspunkt til å si noe om hva brukeren gjør ved bestemte steder. Dette er kalt "geo-fencing" som kan sees på som en virtuell avgrensning[8]. De geografiske objektene som relateres til oppgaven er områder knyttet til kollektivtransport.

[8]Geo-fence `https://en.wikipedia.org/wiki/Geo-fence`

4.3.1 PostGIS

For å behandle romlige data er PostGreSQL[9] med PostGIS[10] utvidelsen benyttet. For å fylle tabellene med data er GDAL (Geospatial Data Abstraction Library)[11] verktøyet ogr2ogr[12] benyttet. GDAL er et bibliotek for å behandle ulike romlige filformater og kan lese og skrive til de fleste kjente formater.

Romlig data er hentet fra OSM (OpenStreetMap)[13] ved overpass-turbo API'et[14]. Dataene blir eksportert som GeoJSON filer før de benyttes til å fylle databasen.

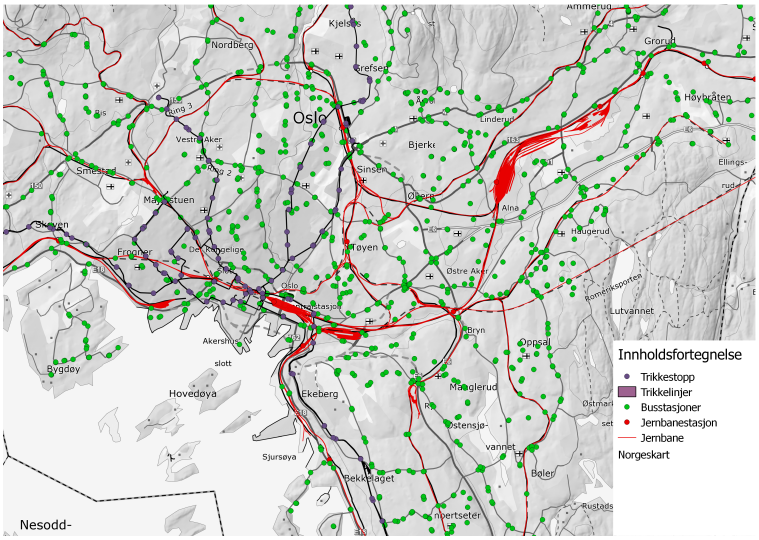
Følgende data benyttes videre i analysen:

- Busstasjoner
- Trikkestasjoner
- T-banestasjoner
- Togstasjoner
- Toglinjer
- T-banelinjer
- Trikkelinjer

Helst skulle bussruter vært en del av datagrunnlaget, men dette er ikke offentlig tilgjengelig enda. Mye av dataene er kun hentet inn fra i Oslo og omegn, da dette har vært området til datagrunnlaget fra datainnsamlingen.

[9] PostGreSQL `https://www.postgresql.org/`
[10] PostGIS `http://postgis.net/`
[11] GDAL `http://www.gdal.org/`
[12] ogr2ogr `http://www.gdal.org/ogr2ogr.html`
[13] OpenStreetMap `https://www.openstreetmap.org`
[14] Overpass Turbo `https://overpass-turbo.eu/`

Figur 4.13 viser et oversiktskart over de romlige dataene som benyttes i Oslo-området.



Figur 4.13: Romlige data

4.3.2 Interaksjon med romlige data

For å interagere med de romlige dataene lagret i PostGIS databasen benyttes psycopg2[15]. Psycopg2 er det mest brukte Python-biblioteket for å behandle PostGIS-databaser. Med dette biblioteket gjøres det SQL-spørringer mot databasen basert på posisjon (lengde- og breddegrader).

Spørring 4.8 returner avstand til nærmeste busstasjon, samt et boolsk utrykk for om brukeren er innenfor en buffer på 10 meter. Spørring 4.9 returner nærmeste tog/t-banelinje og også her om brukeren er innenfor buffersonen på 7 meter. I dette eksempelet beregnes den korteste avstanden til linjestykker. Spørringen benytter k-nærmeste nabo (KNN) med "ST_Distance" basert på geodetiske avstander ved datatypen "::geography". Videre kan det sees på om punktet er innenfor bufferen ("ST_BUFFER") med "ST_DWithin". Med "ORDER BY wkb_geometry $<-> $ ST_POINT(lon, lat) LIMIT 100" blir de 100 best kvalifiserte linjestykkene evaluert. Deretter tas avstanden for å finne nærmeste linjestykke.

[15]psycopg2 `http://initd.org/psycopg/`

Kode 4.8: Busstasjoner

```
WITH index_query AS (
SELECT ST_Distance(
wkb_geometry ::geography,
ST_POINT(lon, lat) ::geography) AS distance,
ST_DWithin(ST_BUFFER(wkb_geometry ::geography, 10), ST_POINT(lon,
↪  lat)  ::geography, 10) AS t, bus AS type, ogc_fid AS id
FROM n50.stasjoner2
ORDER BY wkb_geometry <-> ST_POINT(lon, lat) LIMIT 100
)
SELECT distance, id, t, type FROM index_query ORDER BY distance
↪  LITMIT 1
```

Kode 4.9: Kollektivlinjer

```
WITH index_query AS (
SELECT ST_Distance(
geom ::geography,
ST_POINT(lon, lat) ::geography) AS distance,
ST_DWithin(ST_BUFFER(geom ::geography, 7), ST_POINT(lon, lat)
↪  ::geography, 10) AS t, type AS type
FROM n50.jernbane
ORDER by geom <-> ST_POINT(lon, lat) LIMIT 100
)
SELECT distance, t, type FROM index_query ORDER BY distance LIMIT
↪  1
```

Med kode 4.10 og kode 4.11 kalles metodene som utfører SQL-spørringene. Datarammen oppdateres med kolonnene "STOPS" og "TRANSPORT" illustrert i tabell 4.7, som gir svar på om brukeren befinner seg ved busstopp og innenfor kollektivtransportlinjer. "Train" representerer både t-bane, tog og trikk.

Kode 4.10: Tog, t-bane, trikkelinjer

```
def public_transport(self):
    """
    Updates dataframe with a boolean operator, describing if a
↪   user is inside or outside of a buffer around public
↪   transport-lines
    """
    activity = []

    for i in range(len(self.data_geo)):
        if Classification.read_api_train(self.data_geo["LAT"][i],
        ↪  self.data_geo["LON"][i])[1]:
            activity.append("Train")
        else:
            activity.append("Not train")
    self.data_geo["TRANSPORT"] = activity
```

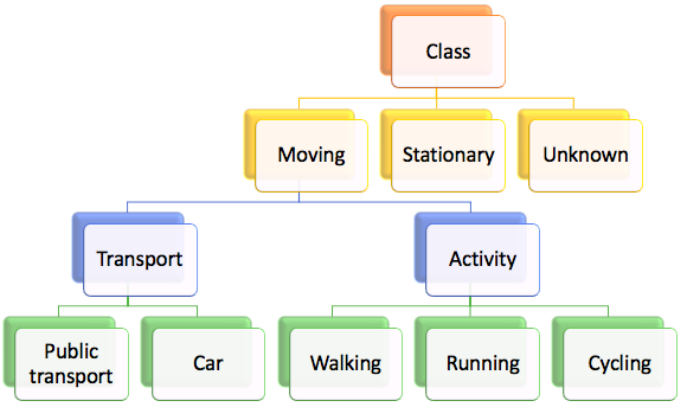
Kode 4.11: Busstopp

```
def stops(self):
    """
    Updates dataframe with a boolean operator, describing if a
↪   user is inside or outside of a buffer around bus stops
    """
    stops = []

    for i in range(len(self.data_geo)):
        req =
        ↪  (Classification.read_api_bus(self.data_geo["LAT"][i],
        ↪  self.data_geo["LON"][i]))
        stops.append(req[2])
    self.data_geo["STOPS"] = stops
```

Tabell 4.7: Oppdatert dataramme for geodata

ID	STOPS	TRANSPORT
210	False	Train
211	False	Train
212	False	Train
213	False	Train
214	False	Train
215	False	Train
216	False	Train
217	False	Train
218	False	Train
219	True	Train
220	True	Not train
221	True	Not train
222	False	Not train
223	False	Not train
224	False	Not train
225	False	Not train
226	False	Not train
227	False	Not train
228	False	Not train
229	False	Not train

4.4 Kombinasjon av data



Figur 4.14: Kombinasjon av data for klassifisering

For å klassifisere dataene ble en metode utviklet der klassifiseringen skjer gradvis beskrevet fra figur 4.14. Første steg er å skille "moving" fra "stationary". Dette er hovedsakelig basert på fart og delvis akselerometer utslag. Hvis farten er målt til å være over 5 km/t blir punktet uansett klassifisert til bevegelse. Ved lavere fart legges det inn en viss usikkerhet, men ved stor sannsynlighet vil all aktivitet målt til over 2 km/t regnes som "moving". Grunnen til at punkter der telefonen har vært i ro, kan registrere en fart er at nøyaktigheten på målingen kan være dårlig. Derav vil registreringene "hoppe" litt rundt i området som medfører at det også registreres en fart. Hvis man i denne situasjonen sitter i ro og det ikke er noen utslag fra akselerometeret vil punktet registreres som "stationary".

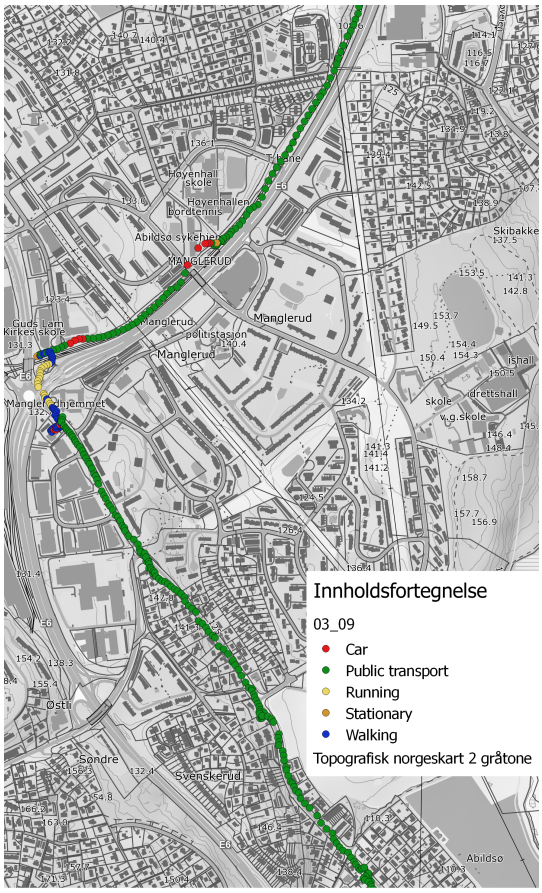
Videre som vist i figur 4.14 skal "moving" klassifiseres til transport eller aktivitet. I denne delen skal transport uten aktivitet som kjøring og kollektivtransport splittes fra transport ved bevegelse som gange, sykling og løping. Av denne problemstillingen blir akselerometeret viktig. Hvis akselerometeret gir et utslag som blir bestemt til aktivitet eller høy aktivitet fra algoritmen 4.6, er sannsynligheten for at punktet klassifiseres som aktivitet høy. Ved høy fart uten utslag på akselerometeret er sannsynligheten tilsvarende høy for at dette regnes som transport.

Fra klassifisering av bevegelse ble klassene transport og aktivitet opprettet. Transport skal klassifiseres videre til kollektivtrafikk eller bilkjøring. Kollektivtrafikk vil si buss, t-bane, trikk og tog. Dette er ganske lett å klassifisere ved

å utnytte geografisk data i PostGIS databasen. Databasen inneholder data som representerer tog-, t-bane og trikkelinjer som diskutert i avsnitt 4.3. Med lengde- og breddegrader som input i metode 4.10, returneres et boolsk uttrykk på om punktet er innenfor bufferen til kollektivtransportlinjene eller ei. Av dette blir t-bane, trikk og tog klassifisert riktig, så lenge nøyaktigheten på geolocation-målingene og de geografiske dataene i databasen er gode nok. Problemet ligger ved klassifiseringen av buss. Foreløpig finnes det ikke geografisk data for buss-ruter, men kun stopp. Om det fantes linjer ville det uansett være problematisk med tanke på at bilene kjører akkurat samme ruter. En metode ble laget der segmenter genereres av "transport", "stationary" og "public transport", der de fire første og siste datapunktene blir sjekket om de innenfor en buffer 10 meter fra busstopp. En metode for å sjekke busstopp ble utviklet, ved algoritme 4.11 og fungerer med samme logikk som algoritmen med kollektivtrafikklinjer, 4.9, der datarammen inneholder en kolonne med et boolsk uttrykk for bestemmelse om datapunktet er innenfor buffersonen til et stopp eller ei. Bussklassifiseringen fungerer, men det er en lite robust metode der det ikke skal mye til for å hindre metoden i å klassifisere riktig. Dette blir diskutert i kapittel 6.

Siste del av klassifiseringen bestemmer type aktivitet. Gange, sykling eller løping. Her blir datapunktene klassifisert som aktivitet evaluert. Fart og utslag på akselerometeret er grunnlag for analysen. Hvis farten er lav (under 8 km/t) er sannsynligheten stor for at brukeren går. Ved fart mellom 8 og 16 km/t regnes løping som sannsynlig. Ved høyere hastigheter enn 16 km/t vil sykling vektes høyest. Når Akselerometerverdiene er målt til "aktivitet" vil sannsynligheten være størst for å gå, og ved en verdi målt til "høy aktivitet" vil løping være sannsynlig.

Figur 4.15 viser resultatet for klassifisering ved ett datasett. En tur er blitt klassifisert til klasser, representer ved ulike farger som kan leses av innholdsfortegnelsen.



Figur 4.15: Klassifisert tur

Kapittel 5

Resultater

Datainnsamling er utført i en periode fra 1. februar til 1.april. Område for datainnsamling er Oslo og Ås. Data er hentet inn ved bruk av ulike bevegelser og transportmidler som tilsvarer klassene som klassifiseres. Brukeren som blir omtalt, er oppgavens forfatter.

Analysene av datasettene leverer gode resultater og gir over 90% riktig klassifiserte punkter på visse datasett. Deler av analysen er lite robust og kan enten gi veldig gode eller veldig dårlig resultater. Dette gjelder hovedsakelig bussklassifiseringer, som vil bli ytterligere diskutert i kapittel 6.

For resultattabellene vil klassene beskrives numerisk. Navn på datasett består av med dato for innsamling av informasjon. Følgende tall vil representere klassene:

1. Stasjonær
2. Gange
3. Løping
4. Sykling
5. Kjøring
6. Kollektivtransport

10. Ukjent

For å tallfeste resultatene av klassifiseringen, genereres det forvirringsmatriser. En forvirringsmatrise viser klassifiserte resultater opp mot sannhetsverdiene.

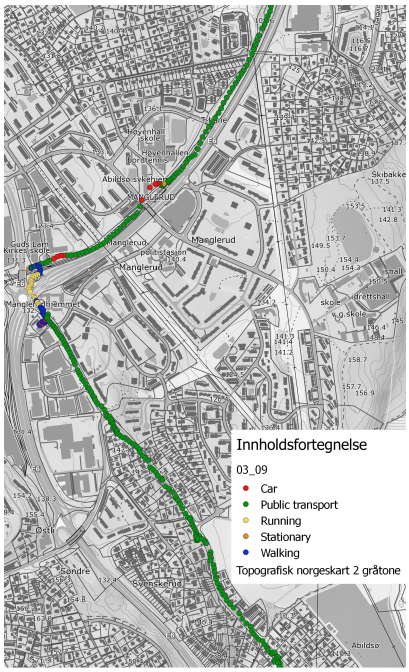
Kolonnene i matrisen viser resultater fra klassifisering og radene viser sannhetsverdiene. Diagonalen i matrisen viser korrekt klassifisert datapunkter. Forvirringsmatriser benyttes innenfor ulike klassifisering-anvendelser, og kommer ofte opp ved maskinlæringanvendelser. Den normaliserte forvirringsmatrisen viser prosentandel av klassifiseringen som har endt i den aktuelle klassen i forhold til antall sannhetsverdier.

I datarammen er det også etablert en kolonne som kalles "CORRELATION" som inneholder et boolsk utrykk om dataene er riktig klassifisert eller ei. Ved å se på antall datapunkter som er riktig klassifisert i forhold til totalt antall datapunkter, får vi en prosentandel av riktige klassifiseringer totalt.

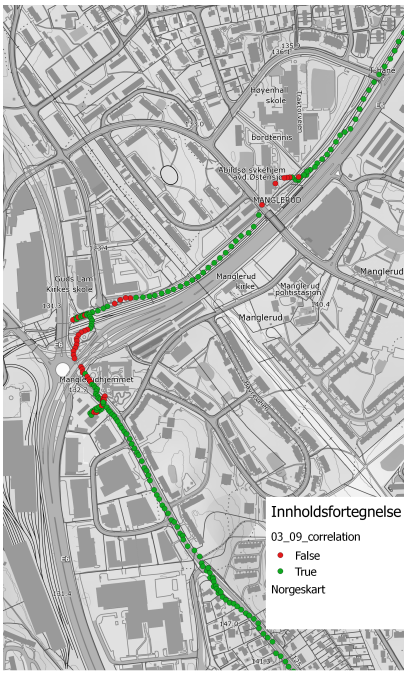
$$Korrelasjon = (Riktig\ klassifisert/Antall\ datapunkter) \times 100 \tag{5.1}$$

I påfølgende eksempler er kode 4.3, "diff_avg()", benyttet, med terskelverdi for aktivitet på 4 og terskelverdi for hard aktivitet på 13. Figurene vil kun vise utsnitt av kartene.

Eksempel 1, med tilhørende kart, figur 5.1, forvirringsmatrise, tabell 5.1 og normalisert forvirringsmatrise, tabell 5.2 viser et tilfelle der klassifiseringen har fungert svært godt. Loggen inneholder hovedsakelig kollektivtransport, med en overgang fra t-bane til buss, samt avstigning fra buss med noe gange. Av 431 punkter med kollektivtransport som sannhet har 419 punkter blitt riktig klassifisert, noe som tilsvarer 97.2%. Gange har blitt klassifisert 58% riktig, dette er ikke særlig imponerende, men kan forklares ut fra menneskelige feil. I en situasjon der man blir nødt til å løpe til bussen er det vanskelig å taste inn ny sannhetsverdi. Gjennomgående i analysen vil slike scenarier oppstå, der punkter kan være klassifisert riktig, men analysen viser feil. Dette blir diskutert i kapittel 6. Samlet resultat for klassifiserte punkter ligger i dette datasettet på 89.0% riktig klassifiserte punkter.



(a) Klassifisering



(b) Korrelasjon

Figur 5.1: 09.03 Klassifisert tur

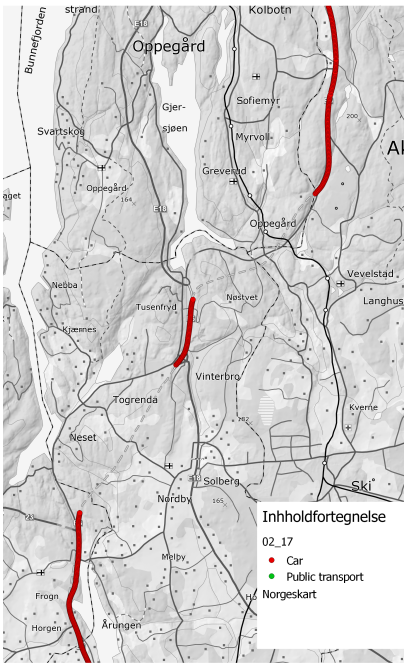
Tabell 5.1: 09.03 Forvirringsmatrise

Predikert Faktisk	1	2	3	5	6	10	Σ
1	0	1	0	0	4	0	5
2	1	57	17	8	15	0	98
3	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0
6	3	0	0	9	419	0	431
10	0	1	0	0	0	0	1
Σ	4	59	17	17	438	0	535

Tabell 5.2: 09.03 Normalisert forvirringsmatrise

Predikert Faktisk	1	2	3	5	6	10
1	0.0	0.2	0.0	0.0	0.8	0.0
2	0.01	0.58	0.17	0.08	0.15	0.0
3	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN
6	0.01	0.0	0.0	0.02	0.97	0.0
10	0.0	1	0.0	0.0	0.0	0.0

Eksempel 2, med kart, figur 5.2 viser et homogent datasett med kun en kjøretur med tilhørende forvirringsmatriser, tabell 5.3, og tabell 5.4. Matrisene viser at klassifisering av en helt normal kjøretur ikke fører til mange problemer. Med et resultat på 99.5% riktige klassifiseringer samlet, er det bra nok. Det er verdt å merke seg hullene i datasettet som er forårsaket av tunneler, der det ikke finnes GNSS signaler som kan gi posisjonsoppdateringer.



Figur 5.2: 17.02 Klassifisert tur

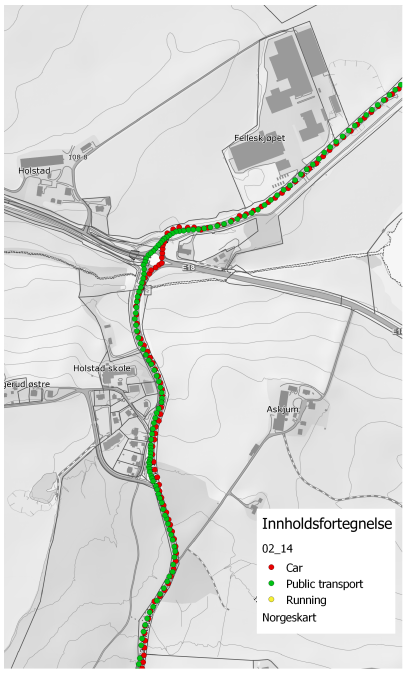
Tabell 5.3: 17.02 Forvirringsmatrise

Predikert Faktisk	5	6	10	Σ
5	930	2	0	932
6	0	0	0	0
10	3	0	0	3
Σ	933	2	0	935

Tabell 5.4: 17.02 Normalisert forvirringsmatrise

Predikert Faktisk	5	6	10
5	0.99	0.002	0.0
6	NaN	NaN	NaN
10	1.0	0.0	0.0

Eksempel 3, med kart, figur 5.3 viser et datasett med en kjøretur en vei og en busstur samme vei tilbake med tilhørende forvirringsmatrise, tabell 5.5 og normalisert forvirringsmatrise, tabell 5.6. Igjen gir klassifiseringen meget gode resultatet med 96% riktige gjettet bilkjøringsdatapunkter og 100% riktig klassifisert busstur. Dette gir totalt 98.4% riktige klassifiserte datapunkter.



Figur 5.3: 14.02 Klassifisert tur

Tabell 5.5: 14.02 Forvirringsmatrise

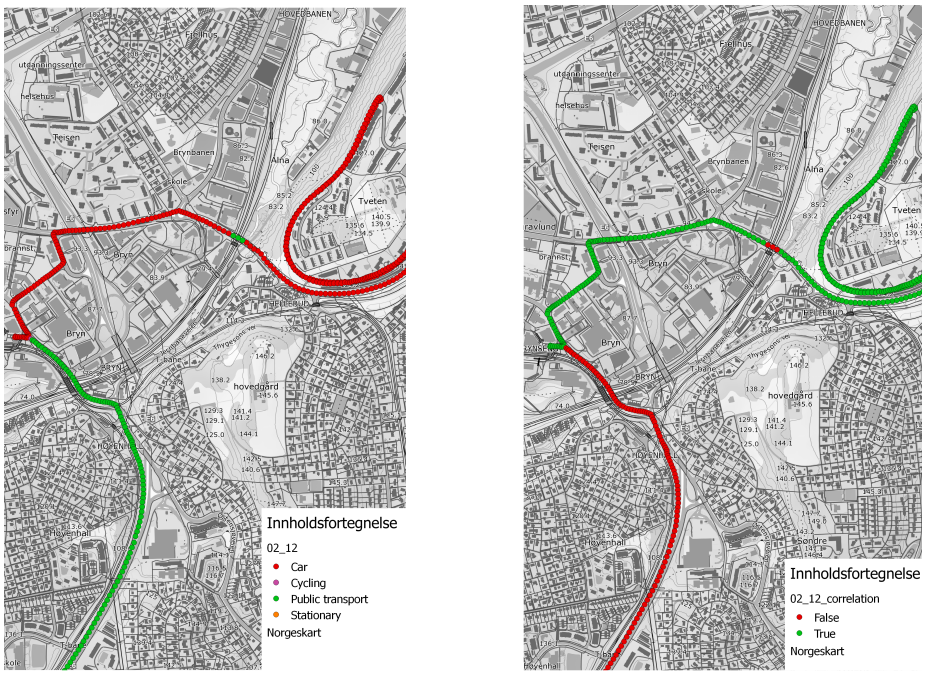
Predikert Faktisk	3	5	6	10	Σ
3	0	0	0	0	0
5	2	421	15	0	438
6	0	0	727	0	727
10	0	1	1	0	2
Σ	2	422	743	0	1167

Tabell 5.6: 14.02 Normalisert forvirringsmatrise

Predikert Faktisk	3	5	6	10
3	NaN	NaN	NaN	NaN
5	0.005	0.96	0.034	0.0
6	0.0	0.0	1.0	0.0
10	0.0	0.5	0.5	0.0

Eksempel 4, med kart, figur 5.4 viser et homogent datasett, der det kun har blitt kjørt ved forvirringsmatrise, tabell 5.7 og normalisert forvirringsmatrise, tabell 5.8. I eksempelet kan vi se fra resultatene at klassifiseringen ikke har gått like bra, med samlet resultat på 81.6%. Deler av kjøreturen har blitt klassifisert som kollektivtransport. En usannsynlig men mulig situasjon har oppstått, der en del av strekningen har blitt klassifisert som buss, grunnet feilklassifiseringer. Ruten klassifiseres som sykling både ved start og slutt, dette betyr at et segment genereres. Segmentet starter ved en busstasjon og ender ved en busstasjon, som betyr at det klassifiseres til buss. Dette er et ekstraordinært tilfelle, der alt går galt.

Deler av segmentet er også nærme t-banelinjer som gjør at brukeren er innenfor den satte bufferen. Ved å endre terskelverdier for differensieringsalgortimene eller bytte algoritme kunne klassifiseringen bedres. Problematikken diskuteres i kapittel 6.



(a) Klassifisering

(b) Korrelasjon

Figur 5.4: 12.02 Klassifisert tur

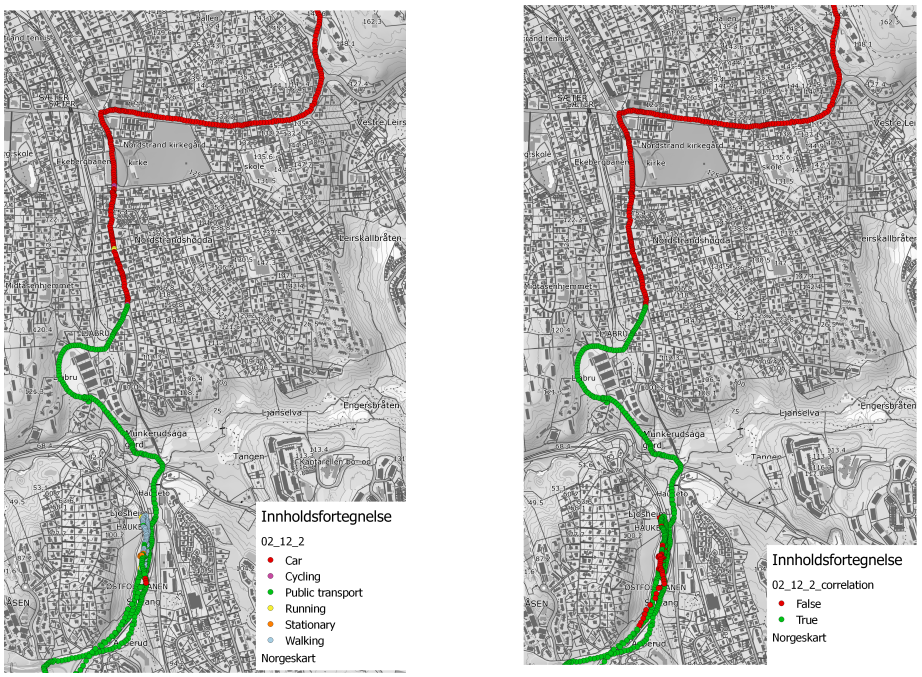
Tabell 5.7: 12.02 Forvirringsmatrise

Predikert Faktisk	1	4	5	6	Σ
1	0	0	0	0	0
4	0	0	0	0	0
5	4	1	1158	256	1419
6	0	0	0	0	0
Σ	4	1	1158	256	1419

Tabell 5.8: 12.02 Normalisert forvirringsmatrise

Predikert Faktisk	1	4	5	6
1	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN
5	0.0	0.0	0.82	0.18
6	NaN	NaN	NaN	NaN

Eksempel 5, med kart, figur 5.5 viser et datasett der tog, gange og buss er blitt benyttet som transport med tilhørende forvirringsmatriser, tabell 5.7 og tabell 5.8. Samlet resultat ligger lavt på 62.3%. Dette er grunnet i en bussklassifisering som ikke har blitt riktig. Brukeren har kjørt buss, men blitt klassifisert som kjørende. Dette skjer fordi noen punkter feilklassifiseres i løpet av segmentet som egentlig skal klassifiseres som buss. Disse feilklassifiserte punktene generer halve segmenter, som igjen enten ikke starter med busstopp eller slutter med busstopp som medfører at bussturen ikke blir bestemt.



(a) Klassifisering

(b) Korrelasjon

Figur 5.5: 12.02 - 2 Klassifisert tur

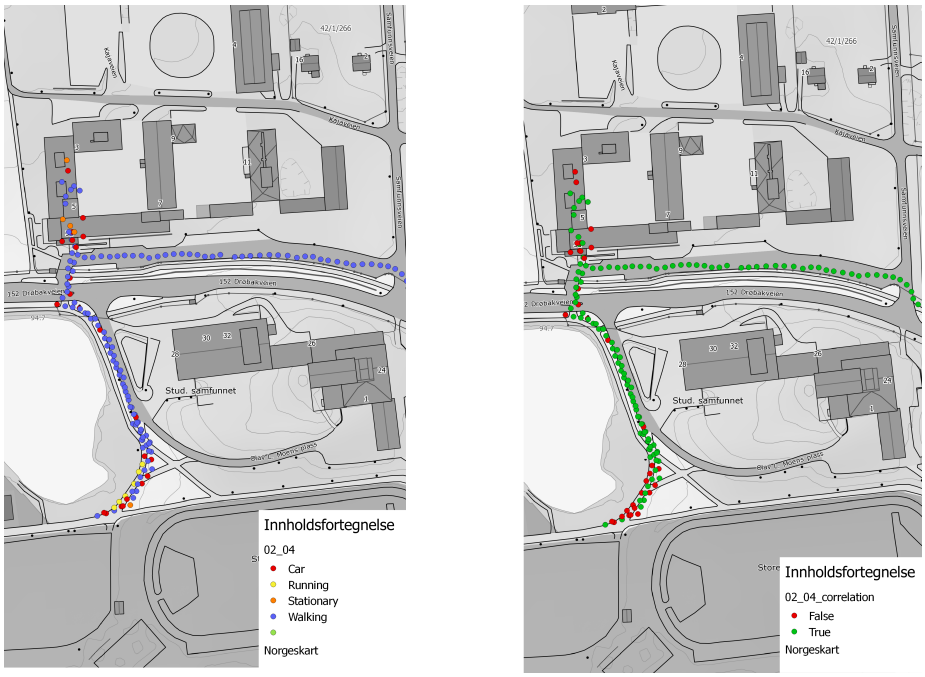
Tabell 5.9: 12.02 - 2 Forvirringsmatrise

Predikert Faktisk	1	2	3	4	5	6	10	Σ
1	1	0	0	0	11	2	0	14
2	3	125	2	0	5	14	0	149
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	23	7	6	4	572	1054	0	1666
10	3	33	0	0	7	22	0	65
Σ	30	165	8	4	595	1092	0	1894

Tabell 5.10: 12.02 - 2 Normalisert forvirringsmatrise

Predikert Faktisk	1	2	3	4	5	6	10
1	0.07	0.0	0.0	0.0	0.786	0.143	0.0
2	0.02	0.84	0.013	0.0	0.034	0.094	0.0
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	0.01	0.0	0.0	0.0	0.34	0.63	0.0
10	0.05	0.51	0.0	0.0	0.11	0.34	0.0

Eksempel 6, med kart, figur 5.6 viser et datasett med gange, tilhørende forvirringsmatrise er vist ved tabell 5.7 og normalisert forvirringsmatrise, tabell 5.8. Dette har gitt et resultat på 83.7% riktig klassifiserte datapunkter. Det kommer fram at datapunkter som har blitt klassifisert inne i bygninger ofte produser feil. Ved enkelte tilfeller får vi målinger fra akselerometer som tilsvarer "ikke aktivitet" av uvisse årsaker. Når dette forekommer blir også enkelte punkter klassifisert som bilkjøring.



(a) Klassifisering

(b) Korrelasjon

Figur 5.6: 04.02 Klassifisert tur

Tabell 5.11: 04.02 Forvirringsmatrise

Predikert Faktisk	1	2	3	5	10	Σ
1	3	2	0	1	0	6
2	1	187	7	25	0	220
3	0	0	0	0	0	0
5	0	0	0	0	0	0
10	1	0	0	0	0	1
Σ	5	189	7	26	0	227

Tabell 5.12: 04.12 Normalisert forvirringsmatrise

Predikert Faktisk	1	2	3	5	10
1	0.5	0.33	0.0	0.167	0.0
2	0.0	0.85	0.03	0.11	0.0
3	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN
10	1	0.0	0.0	0.0	0.0

Kapittel 6

Diskusjon

Som en helhet har analysen levert gode resultater ved optimale situasjoner, der resultatene tilsvarer eller overgår lignende studier. Noen deler av klassifiseringen varierer mye med tanke på resultater. Dette kommer av algoritmer som er lite robuste, der ulike scenarier kan gi problemer. Det finnes mange mulige løsninger for å analysere dataene, og noen har blitt testet ut i denne studien. Det finnes også mange muligheter for å forbedre analysene med ganske små grep, som diskuteres i dette kapittelet.

For å verifisere resultatene ble det benyttet tekstbokser der brukeren skrev inn sanne aktiviteter i appen. Her finnes det flere menneskelige feilkilder. Feilkilden har oftest kommet ved hyppige skifter av aktivitet. Ved scenarier som at bruker må løpe til bussen har det ikke blitt skrevet inn ny verdi. Det samme gjelder for stasjonære punkter, der bruker kan ha stoppet opp for en periode. Skrivefeil er også en feilkilde, men dette har blitt korrigert for i ordboken ved preprosesseringen. Hvis tekstboksen ble byttet ut med knapper for sanne aktiviteter i appen, ville gjort jobben til brukeren enklere, noe som ville medført bedre statistikk i analysen. Dette var dessverre ikke mulig, fordi klassene som benyttes i analysen ikke var avgjort ved oppgavestart.

Applikasjonen samlet inn akselerometerdata med en frekvens på 3.33 Hz og geodata hver fjerde meter. Det ville vært interessant å teste ulike frekvenser ved datainnsamlingen og sett om dette ville endret resultatene i analysen. Det er problematisk at telefonen som ble brukt til datainnsamling har blitt benyttet som en vanlig mobiltelefon og hovedsakelig ligget i lommen under innsamling, noe som kan skape støy i målingene. En løsning ved å ha telefonen fastmontert på kroppen ville redusert denne støyen. Selv om det skaper støy når telefonen ikke er fastmontert på kropp så skaper det likevel en realistisk

situasjon. Datainnsamlingen ble kun gjort for enkelte turer, det kunne også vært nyttig å samle data for lengre perioder. Dette ble vanskelig med tanke på både tidsforbruk ved analyse i Python, samt store filer som ville blitt generert.

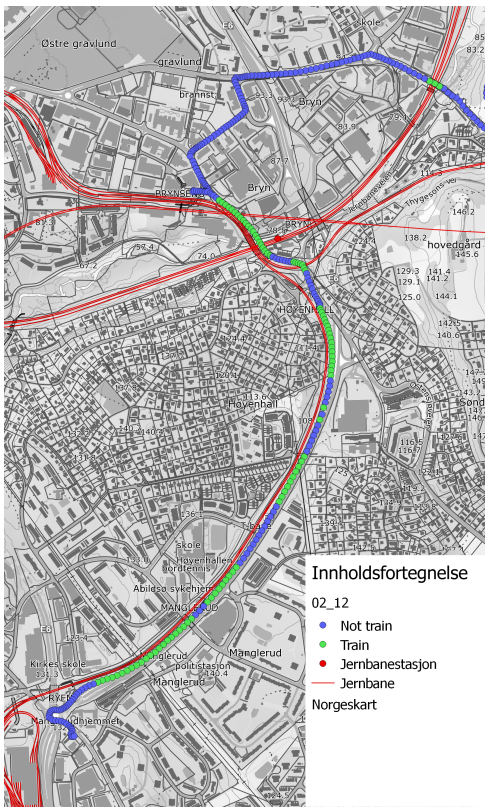
I analysene har det vist seg at homogene data, lengre strekk med samme aktivitet ofte klassifiseres med godt resultat. Ved visse tilfeller dukker det likevel opp enkeltpunkter eller mindre segmenter som er feilklassifisert. Ved interpolasjon/glattingsteknikker kan slike problemer løses. Et eksempel kan være en lengre gåtur som inneholder enkeltpunkter som er klassifisert som bil. Dette er lite sannsynlig og kan lukes ut. I bussklassifiseringen vil en slik glatting være essensiell på grunn av feilklassifiserte punkter som ødelegger klassifiseringen.

Bussklassifisering er den mest kritiske delen i analysen. Dette er den eneste delen der det benyttes segmenter. Segmentene består av det som da foreløpig er klassifisert innenfor transport i tillegg til stasjonære punkter. Som forklart i avsnitt 4.4 vil segmentene bli klassifisert som en busstur om de 4 første og 4 siste delene av segmentene er innenfor buffersonen til et busstopp. Segmentene vil ikke bli fullstendige ved feilklassifiserte punkter som vil ødelegge bussklassifiseringen. Bussklassifiseringen kan også bli ødelagt om kartdata ikke er oppdatert, eller uregelmessigheter ved bussruter og stoppeplasser. Det finnes flere muligheter for å gjøre denne klassifiseringen annerledes. Et eksempel som er vurdert er å lage en algoritme som ser på fart korrelert med busstopp. Om farten er lav/stillestående ved en stor del av busstopp i løpet av et segment er sannsynligheten stor for at det er en busstur. Dette ville vært en mer robust metode, fordi klassifiseringen ikke er avhengig av å treffe innenfor bufferen til busstopp ved start og stopp. Med denne metoden ville sannsynligheten for riktig klassifisering økt.

Som med bussklassifiseringen, ville det vært fornuftig å sett på segmenter i stedet for enkeltpunkter i resten av klassene. Dette ville også hjulpet med glattingen av produktet av analysen, noe som kunne hjulpet til med å øke prosentandelen riktig klassifiserte punkter.

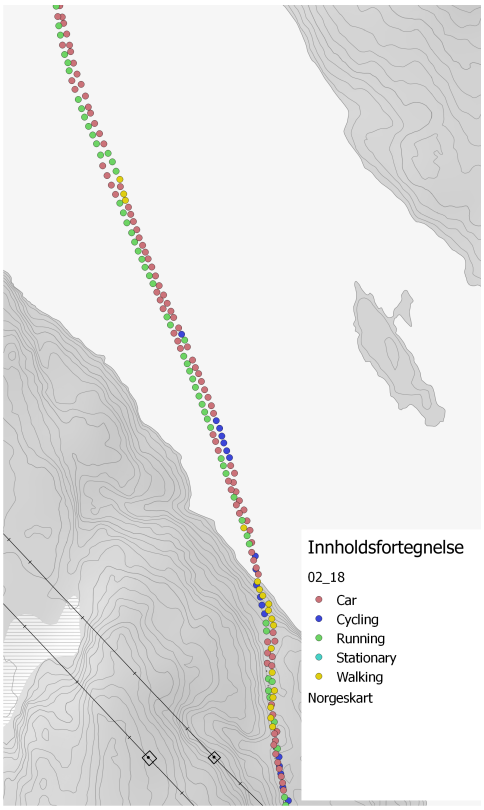
Sykling er en vanskelig klasse å bestemme. Dataene fra sykling ligner på bil, gange og løping, der bevegelsene har flere like egenskaper. Ved å se på et segment sine gjennomsnittshastigheter og gjennomsnittlige akselerometerverdier, kunne klassifiseringen forbedres.

En annen problemstilling dukker opp når en bruker kjører nær en toglinje/t-banelinjer, som illustrert i figur 6.1. Kartet viser at deler av turen er innenfor buffersonen til t-banelinjene, representert ved grønne punkter, punktene som er utenfor buffersonen er blå. Punktene er basert på "TRANSPORT" kolonnen i egenskapstabellen forklart i kapittel 4. T-banedata er markert med røde linjer i kartet. Datapunktene som er innenfor bufferen vil automatisk feilklassifiseres som kollektivtransport. Glattingteknikker kunne hjulpet til her ved å luke ut små segmenter klassifisert til kollektivtransport. Det er lite sannsynlig at en tur med t-bane tilsvarer små segmenter med datapunkter.



Figur 6.1: Kjøretur ved t-banelinjer

Ved bevegelser som ikke klassifiseres i analysen, kan det oppstå lite meningsfulle resultater. Figur 6.2 viser et datasett fra en skitur. Som kartet viser er klassifiseringen veldig spredt, der bevegelsene detekteres som gange, løping, kjøring, sykling og stillstand. En skitur gir ikke homogene resultater grunnet ulike aktiviteter. En kan kjøre nedover en bakke der farten er høy og akselerometeret ikke gir utslag for dermed å bli klassifisert som kjøring. Det kan gås i lav fart opp enn bakke der akselerometeret gir utslag, noe som klassifiseres til gange. Alle klassene bortsett fra kollektivtransport kan potensielt klassifiseres ved analyse av skiturer. En annen faktor er at teknikk og fart har en stor individuell variasjon.



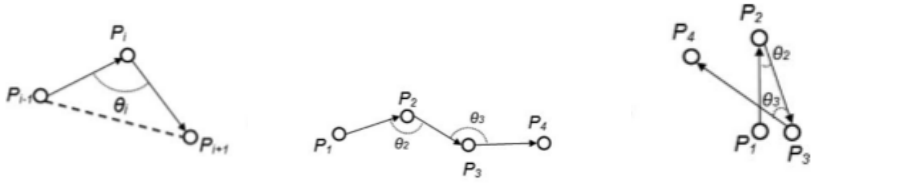
Figur 6.2: Skitur

Figur 6.3 viser et datasett fra innendørs aktivitet bruk der nøyaktigheten er dårlig. Nøyaktighet tallfestes fra geolocation-API'et Feilene for disse målingene er fra 15 til 58 meter. Dette gir problemer ved innendørs bruk. En løsning kunne vært å klassifisere alle punkter med dårlig nøyaktighet til innendørs eller stasjonær. Det kunne også tenkes at alle datapunkter med dårligere nøyaktighet enn en terskelverdi skal fjernes. Transport som kjøring, t-bane, buss med mer gir heller ikke den beste nøyaktigheten, mellom 10 og 20 meter. Dette er ikke et stort problem fordi datapunktene relativt til hverandre har en god presisjon. Som en følge av dette vil det være problematisk å slette punkter, fordi de relativt til hverandre har en god presisjon. En god løsning ville vært å klassifisere alle punkter med dårlig nøyaktighet og lav fart til innendørs eller stasjonær.

Figur 6.3: Innendørs innsamling av romlige data med GNSS-mottaker i mobiltelefon

Fart er den viktigste egenskapen fra geolocation-data fordi forskjellige bevegelser gjerne har ulike hastigheter. Farten blir regnet ut basert på avstand mellom hvert punkt i forhold til deres tidsstempel. Ved signaltap fra GNSS-mottakeren og dårlige målinger vil dette påvirke posisjon og fart. Av denne grunn bør lange avstander eller store tidsdifferanser mellom etterfølgende punkter behandles som anomalier. Studien til [Wan and Lin, 2013] setter da enten farten til 0 eller fart tilsvarende forrige punkt.

Figur 6.4 viser hvordan vinkelen i datasettet kan benyttes som en ekstra variabel ifølge studien til [Wan and Lin, 2016]. Vinkel defineres ved forholdet fra forrige punkt gjennom referansepunktet til påfølgende punkt, slik det er illustrert i figuren. Punktvinkler blir store ved bevegelser som gange og transport og små når brukeren står i ro eller er inne grunnet klyngeeffekter. Det er problematisk å benytte disse vinklene i denne studien fordi distanse-filteret i geolocation-API'e ikke svekker slike klyngeeffekter.



(a) Definisjon av punktvinkel

(b) Punktvinkel ved bevegelse

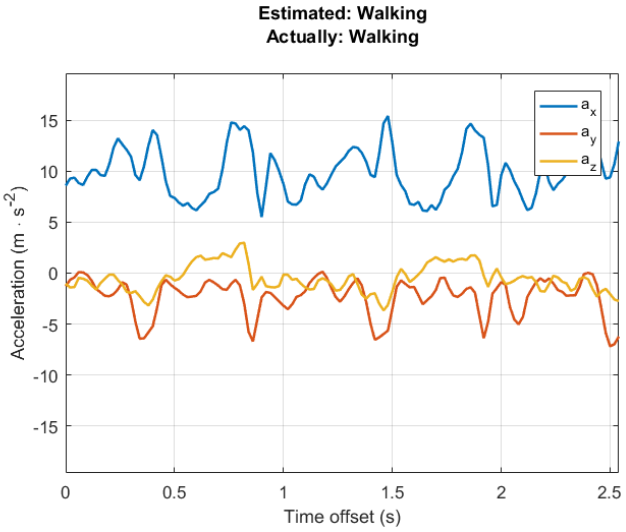
(c) Punktvinkel for stasjonære datapunkter

Figur 6.4: Punktvinkel [Wan and Lin, 2016]

Dersom oppgavene skulle videreføres ville det vært nyttig å integrere flere sensorer og mer kartdata i analysen. Det finnes flere muligheter for å gjøre analysen mer robust. Pedometerinformasjon (skritteller) kan gi enda bedre forståelse for hva som er aktivitet og hva som ikke er aktivitet. Pedometeret er en avledning av akselerometerdataene, så selve informasjonen bygger i bunn og grunn på det samme. Ved å benytte data fra sensorer som barometer, termometer og luftfuktighet kan endringer av omgivelser detekteres kjapt, for eksempel om en bruker er ute eller inne. En annen applikasjon ville vært muligheten til å hente informasjon om hendelser på mobiltelefonen. Om brukeren er aktiv på telefonen samtidig som det klassifiseres transport vil sannsynligheten være høy for at brukeren benytter kollektivtransport eller eventuelt er passasjer i en bil. Datakilder fra Statens Vegvesen som NVDB (Norsk vegdatabank) og trafikkdata i samtid kunne blitt integrert i analysen for å levere informasjon til bestemmelse av transporttype. Værdata fra met.no (Meterologisk Institutt) kunne potensielt bli brukt i kombinasjon med barometer og termometer for å

bestemme endring av omgivelser.

Dersom oppgavene skulle utvides burde maskinlæringsteknikker benyttes. Mathworks[1], som leverer programmeringsplattformen Matlab er en aktør som forsker og leverer produkter innen fagfeltet. Webinaret "Signal Processing for Machine Learning"[2], med supplerende kode[3] viser muligheter for å benytte akselerometerdata ved signalbehandling og maskinlæringsteknikker. Figur 6.5 viser et utsnitt fra resultatet, der akselerometerverdiene gir grunnlag for å bestemme klassene: Gange, gange i trapp opp og ned, sitting og ligging. Dette er løst ved å trene et nevralt nettverk med "Signal Processing Toolbox" og "Neural Network Toolbox". For slike anvendelser er det nødvendig med høyere frekvens på innsamlede data enn det som er benyttet i denne oppgaven.



Figur 6.5: Mathworks maskinlæring

Ved å anvende maskinlæringsteknikker i oppgaven kunne mer informasjon bestemmes ut fra akselerometerdataene og dette ville bidratt til å bedre kvaliteten på resultatene. Ved analyse av signalene kunne aktivitet blitt klassifisert direkte, med støtte fra geolocation-informasjon. Maskinlæringsteknikker kunne også blitt benyttet utover kun signalbehandling. Ved å benytte treningsdata med sannhetsverdier, kunne klassifiseringssmodulen optimaliseres der analysen ville blitt trent til å finne optimale parametere og terskelverdier. Terskelverdiene i

[1] Mathworks `https://se.mathworks.com/`

[2] Signal Processing for Machine Learning `https://se.mathworks.com/videos/signal-processing-for-machine-learning-99887.html`

[3] Kode for Signal Processing for Machine Learning `https://se.mathworks.com/matlabcentral/fileexchange/49893-code-for-webinar--signal-processing-for-machine-learning-`

oppgaven er empirisk bestemt og det er ikke gjort mye testing for å optimere parametere, noe som er en svakhet i analysen. De ulike differensieringsalgoritmene burde også vært testet mer mot hverandre.

Kapittel 7

Konklusjon

7.1 Del 1

Del 1 av problemstillingen innebar å utvikle en applikasjon for innhenting av sensordata fra mobiltelefon. Appen ble utviklet med rammeverket React Native. 162 linjer kode (som lett kan komprimeres) og noe konfigurasjon av biblioteker illustrerer at React Native er effektivt og enkelt rammeverk. Dette kan også sies om applikasjonen som ble utviklet, enkel men effektiv for sitt formål.

Lagringen av data med server i skyen og API bygget ved Python-rammeverket Flask har også blitt til ved forholdsvis enkle grep. Dette har sammen utgjort en enkel og funksjonell applikasjon for innhenting av sensordata av god kvalitet.

For ferdigstilling av appen, bør tekstboksen byttes ut ferdigdefinerte knapper for sannhetsverdier.

7.2 Del 2

Kombinasjonen av romlige data, geolocation- og akselerometerdata har vist seg å være overraskende effektivt. Analysen av datasettene har generelt levert gode resultater, med noen unntak.

Geolocation data med egenskaper som posisjon, fart og nøyaktighet gir et godt utgangspunkt for analysen. Ved å legge til aktivitetsanalyse basert på akselerometerverdier styrkes klassifiseringen, der aktiviteter skilles fra transport.

Klassifiseringen styrkes ytterligere med geo-fencing der romlige data har blitt benyttet til levere parametere basert på brukerens posisjon.

For å øke resultat på klassifiseringen ytterligere kunne algoritmene utviklet for oppgaven vært forbedret med dypere testing, slik at problematiske scenarier tas hensyn til. Testing av ulike differensieringsalgoritmer, for å bestemme optimal løsning ville vært fornuftig. Terskelverdier for algoritmer burde også optimaliseres. Implementering av maskinlæringsteknikker burde sterkt vurderes ved videre forskning, trente nettverk vil gjøre enn bedre jobb ved å evaluere akselerometerdata og bestemme parametere for klassifiseringen.

Med resultatene fra denne studien er den opprinnelig ideén for studentprosjektet Road2zero oppnåelig. En app som skulle identifisere bruk av transportmidler automatisk, for å gi en poengsum basert på klimavennlighet.

Det finnes mange muligheter for integrering av sensorer, utarbeidelse av analysen og utnytte mer kartdata. Analysen kan benyttes til å forske på ulike fagfelt der oppgavens produkt kan benyttes som et verktøy.

Bibliografi

Android-Developers (2017). Motion sensors. https://developer.android.com/guide/topics/sensors/sensors_motion.html#sensors-motion-linear. 2017-01.

Bevaqua, N. (2017). React, jsx and es6: The weird parts. https://ponyfoo.com/articles/react-jsx-and-es6-the-weird-parts. 2017-02.

Brooke, J. D. and Whiting, H. T. A. (1973). *Human movement : a field of study*. Henry Kimpton Publishers.

Dvergsdal, H. (2017). Store norske leksikon, server. https://snl.no/nevralt_nettverk. 2017-04.

Eisenman, B. (2016). *Learning React Native - Building Native Mobile Apps with Javascript*. O'Reilly.

Forssell, B. (2017). Store norske leksikon, gnss. https://snl.no/GNSS. 2017-02.

GitHub (2017). Python libraries by github stars. https://github.com/search?o=desc&q=language%3APython&s=stars&type=Repositories&utf8=%E2%9C%93. 2017-04.

Godfrey, A., Conway, R., Meagher, D., and ÓLaighin, G. (2008). Direct measurement of human movement by accelerometry. *Medical Engineering & Physics*, 30(10):1364 – 1386.

Granevang, M. (2017a). Store norske leksikon, backend. https://snl.no/Backend. 2017-05.

Granevang, M. (2017b). Store norske leksikon, frontend. https://snl.no/Frontend. 2017-05.

Grinberg, M. (2014). *Flask Web Development - Developing Web Applications with Python*. O'Reilly.

Gup, A. (2017). Html5 geolocation api - how accurate is it, real-

ly? http://www.andygup.net/html5-geolocation-api-%E2%80%93-how-accurate-is-it-really/. 2017-02.

Hagen, C., Evans, H., Ciobo, M., Miller, J., Wall, D., and Yadeav, A. (2013). Big data and the creative destruction of today's business models. *Atkearney*.

Hamill, J. and Knutzen, K. M. (2003). *Biomechanical Basis of Human Movement*. Lippincott Williams & Wilkins.

Hofmann-Wellenhof, B., Lichtenegger, H., and Wasle, E. (2008). *GNSS - Global Navigation Satellite Systems. GPS, GLONASS, Galileo & more*. Springer. Wien/New York.

Liseter, I. M. (2017). Store norske leksikon, server. https://snl.no/server. 2017-04.

Mafrur, R., Nugraha, I. G. D., and Choi, D. (2015). Modeling and discovering human behaviour from smartphone sensing life-log data for identification purpose. *Human-centric Computing and Information Sciences*, 5(1):31.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill Science/Engineering/-Math.

Rigaux, P., Scholl, M., and Voisard, A. (2001). *Spatial Databases With Applications to GIS*. Morgan Kaufmann.

Seeber, G. (2003). *Satellite Geodesy*. Walter de Gruyter, 2 edition.

Tipler, P. A. and Mosca, G. (2007). *Physics For Scientists and Engineers*. W. H. Freeman and Company, 6 edition.

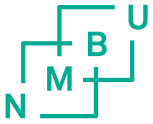
Wan, N. and Lin, G. (2013). Life-space characterization from cellular telephone collected telephone gps data. *Computers, Environment and Urban Systems*, 39:63–70.

Wan, N. and Lin, G. (2016). Classifying human activity patterns from smartphone collected gps data: A fuzzy classification and aggregation approach. *Transactions in GIS*, 20:869–886.

Wikipedia (2017a). Akselerometer. https://no.wikipedia.org/wiki/Akselerometer. 2017-01.

Wikipedia (2017b). Maskinlæring. https://no.wikipedia.org/wiki/Maskinl%C3%A6ring. 2017-04.

Wikipedia (2017c). Simple features. https://en.wikipedia.org/wiki/Simple_Features. 2017-03.



Norges miljø- og biovitenskapelig universitet
Noregs miljø- og biovitskapelege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway