NTNU

Kunnskap for en bedre verden

BIELSYS

IELS2001 - Ingeniørprosjekt I

Smart City-prosjekt for Ytre Ringvei

Av:
Ali Amlashi, Emil Hallingstad, Ove Ljosland, Andreas Lindeman og Mathias Schiøtz

6. Juni 2024

1 | Abstrakt

På grunn av vanskeligheter med å oppnå klimamålene har kommunen har bestemt at det skal skje en elektrifisering av biler innenfor ytre ringvei.

Det presenteres flere løsninger for å håndtere overgangen til en fullelektrisk bilpark innenfor Ytre Ringvei. Fra et brukerperspektiv går den første løsning ut på å samle inn data om kjøremønster som senere kan brukes til å straffe eller belønne sjåfører. Videre ble det utviklet en løsning for å bedre kunne bestemme når elbiler burde bli ladet, samt at strømprisen kan bli forandret basert på kjøremønsteret. I tillegg ble en løsning for kommunen utviklet, hvor det ved hjelp av datasyn og bildegjenkjenning er mulig å detektere ikke-elektriske biler og gi bøter hvis disse kjører på områder de ikke har tillatelse til. Til slutt ble det også utviklet en løsning for et serversystem for å koble sammen alle modulene.

Totalt sett gir løsningene et godt og bredt utgangspunkt for å kunne elektrifisere kommunen. Enkeltpersoner får muligheter til å engasjere seg i hvordan de kan bidra til elektrifiseringen, samt at kommunen får kraftige verktøy til å sikre at bestemmelsene de har gitt faktisk blir opprettholdt. Alt dette blir satt sammen av et felles server-system, som gjør det mulig å enkelt opprettholde alle løsningene i tillegg til å like enkelt legge til nye løsninger etterhvert som dette ses som nødvendig.

Innhold

2 | Introduksjon

På grunn av vanskeligheter til å oppnå klimamålene satt i den gjeldende storbyen, har det blitt bestemt at det nå må gjøres drastiske tiltak for å få til å oppnå dem. Dermed har valget blitt tatt om at det fra 2025 bare skal være helelektriske biler innenfor ytre ringvei, hvor 80 prosent av befolkningen bor. Dette valget vil nødvendigvis føre til en betydelig belastning av strømnettet, noe som ikke kan utbedres i tide til reguleringsendringene. Dermed er det krav om å finne løsninger som kan hjelpe til med å redusere strømbruken i kommunen i perioden hvor strømnettet er tilstrekkelig godt nok.

Gjennom semesteret som prosjektet har vært en del av, har det vært flere mål som kan organiseres i de ulike kategoriene: kunnskaper, ferdigheter og generell kompetanse. For kunnskap gjelder dette å kunne bruke kunnskapen lært i andre emner som tas parallelt og bruke det for å finne løsninger til utfordringer som oppstår i prosjektet. For ferdigheter gikk det ut på å kunne designe mer komplekse IoT-løsninger, samt å organisere dette effektivt og med gode rutiner for gjennomføringen av prosjektet. Til slutt går generell kompetanse ut på å kunne jobbe strukturert gjennom et helt prosjekt[1].

Disse læringsmålene har i emnet kommet som form av et todelt opplegg, et prosjekt og 7 øvinger som først gikk ut på å gi en introduksjon av ESP32 og forskjellene mellom den og en Arduino Uno, for så å gå mer i detalj på hvordan en ESP32 kan brukes til trådløs kommunikasjon, blant annet sammen med ulike IoT-platformer. I tillegg ble det også sett på hvordan man kan bruke en Raspberry Pi, både som en vanlig, men mindre kraftig datamaskin, og som en mye mer kraftig mikrokontroller.

Alt dette ble senere brukt til prosjektet som dette dokumentet omhandler. Først og fremst var øvingene som ble gjennomgått essensielt for å få et godt grunnlag for å kunne gjennomføre prosjektet. Dette er fordi det ikke vil være realistisk å kunne gjennomføre et et prosjekt som det dokumentet beskriver, uten å på en eller annen måte bruke en kosteffektiv måte å kunne innhente og analysere data fra enkeltbrukere. Dette er hvor mikrokontrollere som ESP32 eller, hvis en trenger en betydelig mengde mer datakraft, Raspberry Pi. Videre har det vært viktig å ha et godt grunnlag i elektronikk, mattematikk og fysikk, som alt er kunnskap innhentet i andre emner i samme semester, for å kunne ha en god nok forståelse av problemstillingen til å kunne lage en god nok prototype som presterer som ønsket. Til slutt

[1]Hentet fra ´om emnet´ fra emnet IELS2001 - Ingeniørprosjekt 1.

er både læringsmålene om ferdigheter og generell kompetanse læringsmål som går direkte inn i et prosjekt siden dette er et område hvor større og mer komplekse systemer kan produseres.

For dette prosjektet har flere ulike utfordringer blitt adressert. Dette inkluderer deteksjon av drifstofftype hos biler for å kunne sikre at bare helelektriske biler er innenfor ytre ringvei, som kan brukes av byen for å blant annet bøtelegge bensin- og dieselbiler som bryter denne loven. Videre brukes deteksjon av kjøremønster av enkeltpersoner for å kunne klassifisere hvor miljøvennlig en sjåfør er. Her er det tenkt at det er et brukergrensesnitt for brukeren for at den skal kunne se hvor bra personen kjører, men at løsningen vil bli utviklet og distribuert av for eksempel byen. Dette systemet er også tenkt å kunne bruks videre til en løsning for et smart ladesystem, hvor det skal være mulig å kunne bestemme ladetidspunkter for å være mer skånsom på strømnettet, samt å kunne variere prisen på ladingen basert på hvor miljøvennlig bilen blir kjørt. Her vil det igjen være et brukergrensesnitt for brukeren, men fortsatt vil løsningen nødvendigvis bli utviklet og distribuert av for eksempel byen. Til slutt har det blitt tenkt en serverløsning som skal brukes til å samle alle løsningene til et sammenhengende produkt og som da vil bli brukt av de som samme som utvikler back-enden til de andre løsningene.

3 | Prosjektmetodikk

Prosjektmetodikk spiller en avgjørende rolle i prototype-utvikling. Det er flere grunner til at dette er viktig. Først og fremst medbringer en veldefinert prosjektmetodikk et rammeverk som hjelper laget med å jobbe strukturert med sikker fremdrift. En annen viktig grunn til å ha en god prosjektmetodikk er at det sikrer god kommunikasjon og samarbeid innad i laget. Prototyping pleier å innebære utfordringer som må løses kollektivt, og en god prosjektmetodikk legger til rette for samarbeidet.

Når det gjelder valg av prosjektmetodikk, er det viktig å vurdere prosjektets spesifikke krav og behov. Agile og Lean er to av de mest anvendte metodikkene innen designtenkning, kjent for å sikre effektivitet, fleksibilitet og kvalitet. Disse metodikkene har røtter i produksjonsindustrien og har blitt tilpasset og videreutviklet for å holde følge med dagens innovasjon.

Agile er en metodikk som fremmer kontinuerlig forbedring, fleksibilitet og legger vekt på å involvere brukeren. Formålet er å skape et produkt som kan tilpasse seg brukerenes nye behov. Metodikken ble utviklet etter behov for å levere

programvare. Agile prioriterer *individer og interaksjoner* over prosesser og verktøy, *fungerende programvare* over omfattende dokumentajon, *kundesamarbeid* over kontraktsforhandling og *tilpasningsevne* over å gjennomfølge prosjektplanen (Schneider 2017).

Lean ble først beskrevet etter Toyota's produksjonsfilosofi, *Kaizen*[2] i tiden etter andre verdenskrig; der grunnlaget ble lagt for Lean designmetodikk ble lagt; kontinuerlig forbedring av kvalitet og fleksibilitet gjennom refleksjon og læring og arbeidsorganisering etter kundebehov (Schneider 2017). Lean metodikk legger vekt på kontinuerlig forbedring og effektivitet gjennom endringer som tilpasser brukerens behov og ønsker. Det gjelder å identifisere essensen av det verdifulle for brukeren slik at det kan anvendes til å rette arbeidet mot sikringen av denne verdien. Innenfor lean metodikken er det sentralt å strebe etter en kontinuerlig forbedring i alle aspekter av prosessen. Det er også viktig å forsyne verdi til brukeren som fort som mulig.

I vårt prosjekt har vi valgt å vektlegge lean-metodikken fremfor Agile. Først og fremst bidrar dette i å skape en kontinuerlig flyt i arbeidsprosessen ved å identifisere og eliminere alt som ikke gir verdi til brukeren. Vi har implementert prinsippene ved å sikre at vi hele tiden forbedrer prosesser og eliminerer ineffektivitet så langt det strekker seg. Dette har ført til at modulene våres er mer tilrettelagt til å gi brukeren verdi. Videre har vi tilpasset metodikken til vårt prosjekt ved å simulere hyppige tilbakemeldinger og justeringer. I motsetning til agile, som ofte preges av korte utviklingsfaser og hyppige leveranser har lean-metodikken gitt oss mulighet til å fokusere på en langsiktig bærekraftig og kontinuerlig forbedring. Mens agile også legger vekt på tilpasning og fleksibilitet ligger styrken til lean i å etablere et kvalitetsprodukt på stedet. Dermed så har lean-metodikken vært mer egnet for vårt prosjekt, hvor kompleksiteten til systemet vårt krever nøye planlegging.

4 | Designtenking og -utvikling

Designtenking har etablert seg som en sentral metode innen innovasjon og problemløsning for å skape brukerorienterte løsninger. Metoden legger vekt på empati, samarbeid, og iterasjon. I denne rapporten skal vi utforske de grunnleggende prinsippene for designtenking og hvordan disse prinsippene kan anvendes i en designutviklingsprosess.

[2]Japansk: *Forbedring*. Prinsippet om kontinuerlig forbedring.

4.1 Empati

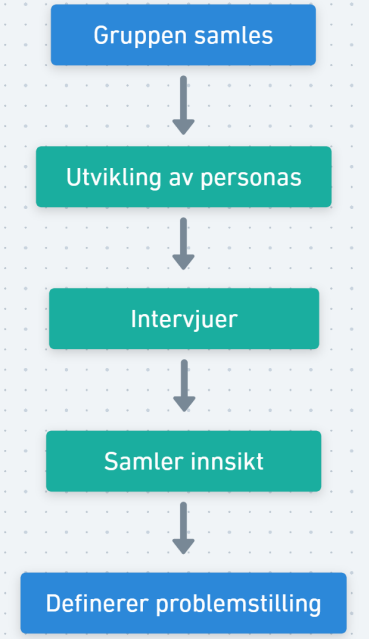
Designtenking starter med empati. For å skape løsninger som møter brukernes behov må vi som designere først forstå behovene til brukeren på et dypt nivå. Dette innebærer å observere og engasjere seg med brukerne, samt hente innsikt i deres følelser, opplevelser og motivasjoner. Innad i gruppen brukte vi målrettede personas for å undersøke brukerene. Personas er fiktive karakterer som vi har laget for mulige innbyggere i ytre ringvei. Disse karakterene skal representere mangfoldet i ytreringvei og gi oss et bedre innblikk på livene til brukerene. En tabell over personas er vedlagt under brukeranalyse med personas i vedlegget. De fiktive personas-ene kan deles opp i to hovedgrupper; de teknologiske anlagte og de som har vanskeligheter med å tilvenne seg moderne teknologi. Michael (35) er et eksempel på en teknologisk anlagt person, mens Kari (67) kan være en person som har vansker med å tilpasse seg nyere teknologi. Målgruppen til Kari er ofte en stor og høylytt gruppe, og er derfor viktig å ta hensyn til med i designet.

Løsningene våres må derfor forsiktig velges slik at man ikke utvikler for mange pain-points"hos brukerene. For Frank (45) hadde det for eksempel vært veldig problematisk hvis vi skulle hatt bannlyst alle bensinbiler. Da hadde Frank ikke hatt mulighet til å kjøre seg selv eller barna til jobb eller skolen. Derfor må andre løsninger til for å løse problemet til Frank. Hvis man eksempelvis gjør rede for en løsning som gjør det fordelaktig å bytte transport middel, enten det skulle være en ny elbil for familien eller kollektiv transport kunne dette vise seg til å være lønnsomt for Frank og familien. Dette kan gjøres på forskjellige måter. En mulighet er å minke gunstigheten av å ha en bensinbil ved å gradvis ta bort parkeringsmulighetene, og øke bensinprisene mens man samtidig tilby gratis parkering for elbiler. På denne måten kan vi hjelpe Frank med å ha en lettere skifte ved at det faktisk lønner seg å gå over til et mer bærekraftig transportmiddel med tanke på økonomien.

Det må likevel sies at selv om dette kunne vært en mulig løsning for Frank, betyr det ikke at alle andre innbyggere i ytreringvei vil kunne tilpasse seg en slik endring. Noen vil overhodet ikke ha muligheten til å kunne investere i en ny elbil, så for de personene må det bli tiltenkt en annen løsning. Det er nemlig ikke et fasit svar når det gjelder designtenking. Det er flere brukerere med ulike mål og ønsker. Slik at en ideell løsning vil innebære å kunne tilfredstille så mange brukere som mulig, og unngå total misnøye hos de brukerene som ikke får det akkurat som de vil.

Det ble også gjennomført intervjuer der vi satte oss inn i personas-ene våres. Dette bidro til å øke empati-en våres ovenfor brukerene ved at man fikk oppleve problemene til brukeren i virkeligheten. Ved å dypdykke i brukerenes liv på denne

måten har vi mulighet til å forstå deres behov og ønsker. Dette er det som fører til at vi samler innsikt. Når man har samlet nok innsikt som fører til empati ovenfor brukeren, betyr det at du har forstått deres behov og ønsker. Dette fører til at vi nå kan lettere tilpasse prosjektarbeidet og løsningen våres til å komme overens med brukerenes utfordringer. Et flytskjema for empati-prossesen er illustrert i figuren under:



Figur 1: Flytdiagram for empati-prossesen

4.2 Definisjon

Definisjonsfasen spiller en sentral rolle i utviklingen av en brukervennlig løsning. Denne fasen omhandler å klargjøre problemstillingen, fokusere prosjektets retning og utvikle designkriterier og forbedre grunnlaget for kreativitet og innovasjon. Gjennom denne prosessen blir innsikten fra empatifasen omformet til en klar og konkret problemstilling som kan veilede arbeidet fremover.

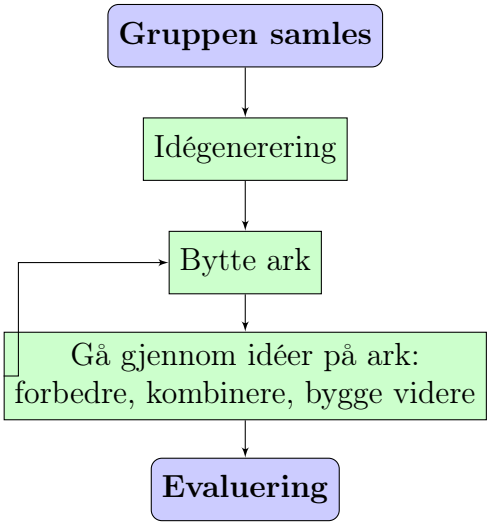
Basert på innsikten vi fikk fra personas-ene og intervjuene, begynte vi å definere og prioritere problemene. Vi fokuserte først å identifisere de ulike smertepunktene til innbyggerne i ytre ringvei. For eksempel er David (28) som er eier av et frakterselskap er bekymret over midler som krever for å elektrifisere bedriften. Når vi har identifisert kjerneutfordringene til hver og en person har vi mulighet for å klargjøre en konkret problemstilling som skal veilede veien videre.

Det neste steget for å definere en god problemstilling er å prioritere behov. Som sagt, i de fleste tilfeller er det ikke mulige å tilfredstille alle sammen, men tanke

på at folk har ulike behov og ønsker. Derfor må vi prioritere de viktigste behovene som vi kan gjøre rede for, med tanke på at vi samtidig følger bystyrets ambisjoner for ytre ringvei. Dette gjorde vi ved å skrive ned hvilke mål bystyret hadde for ytre ringvei samtidig som undersøkte personas-ene våres mål og muligheter. Da gjenstod det å undersøke ulike måter vi kunne kombinere innsikten vi hadde fått, med ambisjonene til bystyret for å etablere en best mulig problemstilling. Dette var det som skapte grunnlaget til ideutviklingen.

4.3 Idéutvikling

Når problemstillingen er definert må laget vurdere et bredt spekter av mulige løsninger. Denne delen av design-tenkningen oppmuntrer til kreativ tenking, og blir ofte omtalt som idefasen. Idéutvikling er en grunnleggende del av designarbeidet, hvor innovasjon og tverrfaglig samarbeid er avgjørende. Smart city-prosjekter involverer ofte komplekse problemstillinger som krever kreative løsninger for å forbedre byliv og bærekraft. Før idégenereringen kan starte, er det viktig å forstå problemstillingen. Derretter starter selve idégenereringen med brainstorming, hvor alle team-medlemmene presenterer idéene sine fritt. For å bedre strukturere idéene og forslagene, utføres idégenereringen etter *brainwriting* prinsippet. Denne teknikken brukes for å generere et stort antall idéer på en strukturert måte. I motsetning til tradisjonell brainstorming der alle team-medlemmene roper ut idéer muntlig, foregår brainwriting skriftlig. Prosessen er vist i flytskjemaet i figur 2 under:



Figur 2: Flytdiagram for brainwriting-prosessen.

Prosessen starter med at alle team-medlemmene samles. Hver person tildeles ark og en tidsramme på 5-10min til å skrive ned så mange idéer som mulig. Idéene

skal være korte og konsise. Når tiden er ute bytter team-medlemmene arkene med hverandre, og forbedrer og bygger videre på idéene. Denne prosessen gjentas til alle idéene er gjennomgått av hvert team-medlem. Til slutt samles arkene inn.

Utviklingen startes ved at team-medlemmene tildeles ark.Dette viste seg å være en bedre løsning, siden tidspresset ved brainstorming ofte kan kvele gode idéer. noe som bidrar til å overvinne tidspresset ved tradisjonell brainstorming

4.4 Prototyping

Som følge av idefasen blir utvalgte ideer omformet til prototyper. Prototypene trenger ikke nødvendigvis å være komplette produkter, men de skal være tilstrekkelig nok til å teste konseptene. Prototyper gir innsikt i hva som fungerer og hva som gjør ikke gjør det. I de fleste tilfeller vil dette føre til en klargjøring av forbedringspunktene til produktet ved iterasjoner og test. Det er ulike måter man kan prototype på, det kan for eksempel være digitalt eller fysisk. Vi i gruppen lagde flere prototyper gjennomgående i prosjektarbeidet. I utgangspunktet brukte vi ark til prototyping. Det vil si at vi tok funksjonaliteten til konseptet vårt, og skrev det ned på et ark. Dette var et veldig simpeltform for prototyping, men det hjalp veldig med å gi oversikt. Det gjorde det klart hvordan vi kunne knytte alle modulene våres og hvordan systemet helhetlig skulle fungere. I tillegg var dette en flott mulighet til å identifisere klare mangler eller forbedringspunkter i konseptet vi skulle bygge videre på. Deretter etter videre arbeid på konseptet, altså etter vi hadde definert klare moduler kunne vi produsere enda bedre protyper. Dette gjorde vi med for eksempel bruk av 3D-printing. Her kunne vi 3D-printe ulike objekter som kunne hjelpe oss med å simulere løsningen våres.

4.5 Testing og iterasjon

Denne delen av utviklingsfasen har som mål å validere løsningskonsepter, identifisere forbedringsområder, øke forståelse av brukerbehov, gjennomføre iterativ utvikling og bekrefte løsningenes effektivitet. Disse målene ble oppnådd gjennom en strukturert og systematisk tilnærming som involverte flere kritiske trinn. Under utviklingsprosessen gjennomførte vi mange tester og iterasjoner. Vi testet prototypene våres ved å sjekke funksjonaliteten, altså om de hadde ønsket oppførsel eller ikke. Vi tenkte også fortløpende på hvordan vi kunne forbedre løsningen vår samtidig som vi itererte. For eksempel for kjøremønster modulen ble det gjort tydelig hva slags algoritme vi skulle sikte for å lage ettersom vi hadde satt opp en prototype og tok den gjennom

en test. Tilbakemelding vi fikk var kollektivt tankedeling innad i gruppen. Sammen tenkte vi på forbedringspunktene i etterkant av testingene. Deretter gjenstod det å bygge på det vi snakket om gjennom flere tester helt til vi stod igjen men et endelig produkt.

4.6 Integrering av utviklingsmetodikk og designtenkning

Under arbeidet med prosjektet har gruppen vår integrert både utviklingsmetodikk og designtenking for å skape en innovativ og brukervennlig løsning. Denne kombinasjonen har gjort det mulig å gjennomføre prosjektet på en strukturert og effektiv måte, samtidig som vi har hatt fokus på å forstå og møte brukerens behov. Dette har vi gjort ved å alltid ha designtenknings-prinsippene i bakhodet fra start til slutt av prosjektarbeidet. Utviklingsmetodikk og designtenking er noe som står sentralt gjennomgående i arbeidet, og er noe som må bli tatt i betrakning hele tiden. Hvordan vil denne endringen påvirke brukeren? Vil det være positivt, vil det være negativt? Hva slags påvirkning vil dette ha for ytre ringvei? Slik var det vi tenkte under arbeidet for å forsikre oss at en god utviklingsmetodikk og designtenking var tilstede. Ved å også legge til rette for bra samarbeid innad i gruppen ble det lettere å integrere god utviklingsmetodikk. Vi tok i bruk mange gode lagaktiviter innenfor designtenking slik som brainstorming og skuespill. På denne måten har vi sikret god fremdrift og integrert kjernen av designtenking i prosjektarbeidet våres.

5 | Brukerinteraksjon og -orientering

Smart City-prosjekter krever en omfattende vurdering av brukerbase og interaksjon. Det er mange ulike personligheter i Ytre Ringvei, og mange perspektiver å utforske. Denne delen skal forklare hvordan teamet adresserer brukerenes behov håndterer disse problemstillingene.

5.1 Empati og brukerundersøkelser

For å adressere brukerbehov og utfordringer, begynte teamet med å sette seg inn i ulike personligheter med ulike bakgrunner og motiv. Fra dette fikk teamet nyttig informasjon om en potensiell brukerbase og deres spesifikke preferanser og bekymringer. Ved å få innsikt i mangfoldet i bydelen, kunne teamet utvikle mer målrettede løsninger som både forbedret brukeropplevelsen og tilpasningsviljen for ny teknologi. Dette inkluderte tilpasninger for å gjøre teknologien mer tilgjengelig for eldre brukere, tiltak for å sikre databeskyttelse som var avgjørende for personvernbevisste individer, og funksjoner som appellerte til teknologientusiaster. Denne innsikten bidro til å skape en mer inkluderende og effektiv strategi for elektrifisering og digitalisering i byen.

5.2 Definisjon av brukerbehov

De nye vedtakene til kommunen påvirker et bredt spekter av brukere med ulike behov og synspunkter. Dette er en omfattende omstilling for myndighetene, bedrifter og privatpersoner. I prinsippet ønskes det en løsning som gir minst mulig friksjon i brukerens dagligliv, og krever klar kommunikasjon om vedtakenes hensikt. Det er derfor lagt vekt på kommunisering med bruker for å bedre brukerens forståelse for beslutningene. Innbyggere fullelektriske biler skal kunne fortsette dagliglivet som normalt, uten forstyrrelser.

5.3 Prototyping og brukertesting

Grunnet mangler av faktiske brukere ble det heller gjort et skuespill som skulle simulere en brukers reaksjon. Ved å ta i bruk prototyperløsninger slik som 3D-printing brukte vi disse til å simulere brukeropplevelse. Et bilde av et av prototypene våres kan man se under:

Figur 3: Prototype av kjøremodul

Som man kan se har vi brukt prototypen, og satt oss i samme sko som brukerene for å simulere brukertestingen. Deretter simulerte vi interjvuer med brukerne for å hente inblikk og se hvilke forbedringspunkter som var mulig å implementere. Etter å ha hentet inn forbedringspunktene diskuterte vi innad i gruppen hva som kunne vært mulig å implementere og hvordan. Her kom det mange forslag, og det forslaget som fikk mest støtte innad i gruppen ble brukt videre.

5.4 Ferdigstilling og bruk av løsningen

Under ferdigstilling av prototypen passet vi på at vi brukte det vi hadde lært under brukertesting og brukerundersøkelsene for at brukerne av produktet skulle ha en fin opplevelse.

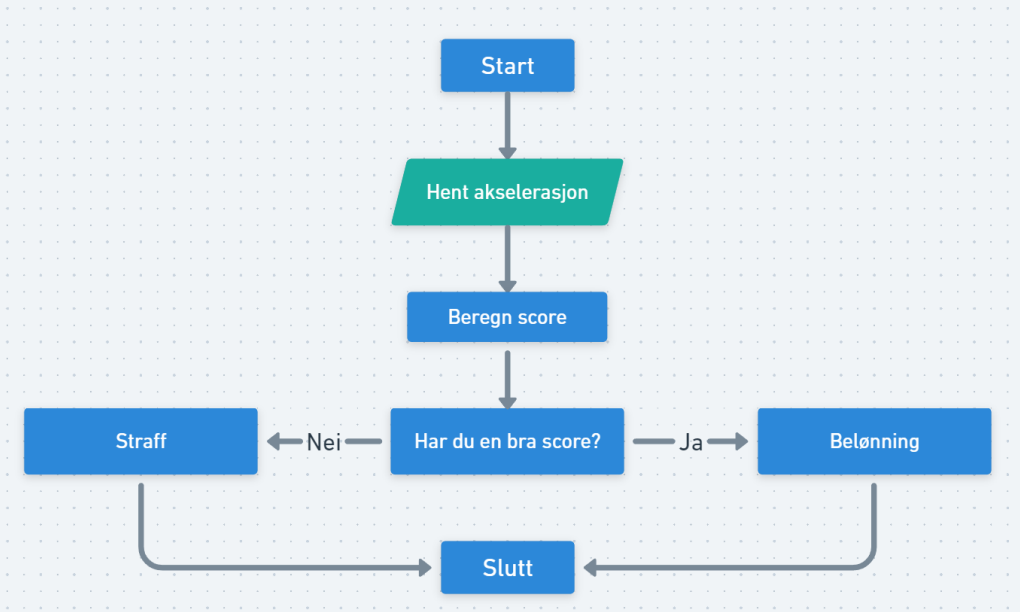
Implementering av systemet er ganske enkelt da alt som trengs alle biler i ytre-ringvei får en kjøremønster-modul i bilene sine for å kunne overvåke kjøremønster. Brukere trenger ikke noen opplæring da systemet ikke trenger brukerinput etter det er satt opp.

6 Prototypeutvikling

6.1 Valg av prototypekonsepter

6.1.1 Kjøremønster

For den første tiltenkte løsningen er det et mål om å kunne hjelpe både innbyggere og innreisende i ytre ringvei. Dette gjøres ved å lage en løsning som kan gjøre hver enkelt bruker klar over sitt eget kjøremønster, som baseres ut ifra hvor miljøvennlig brukeren kjører. Her er det tenkt at det brukes sensorer og programvare internt i bilen, hvor blant annet høy akselerasjon kan bli målt for å danne en profil på den gjeldende brukeren. Denne informasjonen kan dermed videre bli brukt til å lage et poengsystem basert på hvor bra brukeren kjører. Dermed kan brukeren bli gitt incentiver til å opprettholde et godt kjøremønster, ved å for eksempel tilby lavere priser på parkering og andre goder. I tillegg kan et dårlig kjøremønster føre til straff som kan inkludere høyere priser på de samme nevnte godene. Et flytskjema for prototypen er illustrert i figur 2 under:



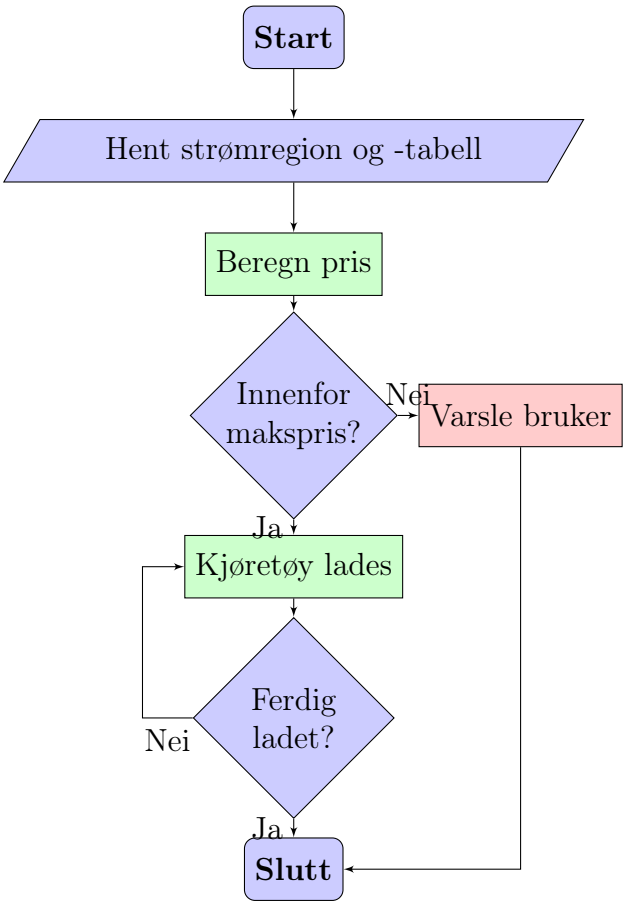
Figur 4: Flytdiagram: Prototype for kjøremodul

For å finne ut om hvorvidt løsningen er mulig å gjennomføre har det blitt sett på flere ulike punkter, blant annet hvor lett det er teknisk å gjennomføre løsningen, hvor mye løsningen kan komme til å koste og hvor mye nytte brukeren kommer til å få ut av løsningen. Med tanke på teknisk gjennomførbarhet virker løsningen veldig gjennomførbar. Dette er fordi det allerede er blitt gjennomført liknende løsninger, både i prosjekter som allerede inkluderer biler, blant annet Tesla som i noen land tilbyr en tjeneste som graderer kjøringen til brukeren med tanke på sikkerhet, men også i enklere eksempler som dronebiler hvor blant annet akselerasjon kan bli målt for mange ulike mål. Det burde dermed være noenlunde trivielt å samle inn noe data som kan brukes til å gradere brukeren med, enten det er med sensorer som er integrert i bilen eller med en selvstendige sensorer. Kravet om å egnede sensorer kan dermed ha en veldig stor effekt på hvor kostnyttig løsningen blir. Dette er spesielt viktig hvis en må ta i bruk en separat modul for sensorene som da må implementeres i hver bil innenfor ytre ringvei. Det kan likevel oppstå noe av den samme effekten hvis det likevel skulle være mulig å kunne bruke integrerte sensorer i alle eller de fleste bilene. Dette kan skje fordi det antakeligvis ikke er de samme sensorene som blir brukt hos alle de ulike biltypene, samt at metodene for å interagere med sensorene også kan variere mye. Denne variasjonen kan dermed føre til økte kostnader for programvaredelen av løsningen, siden en potensielt må produsere flere ulike programvarer for de ulike bilmodellene. Disse to ulike variasjonene av løsningen kan også ha en stor innvirkning på opplevelsen som brukeren kommer til å oppleve. For mens metoden som bruker integrerte sensorer kan gi en potensielt sømløs opplevelse fra brukeren, kan metoden med en separat sensormodul føre til mye mer irritasjon

hos brukeren. Her er det flere ulike punkter i løsningen som kan feile, som da kan skape problemer og usikkerhet rundt hvordan poengsystemet skal virke rundt dette. Derimot, hvis løsningen funker slik den skal, er det tenkt at løsningen vil ha en potensielt stor innvirkning for å løse problemstillingen. Dette er fordi løsningen både gir brukeren innsyn i hvordan den kan bidra til å kjøre mer energieffektivt, samt kan bidra med å gjøre det enklere å gjennomføre denne positive forandringen ved å gi ulike insentiver for bedre kjøring.

6.1.2 Ladesystem

Andre aspekter ved problemstillingen krever en løsning for å avlaste presset på strømnettet under utbyggingen. Dette kan løses ved et smartladningssystem for elbiler; der enheten henter inn strømdata og bestemmer ladetidspunkt og -mengde etter pristabellen. Ladeenheten stopper ladningen når strømprisen overskrider en gitt makspris, og gjennopptar ladningen når strømnettet er mindre belastet, og strømmen er billigere. Denne løsningen vil dermed avlaste press på strømnettet og potensielt gjøre lading billigere for brukeren. Et flytskjema for prototypen er illustrert i figur 5 under,



Figur 5: Flytdiagram: Prototype for smartlader.

Figur 5 over viser konsepter bak lademodulen i enkle trekk. Det legges vekt på å gjøre brukeren klar over dårlig kjøremønster ved varsel. Brukeren får en påminnelse ved tilfelle at kjørevanene fraskrider normalen. Ytterligere dårligere kjøring medfører pristillegg for lading/parkering innenfor bydelen. Denne kommunikasjonen med brukeren kan realiseres med en egen app eller et grafisk grensenitt for ladestasjonen/kjøretøyet og vil være nødvendig for å kommunisere hensikten med løsningen til brukeren. Prototypesystemet starter ved tilkobling av bilen til laderen; strømprisen beregnes etter gjeldende strømregion og gir relevant tilbakemelding til bruker før lading starter/avsluttes.

Selv om det i likhet med forrige løsning er flere prosjekter som allerede har blitt gjennomført før, er dette fortsatt en løsning som er identifisert som mer teknisk krevende. Dette er blant annet fordi det er flere elementer som må jobbe sammen for at det skal være mulig å gjennomføre løsningen. Det må både være mulig å kunne hente lokal data på strømpriser i sanntid, interagere med forrige løsning som kan ha en innvirkning på for eksempel strømpriser, i tillegg til å samhandle med en bil for at det skal være mulig å lade bilen ved de gitte tidspunktene. En løsning for å gjøre dette enklere vil være å bruke eksisterende metoder til å blant annet hente sanntidsdata om strømpriser og for å lade bilen, men også implementere egendefinert logikk for å modifisere disse til å passe inn i den aktuelle løsningen. Dette vil derimot kunne ha en effekt på hvor mye løsningen vil koste. Den høyere kostnaden kan derimot kanskje bli begrunnet med å ta hensyn til hvor mye nytte den aktuelle løsningen kan ha for både en enkeltbruker og for strømnettet i helhet. For enkeltbrukeren vil løsningen, uavhengig av hvordan insentivene ender opp, ende opp med å kunne gi en større besparelse ved å unngå å betale mer enn nødvendig for å lade bilen. I tillegg vil distribueringen av strømbruken kunne ha en stor effekt på strømnettet siden perioder med allerede høy strømbruk kan unngås å se en økning på grunn av lading av biler, samt at perioder der strømnettet har høyere potensial enn det som allerede brukes kan se en bedre utnyttelse av strømnettet. Dette vil dermed føre til at det kan bli lettere å forutse strømbruken på strømnettet, i tillegg til at denne bruken med fordel kan bli jevnet ut til å ta bedre nytte av ressursene tilgjengelig på strømnettet.

6.1.3 Skiltgjenkjenning

Neste løsning prøver å løse problemet med at biler som ikke er hel-elektriske kan kjøre innenfor ytre ringvei, selv om dette ikke skal være tillatt. For dette er en løsning å bruke en kameraløsning som kan gjenkjenne bilkilt og identifisere om bilen er hel-elektrisk eller ikke. Deretter kan en for eksempel bøtelegge bilistene som ikke

kjører hel-elektriske biler. Her er det tenkt at kameraløsningen kan installeres både hos stasjoner ved alle innganger til ytre ringvei for å fange alle bilene som kjører inn i det regulerte området, i tillegg til at det for eksempel kan brukes biler som kjører rundt regelmessig for å fange opp potensielle parkerte ikke hel-elektriske biler innenfor ytre ringvei.

En del konsepter ble vurdert for denne prototypen. For å forenkle problemet kan vi dele det inn i tre deler. Å finne bilkilt i et bildet, og lese av selve bilskiltet, og til slutt finne data om bilen bilskiltet tilhører.

6.1.3.1 bilkskilt indentifisering

Å finne objekter fra et bildet er et godt dokumentert problem. Det første konseptet vårt innebar å trene et nuralt nettverk for gjenkjenning av bilkilt. For å lage denne prototypen trengter vi mye data og prosessorkraft for å få en robust og generalisert ai. Vi tok i bruk noen steg for å redusere den nødvendige dataen og prosessorkraften som trengs. Det mest drastiske tiltaket var at vi trente vår ai på en allerede eksisterende og optimalisert objekt-gjenkjennings nuralt nettverk. Ved å trene på en pre-trent model slipper vår model å lære alt fra bunnen, dette gir mulighet for en velidg sterk og opptimalisert nuralt netverk med mindre data og treningstid en det en model trent fra bunnen hadde vært.

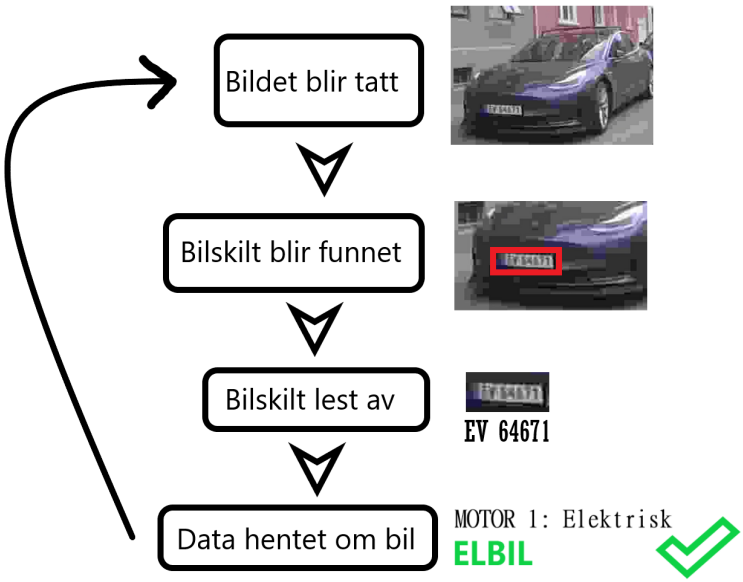
For valg av pre-trent nuralt nettverk var det enkelte ting vi så etter. Vi lette etter et netverk som var lite, robust, generalisert, lett og trene og gitt ut gratis. Vi fant fort objektgjennkjening ai'en yolo-v8 fra Ultralytics som passet perfekt til våre betingelser. Den er lettvekt og har høy ytelse basert på fart og nøyaktighet. Denne modellet trente vi på et datasett fra roboflow som inneholdt litt over 10,000 bilder med bilkilt og kordinater.

6.1.3.2 Lese av bilkilt

Prototypekonseptet vårt for og lese av teksten på bilkilt er basert på på at bilkilt tekst er sekvensiell data. Dette er spesielt relevant for bilkilt ettersom all teksten er på en linje. Dette gjør at vi kan sende pixel data rad etter rad inn i et nuralt netverk. Det nurale nettverket må så ha en arkitektur som funker godt på sekvensiell data som en RNN (recurrent neural network). Dette funker bare ettersom teksten er på en linje og ikke er fordelt over fler linjer.

Det var to hovedkonsepter vi hadde for denne kunslig intelligensen, vi kunne enten trene vår egen ai, på et datasett, eller bruke en eksisterende modell.

For å finne et datasett å trene på trengte vi masse bilder av ord. Vi valgte



Figur 6: Fullt skiltavlsenings system

dermed å bruke et stort datasett på mange gamble tekst-baserte capchas. Capchas er laget for å være utfordrende å lese, så en ai trent på denne dataen ville være veldig robust.

Den andre muligheten vi hadde var å bruke en kunstig intelligens trent på et stort datasett av capchas for å lese av bilkiltene. Å bruke et allerede trent nuralt nettverk sparrer mye tid og prosseseringskraft. Det gir oss også muligheten for å bruke modeller trent på datasett som er større en datasettene vi har mulighet for å bruke under dette prosjektet.

6.1.3.3 Finne informasjon om biler

Etter vi har funnet hva som står på bilkiltene må prototypen vår også finne informasjon om bilene bilkiltene tilhører. For å gjøre dette kan vi benytte oss av statens vegvesen sitt api. Statens vegvesen kan gi oss masse informasjon om de forskjellige bilene. Informasjonen vi bryr oss om er drivstoffet bilene bruker. Ettersom vi bare vil ha fullelektriske biler innenfor ytre-ringvei kan vi bruke data-en fra states veivesen til og sortere ut disse.

Den totale oversikten

Denne løsningen er den som er betydelig mest komplisert og teknisk utfordrende å gjennomføre. Dette er hovedsakelig fordi løsningen implementerer teknisk vanskelige konsepter som kreves for å kunne få til å gjennomføre den likt som det er beskrevet. I likhet med de andre løsningene er det fortsatt noen prosjekter al-

lerede gjennomført, men det kreves fortsatt at det gjennomføres teknisk krevende spesialiseringer for å kunne gjennomføre løsningen. Dette, i tillegg til at løsningen har noen datamessige krevende deler, fører til at løsningen også kan ende opp med økte kostnader. Derimot kan prisen potensielt variere en del basert på hvor stor effekt som ønskes ut av løsningen. For eksempel vil en implementasjon som bare har stasjonære komponent-moduler ved innkjøringer til ytre ringvei ha en potensielt betydelig reduksjon i kostnad, men kan da også ende opp med minket effekt. Derimot vil en utvidelse til å implementere mobile moduler ha en høyere kostnad for komponenter, men da igjen kunne ha effekt som er mer som ønsket for problemet. Hvis det blir bestemt at det skal implementeres straffer i form av bøter kan derimot den høyere kostnaden blir motvirket av at den høyere effekten kan dra inn høyere inntekter i form av bøter. Et problem er at dette også vil føre til et mer negativt inntrykk av løsningen, siden løsningen nødvendigvis må implementere en form for straff. For brukeren vil løsningen kunne bidra positivt, siden den kan bidra til en potensiell inntekt samt å gi brukeren betryggelse om at avgjørelsen om at bare full-elektriske biler skal kunne være innenfor ytre ringvei. Likevel kan denne avgjørelsen føre til økte kostnader for brukeren når offentlige tjenester må gjøre et raskt bytte om til full-elektriske biler. Denne kostnaden kan kanskje reduseres eller splittes over en lengre tidsperiode hvis det blir gjort en bestemmelse om at noen typer av ikke full-elektriske biler på en eller annen måte får lov til å operere innenfor ytre ringvei i en utvidet periode, men dette vil kunne føre til ekstra kostnader på grunn av programvaren da blir mer kompleks, i tillegg til å potensielt kunne skape usikkerhet rundt hva som er tillatt å gjøre på grunn av et mer komplekst reglement. Videre vil det også være vanskeligere for personer som kommer utenifra å vite reglementet til enhver tid, som dermed kan føre til flere misforståelser og liknende, noe som kan være lite ønskelig for brukeren.

6.1.4 Dronebil

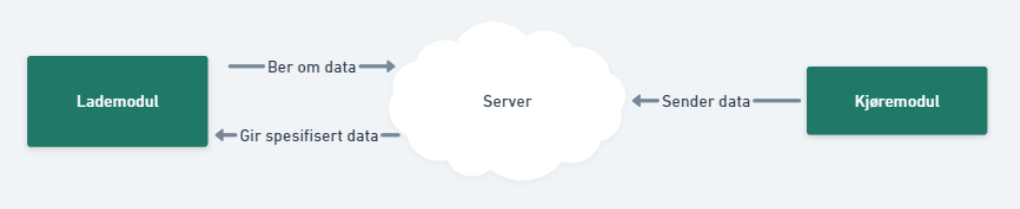
Under oppstarten av prosjektet ble det identifisert et problem med at det kunne bli vanskelig å gjennomføre rask prototyping hvis en skulle prototype på ekte biler og liknende komponenter. Dermed ble en mulig løsning å bruke en robot kalt Zumo32U4, en liten bil som kunne bli brukt til å simulere en personbil. Denne vil gjøre det både lettere og raskere å prototype, blant annet på grunn av at robotbilen lett kan brukes innendørs og samtidig som man utvikler løsningen sin.

Under evaluering av løsningen er det funnet ut at løsningen teknisk sett burde være greit enkel å gjennomføre. Dette er fordi Zumo-bilen inkluderer alt som trengs for å kunne gjennomføre løsningene som kan ha nytte av en prototype-robot. Blant

annet har Zumo-bilen et akselerometer som kan brukes til kjøremønster, i tillegg kan en papirlapp enkelt legges til for å simulere bilskilt for skiltgjenkjenning. Derimot har denne løsningen en bivirkning om at alt må implementeres to ganger, en gang for prototypen av Zumo-bilen og en gang til for den faktiske implementasjonen av løsningene. Dette fører dermed til mer kompleksitet og flere steder hvor feil kan oppstå. Derimot kan løsningen som nevnt redusere kostnadene betydelig, siden det ikke lenger er nødvendig å bruke personbiler for prototypene, men heller bare for den ferdige løsningen.

6.1.5 Server

Til slutt er det et potensielt problem med flere ulike løsninger som kanskje må snakke med hverandre, og hvor alle potensielt må lage sin egen metode for å kommunisere med hverandre. Dermed er det tenkt at det å samle all data fra flere løsninger inn i en og samme server, med standardisert kommunikasjon mellom løsningene og serveren, kan gjøre alt enklere og mer oversiktlig.



Figur 7: Flytskjema for interaksjoner mellom lade-modul og kjøremønster-modul gjennom server

Selv om denne løsningen individuelt sett er ganske kompleks, siden en må produsere en generell måte å kommunisere mellom moduler på, vil faktumet at hver løsning slipper å produsere sin egen kommunikasjonsløsning føre til at denne løsningen totalt sett gir en mindre kompleks løsning. Dette fører også potensielt sett til kost-besparinger, siden mindre tid blir brukt til å implementere kommunikasjon mellom løsninger. Dette kan også føres videre til hvis det senere ønskes å legge til flere løsninger, hvor det dermed blir trivielt å legge til de nye løsningene. For brukeren vil også simpelheten i å legge til ny funksjonalitet føre til en veldig økt brukeropplevelse som blant annet også gjør det lett å eksperimentere med potensielle nye løsninger.

6.1.6 Beslutningsprosess

Siden hver løsning generelt sett kan ses på som separate moduler ble det bestemt å fordele løsningene som er nevnt mellom gruppemedlemmene. Unntaket her er server-

modulen som ble bestemt som kompleks nok til å trenge to personer for løsningen. Dermed ble det mulig å realisere alle løsningene diskutert, med unntak av løsningen med Zumo-bilen siden gruppen hadde tilgang til en personbil som kunne brukes til prosjektet.

6.2 Utvikling og konstruksjon av prototyper

Denne seksjonen detaljerer utviklingsprosessen av modulene.

6.2.1 Utvikling av kjøremønster modulen

For å avgjøre om sjåførene i ytreringvei har et godt kjøremønster blir det brukt et poengsystem til å beregne en score til hver sjåfør. For å realisere denne løsningen ble programmeringspråket Python brukt. Dette valget er basert på hvor enkelt det er å bruke python i samarbeid med raspberry pi-en. Python har et omfattende bibliotek der man blant annet har sense-hat-biblioteket som gir intuitive funksjoner for å kontrollere sensorene. Det første vi måtte gjøre før vi begynte å utvikle kjøremønster koden var å finne ut i hvilket fomat vi ville ha daten fra akselerometere til senshaten på raspberry pien vi brukte. Etter å prøve mange forskellige formater valgte vi å bruke absoluttverdien til akselrasjonsvektoren i andre. Ettersom vi har opphøyd akselerasjonsvektoren i andre vil dette straffe høyere akselrasjon mye sterkere enn det den vil straffe lav akselrasjon. I tilleg sparte dette på prosseseringskraft ettersom vi kunne regne ut akselrasjonsvektoren i andre direkte fra pytagoras. Under ser du hvordan absolutverdien til akselrasjonsvektoren i andre lett blir kalkulert, ax står for akselrasjon i x reting, ay for akselrasjon i y retning, og az for akselrasjon i z reting.

$$\text{Val} = ax^2 + ay^2 + a^2 \tag{1}$$

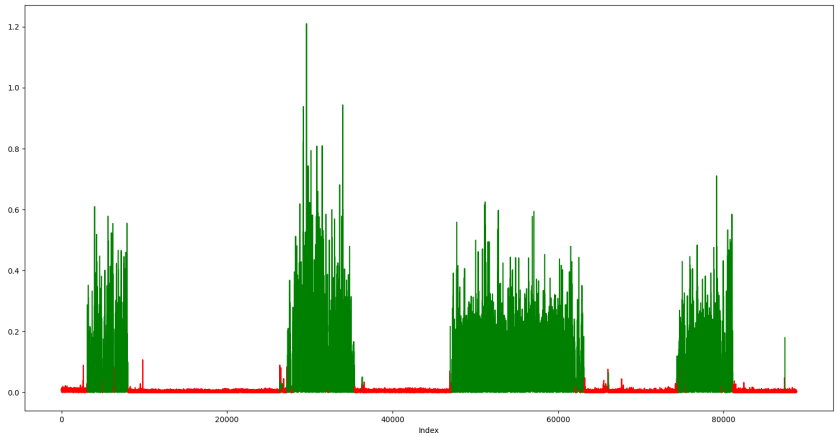
Dette uttrykket kan vi lett implementere og optimalisere i python. Under ser du vår implementasjon av denne funksjonen i python, her har vi også valgt å gjøre antall dimensjoner dynamisk.

```
1 def p(*a):
2     return sum(list(map(lambda x: x**2,a)))
```

Når vi nå har formatet til dataen vi heter fra akselrometeret må vi finne ut av hvor ofte vi vil at dataen skal bli analysert. Grunnen til at vi må ha en konstant antall

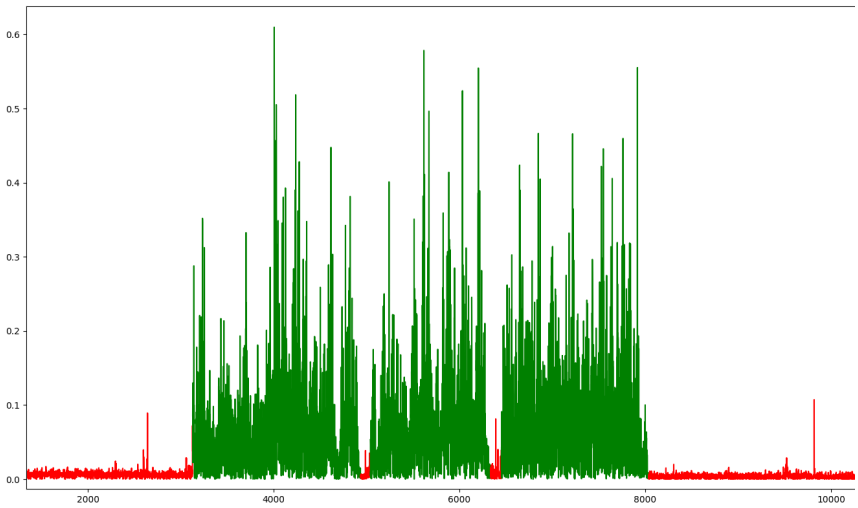
ganger i sekundet dataen skal bli analysert på er for å forhindre at kjøremønsteret raskere endrer seg hvis raspberry pien kører fortere. Hvis vi ikke hadde hatt noe konstant tid og raspberry pien plutselig gjorde noe arbeid i bakgrunnen som gjorde at den kjørte kjøremønster analysen halvparten så raskt, ville den hatt halvparten så stor påvirkning på resten av systemet.

Nå når vi skal begynne å utvikle hoved systemet i kjøremønster modulen trenger vi litt data å analysere. Ettersom at vi har funnet formatet vi vil ha akselrasjonsdataen vår på kan vi bruke denne informasjonen til å samle data vi senere kan analysere. For å gjøre dette måtte vi sette en høy men stabil frekvens til insamlingen av akselrasjonsdataen. Vi valgte 10 ganger i sekundet under henting av daten. Under ser du daten samelet over en to og en halv time lang periode.



Figur 8: Total rådata for analyse

Denne dataen har vi allerede analysert litt før denne visualiseringen. For å unngå at kjøremønster verdien endres når bilen står stille fjærnet vi data som ble hentet når bilen sto stille, i visualliseringen er denne daten vist i rødt. Hvis vi zoomer inn på den første bulken av dataen med aktiv kjøring kan vi fort se hvorfor dette ble impelementert.



Figur 9: Rådata for analyse under bykjøring

Denne bulken med data ble innsamlet under bykjøring. Under kjøringen måtte vi stoppe på to røde lys. Under disse røde lysene sto bilen stille, i denne tiden hadde vi naturligvis null akselrasjon. Dette førte til at det virket som bilen kjørte unaturlig fint med ingen bevegelse. I demoene våre hvor stasjonær data ikke ble fjernet gikk dermed kjøresmønster verdien sterkt oppover i disse periodene. Dette belønnet kjøremønstere som stoppet ofte, noe som ikke tilsvarer et fint kjøremønster. Etter vi fjernet data innsamlet når bilen sto stille hjalp dette i stor grad på å forbedre nøyaktigheten til kjøremnønsteret. Dette gjorde også at man ikke lenger fikk bedre kjøremønster av å stoppe ofte eller å stå stille generelt.

Selvom vi vil analysere dataen en hvis antall ganger i sekundet, kan vi fotstatt ha den største nøyaktigheten systemet klarer til en hver tid. For å forklare hvordan må vi første finne hvor ofte akselrometer dataen skal analyseres. Vi fant ut etter prøv og feiling at det beste var å analysere dataen 2.5 ganger i sekundet, dette vil si 0.4 sekunder forsinkelse mellom hver analyse. Når du kjører skriptet henter den først ut tidspunket den begynte på, så henter den knstant data fra akselrometeret. Denne dataen blir gjort om til formatet vi vil ha det på samtidig som det blir hetnet. Etter 0.4 sekunder har gått vil den så analysere daten ved å ta gjennomsnittet av dataen hentet fra akselrometeret i peridoen siden siste analyse. Dette systemet gjør kjøremønster deteksjonen har full nøyaktighet og ikke endres utifra ytelseskraften til raspberry pien. Selv i situasjoner hvor bilens akselrasjon endres brått innenfor de 0.4 sekundene vil veriden være riktig. La oss se på et eksempel.

La oss si en bil kjører rett fram i x retning, den har null akselrasjon i hverken y eller z retning. I løpet av 0.4 sekunder gasser bilen på så den får høy x akselrasjon før den brå bremser og får negativ x akselrasjon. Vi kan tenke oss at akselrasjonen

målt i meter per sekund i løpet av denne periden ser slik ut:

$$ax = [4, 3, 2, 1, 0, -1, -2, -3, -4] \tag{2}$$

I vår eksempel tokk raspberry pien 9 målinger fra akselrometeret i løpet av disse 0.4 sekundene. I vårt format blir verdiene da.

Vi bruker dette formatet:

$$a = ax^2 + ay^2 + az^2 \tag{3}$$

ay og ax er null

$$ay = az = 0 \tag{4}$$

ax er en bil som først akselererer framover så bakover

$$ax = [4, 3, 2, 1, 0, -1, -2, -3, -4] \tag{5}$$

Første element i akselrasjonen på vårt format blir da

$$a[0] = 4 * *2 + 0 * *2 + 0 * *2 \tag{6}$$

Akselrasjonen på vårt format i perioden blir

$$a = [16, 9, 2, 1, 0, 1, 4, 9, 16] \tag{7}$$

Verdien vi regner med videre i analysen blir

$$avg(a) = 6.67 \tag{8}$$

Her ser du tydelig fordelen med at formatet vårt er absoluttverdien til akselerasjoen i andre. De høye akselerasjoenene som i vårt eksempel rask gass etterfulgt av brå brems vil påvirke akselerasjonsverdien kraftigere å føre til at målingen vår i perioden er høy selv med perioder uten akselrasjon i målingene. Det fører også til at den positive og negative akslerasjonen ikke kanslerer hverandre ut og dermed blir du alltid straffet i full grad for dårlig kjøring, selv i situasjoner hvor perioden med dårlig kjøring er mindre enn analyseintervallene.

Koden for denne delen av systemet kan du se under.

```
#Funksjon for å få akselrometer dataen på vårt format
def p(*a):
    return sum(list(map(lambda x: x**2,a)))

while True: #loop foraltid

    a = sense.get_accelerometer_raw() #hent data fra senshatten

    temp_s = p(a['x'],a['y'],a['z']) #gjør om dataen til vårt format
    temp_arr.append(temp_s)  #lagrer den midlertidig

    if time.time() > start + 0.4: #vent 0.4 sekunder etter siste analyse

        start = time.time() #når siste analyse ble gjort blir optatert

        s = sum(c)/len(c) #s blir gjennomsnittet av alle midlertidig lagrede verdier

        #KJØREMØNSTER OPPTATERING

        temp_arr = [] #gamel data blir slettet
```

Nå som vi har hentet data vi kan analysere og valgt hvordan format dataen er gitt på samt hvilken frekvens den analyserte daten kommer på, kan vi begynne på algoritmen som skal lage og endre kjøremønster scoren.

Det er en del ting vi må tenke på når vi skulle lage denne scoren. Vi begynte med å velge en max score og en minimum score. Verdiene vi valgte for dette var 0 og 100. Med andre ord, kjøremønster scoren må være en verdi mellom 0 og 100. Etter dette måtte vi bestemme hvor vi hvilken verdi vi ville gi den gjennomsnittlige sjåfør. For dette valgte vi 70. Tanke gangen bak denne verdien er at det er mindre forskell på energi spart ved å gjøre perfekt i fårhold til den gjennomsnittlige sjåfør iforhold til energi tapt ved en forferdelig sjåfør. Denne verdien er kjøremønsterscoren vi gir nye sjåfører. Vi forestilte oss også at kjøremønster dataen vi hadde samlet var en fin representasjon av en gjennomsnittlig sjåfør.

Når det kommer til selve kjøremønster-score algoritmen var det tidlig tydelig at vi ville ha et ulinjert system. For å forklare dette kan vi bruke et overdrevent eksempel. La oss se for oss at det skjer en hendlese, foreksempel en brå brems. Raspberry pien plukker opp den høye akselrasjonen og kjøremønsterscoren skal bli straffet. La oss så si at en person med 70 i score får en straff på 1, med andre ord scoren deres ender opp som 69. En anen person derimot med en kjørescore på 20, er ikke en så brå brems så unormalt i kjøremønsteret deres, dermed vil de bare få en straff på rundt 0.2 og ende opp med en kjøremønsterscore på 19,8. I andre siden av spekteret vil en person med høy kjøremønster score foreksempel 95 få en høyere staff ettersom finere kjøring er forventet av en person med så høy score. Deres score kan foreksempel falle ned til 93. Dette fører også til at hvis personen med 70 i score gjentar denne bråbremsen 100 ganger vil de ikke ende opp med en score på -30, men heller en score på rundt 35.

For å forklare algoritmen vi endte opp med er det lettere og først vise det ferdige produktet og så vise hvordan vi har tenkt. Under ser du python implementasjonen av algoritmen vår hvor ser den formaterte akselerasjonsdataen og kj_scoreer kjøremønster scoren.

```
1 kj_score += max(-0.03, s**((kj_score+130)/200) - 0.1)/1000
```

Denne python linjen er i analyse delen av kjøremønsteranalyse scriptet og kjører dermed 2,5 gang i sekundet. Selve matten er ikke så komplisert. Vi begynner med å ta ssom er den formaterte akselrasjonen å skalere den. Måten vi skalerer den på er å opphøye den i den (nåværende kjøremønster-scoren + 130)/200. Dette gjør at hvis scoren er på 70, blir akselerasjonsdata-en opphøyd i 1, med andre ord den vil ikke endres. Hvis scoren er på 100 derimot vil akselrasjonen bli opphøyd i 1.15, og hvis scoren er 0 vil den opphøyes i 0,65. Eksempel på hvordan akselerasjonsdata-en blir skalert ved de forskjellige kjøremønster scorene ser du under.

Vi ser for oss at den rådatene fra akselerometeret er ax = 4, og ay = az = 0. Akselerasjonsdataen blir dermed

$$s = ax^2 + ay^2 + ax^2 = 4^2 + 0^2 + 0^2 = 16 \tag{9}$$

Ved å bruke skalerinsformlen:

$$new\_s = s^{((kj\_score+130)/200)} \tag{10}$$

For en person med kj_score på 100 blir akselerasjonsdata-en skaler til:

$$new\_s = s^{((100+130)/200)} = s^{1.15} = 16^{1.15} = 24.25 \tag{11}$$

For en person med kj_score på 70 blir akselerasjonsdata-en skaler til:

$$new\_s = s^{((70+130)/200)} = s^{1} = 16 \tag{12}$$

For en person med kj_score på 0 blir akselerasjonsdata-en skaler til:

$$new\_s = s^{((0+130)/200)} = s^{0.65} = 16^{0.65} = 6.06 \tag{13}$$

Fra disse eksemplene kan du tydelig se hvor mye vanskeligere det er å holde

en høy kjøremønster-score som fører til at bare de konsistent bra sjåfør for høy score og de konsistent dårlige sjåfør for en lav score.

Denne nye skalerte akselerasjonsverdien påvirker ikke kjøremønster scoren direkte. Først trekker vi fra 0.1 før vi velger max verdi av den nye verdien og negativ 0.03 og så skalerer endringene for å unngå store svingninger i kjøremønster scoren. Formlen for dette kan du se under, vi setter inn SCALE for verdien som skalerer, i vårt produkt er denne verdien 1000.

$$kj\_score+ = max(-0.03, new\_s - 0.1)/SCALE \tag{14}$$

Hvis new_s har en verdi på 0 vil kj_score oppdateres slik:

$$kj\_score+ = max(-0.03, 0 - 0.1)/SCALE = -0.03/SCALE \tag{15}$$

Hvis new_s har en verdi på 0.05 vil kj_score oppdateres slik:

$$kj\_score+ = max(-0.03, 0.05 - 0.1)/SCALE = -0.03/SCALE \tag{16}$$

Hvis new_s har en verdi på 0.1 vil kj_score oppdateres slik:

$$kj\_score+ = max(-0.03, 0.1 - 0.1)/SCALE = +0 \tag{17}$$

Hvis new_s har en verdi på 1 vil kj_score opptateres slik:

$$kj\_score+ = max(-0.03, 0 - 0.1)/SCALE = +0.9/SCALE \tag{18}$$

Det som er verdt å legge merketil her er at kjøremønster-scoren bare øker hvis new_s har en verdi på under 0.1. Denne verdien kan du se beregnet under, vi ser fortstat for oss et eksempel hvor vi bare bruker akselrasjonen i x retning, med andre ord ay = az = 0:

Vi ser for oss an brukeren har 70 i score, formlen for new_s og a blir dermed den samme:

$$new\_s = a = ax^2 + ay^2 + az^y \tag{19}$$

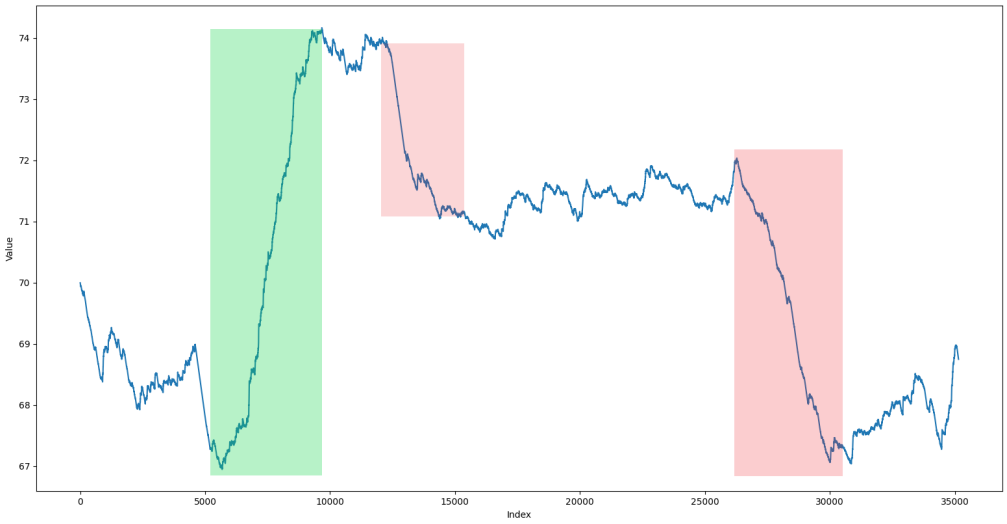
$$0.1 = ax^2 + 0^2 + 0^2 -> ax = sqrt(0.1) -> ax = 0.32 \tag{20}$$

Dette vil si at hvis den totale akselrasjonen på bilen er på under $0.32 m/s^2$

vil kjøremønster scoren forbedres, og ellers vil den bli lavere. Dette vil også si at hvis du har en konstant akselrasjon på $0.32m/s^2$ vil du tilslutt ende opp med en kjørescore på 70.

En annen ting å legge merke til med algoritmen vår er at alle new_s scores på 0.07 til 0 har like stor påvirking på scoren. Dette har vi valgt å gjøre for å forhindre at folk kjører bilen rundt sakte for å forbedre kjøremønster scoren sin. Maksimum belønning for kjøremønster skal kunne oppnåes i normal hastighet. Dette forhindrer at folk kan kjøre tregt rundt i kort tid og utnytte den lave akselrasjonen man har i lav fart.

Under ser du en graf over kjøremønsteret generert fra eksempel daten vi samlet inn, når vi samlet denne daten hadde vi en fase hvor vi prøve og kjøre så fint som overhodet mulig, og en hvor vi kjørte med bråe bevegelser for å senere kunne finne disse i kjøremønster score daten senere. I eksempelet du ser under har vi også satt SCALE til 10 isteden for 1000 så endring i kjøremønster scoren skal være tydeligere.



Figur 10: Kjøremønster score til eksempeldata, god kjøring markert i grønn, dårlig i rød

6.2.2 Utvikling og valg av teknologi for lademodulen

Python ble valgt som hovedspråk for utviklingen av prototypen. Dette valget ble gjort på grunn av Pythons rike økosystem og tilgjengelighet av mange tredjepartsbiblioteker som lett integreres med ulike API-er[3]. Tilgangen til bilioteker for å håndtere HTTP-forespørsler, nettverkskommunikasjon og håndtering av tid- og posisjonsdata var nødvendig under utviklingen av programmet.

[3]Engelsk: *Application Programming Interface.*

Materialer og designverktøy

- Maskinvare: For test- og utviklingsmiljøet brukes en standard datamaskin i tillegg til ettkortsdatamaskinen Raspberry Pi 3B for å undersøke hvorvidt programmet kjører på enklere systemer og varierende nettverksforhold.
- Utviklingsmiljø: Produksjon og feiltesting av kode ble utført i PyCharm, et integrert utviklingsmiljø laget for utvikling i Python.
- Biblioteker:
  - `requests` for å håndtere HTTP-forespørsler og kommunisere med tredjeparts API-er.
  - `socket` for å etablere nettverksforbindelser og kommunisere med servermodulen.
  - `pickle` for å serialisere data som sendes over nettverket.
  - `datetime` for beregning av dato og tid.
  - `geocoder` for henting av geolokasjonsdata basert på IP-addrese.
- API-er
  - Kommuneinfo fra Kartverket for å bestemme fylke og strømregion etter koordinater levert av `geocoder`.
  - Strømpris API fra Beneficial Apps[4] for tilgang til strømtabell i JSON-format.

Integrering av komponenter Integreringen av komponentene til en fungerende prototype ble gjort i flere trinn. Ved oppstart blir lengde- og breddegrader hentet ved hjelp av `geocoder`-biblioteket og IP-adressen, posisjonsdata sendes med en HTTP forespørsel til Kartverkets API for å bestemme hvilket fylke brukeren befinner seg i; deretter bestemmes strømregionen etter fylkets gjeldende strømkode. Strømprisen reguleres i etterkant etter kjørescoren bestemt av kjøremodulen, denne hentes fra serveren via nettverksforbindelse med `socket`.

6.2.3 Bilskilt avlesning

Utvikling av bilskilt avlesning systemet startet med konstruksjons av en lokal demo for å teste påliteligheten og finne eventuelle problemer med det tenkte systemet. Den første delen av av utviklingen gikk ut på trening av den kunstige intelligensen systemet benytter seg av.

[4]https://www.beneficial.no, https://www.hvakosterstrommen.no/strompris-api

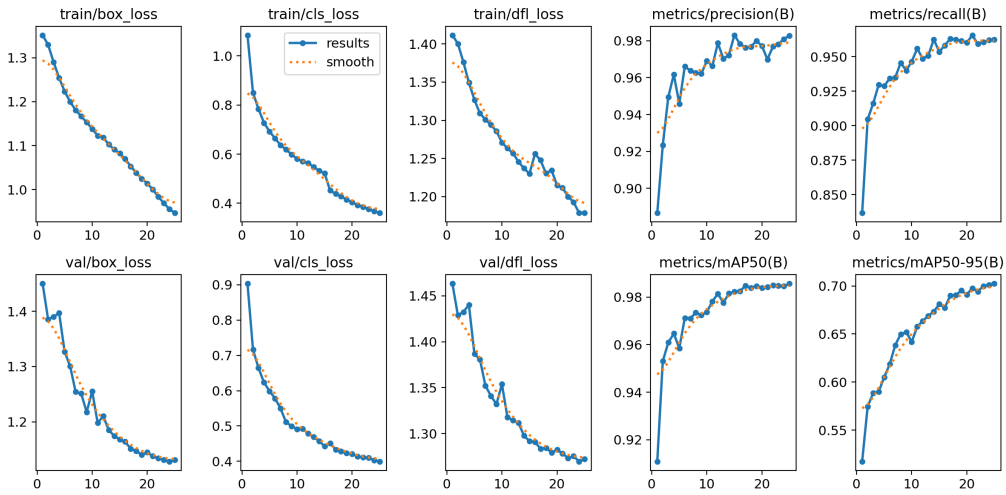
6.2.3.1 Utvikling av skilt-gjenkjenning ai

Trening av skilt-gjenkjennings ai'en var det første vi begynte med, ettersom å trene en objektgjennkjening's ai er en prosess som tar lang tid. Vi valgte å bruke den pre-trente modellen YOLO-v8 og trene vår ai på denne for å forminske prosesseringskraft, tid og datasett som trengtes under trening. Datasette vi valgte heter license-plate-recognition-rxg4e_dataset"og inneholder 10,126 bilder gitt ut av robolfow. Fra roboflow kan vi laste ned informasjonen om hvor i bildet de forskjellige skiltene er i samme format som det YOLO-v8 bruker, samtidig som vi laster ned bildene. Yolo-v8 har også mye god dokumentasjon vi kan bruke for ettertrening av modellen.

For å kunne trene ai'en på samme bilder fler ganger enderer vi bildenne litt hver gang modellen ser bildene under trening. Endringene kan være små, som å zoome ut eller inn, eller så kan de være store, som oppkutting og sette sammen fler bilder, eller andre endringer sålenge bilskiltene fortsatt er synlig. Under ser du et typisk bildet av hvordan bildene ser ut når modellen trener på de.

Figur 11: Bildene under trening

Disse endringene i bildene, samt det store datasettet tillater oss å trene modellen på hele datasettet 25 ganger. Etter 25 ganger så det ikke ut som presisjonen til modellen forbedret seg. Denne trengingsprosessen tokk rundt 254 timer, rundt 10 timer perr gjennomgang av datasettet. Under ser du data av loss"og precition"av modellen for hver gjennomgang av datasettet.



Figur 12: Nøkkeltall fra trening

Dataen er delt inn i tre deler, de som begynner på Trainer dataen fra selve treningsprosessen på treningsdataen. Ettersom modellen raskt blir ekstremt god på de bildene den trener på har man også noe som heter et validerings sett. Valdieringssettet er bilder som ai'en aldri har sett før. Data hentet fra validerings settet er ofte det man fokuserer mest på, og er i vår visualisering begynner disse grafene på "Val". Val/box_loss sier hvor tett"området modellen tipper bilskiltet er fra hvor bilskiltet faktisk er. Den vil gi en høyere verdi desto større feilen er, og du kan tydelig se filen går nedover etterhvertsom den trener. "Val/cls_loss"måler filen av klassifiseringen til hver forutsagt grenseboks. De forskjellige klassifiseringene vi har er bakgrunn eller bilskilt, så denne vil naturligvis ligne veldig på "val/box_loss". Resten av målingene (de som begynner på "metrics") er alle forskellige måter å måle presisjon på. "Metrics/precision(B)"og "metrics/recall(B)er de to viktigste. Precision(B) måler presisjonen, at den har en presisjon på 98% tilsier at den tar rett 98% av gangene den tipper. Recall(B) er hvor ofte den klarer og identifisere alle skiltene i et bildet. Ettersom at flesteparten av bildene i vårt datasett bare inneholder ett bilskilt ligner disse to grafene i stor grad. All dataen under "metricser generert fra validerings datasettet. Under ser du et eksempel på modelen som predikterer hvor bilskiltene er i valideringsdatasettet.

Figur 13: Prediksjoner fra valideringsdatasettet

Noe å legge merke til med disse prediksjonene er at den klarer fint å prediktetre bildet nederst til høyre, visualisert igjen under.

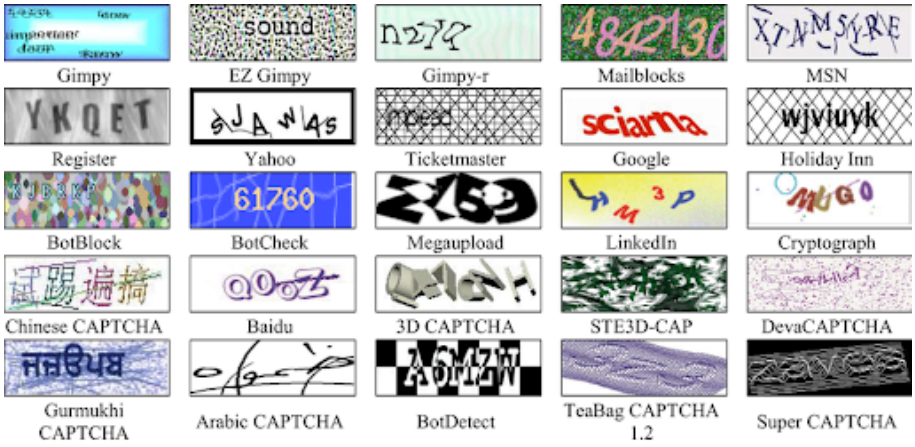
Figur 14: Forstørret bildet

I dette bildet ser ikke bilskiltet ut som et typisk bilskilt, men ettersom vi tydelig ser fronten på en bil kan modellen utifra konteksten fortsatt prediktere bilskiltet. Dette er et mønster modellen vår mye fortere plukker opp ettersom den allerede før vi begynte vår trening hadde mulighet til å kjenne igjen biler. Å få samme mønstergjennkjenning i en modell trent fra bunnen hadde tatt mye lenger tid, prosesseringskraft og større datasett.

6.2.3.2 Utvikling av skilt-avlesnings ai

Når vi nå har funnet bilskiltene fra bildene må vi så lese av bilskiltene. Dette trenger en ny algorimte for avlesning av bilder og vi har da benyttet oss av en RNN (recurrent neural network). RNN er et nuralt netverk arkitektur som fungerer veldig bra på sekvensiell data. Grunnen til at dette funker så bra i vårt brukertilfellet er at teksten vi skal lese bare er på en linje. Dette gjør at en modell som leser rad etter rad bortover på bildet og konstant husker det den har lest, vil kunne lese all dataen en bokstav om gangen. Vi kan også gjøre RNN'en multi direktionell hvor den da leser bildet fra begge sider samtidig.

Når vi nå har bestemt oss for en arkitektur må vi også finne et datasett og trene på. Datasettet vi valgte å trene på er gammeldags tekst-basert captchas. Under ser du eksempler på Captchas. Datasettet vi brukte innehold bare skandinaviske tall og bokstaver.



Figur 15: Eksempel Captchas

Etter modellen var trent på våre begrensede resurser sto den forstatt ikke helt opp til forventningene våre. Ettersom Capcha breaking er et vellkjent problem fant vi fort en modell med lignende arkitektur, trent på et mye større datasett over mye lenger tid. Modellen Text_Captcha_breaker"av docparserpå huggingface var modellen vi endte opp med. Vi testet denne modellen ved å sende bilskilt og den klarte med enkelhet alle testene våre med 100% nøyaktighet. Denne modellen var også relativt raskt noe som vi kan utnytte til å preprossesere bildene på forskjellige måter før den analyserer de. Dette gir modellen enda høyere nøyaktighet og gjør den enda mer robust.

6.2.3.3 Utvikling av algoritme for henting av bildata

For å finne ut om bilene er elbiler eller ikke etter vi har funnet ut hva som står på

bilskiltene kan vi bruke statens vegvesen sitt api. Statensvegvesen gir oss masse informasjon på bilene bilskiltene tilhører. Denne informasjonen inneholder veldig mye data, men den dataen vi trenger og skal fokusere på er hvordan drivstoff motorene i bilen bruker. Denne informasjonen henter vi ned som en liste ettersom biler kan ha flere motorer. Vi kan dermed lage en funksjon som gir oss en verdi på Falseeller True"utifra om bilskiltet vi bruker sominput i funksjonen er fullelektrisk eller ikke. Denne funksjonen var relativtlett og implementere og oppsettet kan du se under.

```
def get_sign(what):
    url = ".../kjoretoyoppslag/v1/oppslag/raw/"  + what
    data = json.loads(requests.get(url.strip(), headers=headers, timeout=10).text)
    data = data['kjoretoy']['godkjenning']['tekniskGodkjenning']['tekniskeData']['motorOgDrivverk']['motor']

    elektrisk = True
    for motorer in data:
        for item in motorer['drivstoff']:
            if item['drivstoffKode']['kodeNavn'] != "Elektrisk":
                elektrisk = False

    return elektrisk

get_sign("EV64761")
get_sign("KH68603")
get_sign("EV6476i")

#>>> True
#>>> False
#>>> True
```

Denne funksjonen er simplifisert og hentet ut fra get_data.py i vedleggene. Denne ekesempelfunksjonen viser hvor lett det er og hente dataen fra statens vegvesen. Noe å legge merketil er testene vi bruker for å forsikre oss om at funksjonen funker slik den skal. Her tester vi første en elbil med kilt EV64671"og ser at den gir verdien Truesom vil si at det er en elbil. Etter dette tester vi med et nytt kilt som ikke er elbil og får verdien False", så systemet funker slik det skal utifra vår lille eksempeltest. Etter disse to tester vi også med nummerskilt EV6467i"med andre ord vi bytter ut det ene "1tallet med en "i". Vi ender fortsatt opp med å få info-en tillhørende elbilen EV64671", dette er fordi statens vegvesen har på sinn side rettet for simple feil som "i"isteden for "1"og "O"isteden for "0". At vi har denne feilmarginen gjør det lettere å for systemet vårt å finne dataen tilhørende biler og forminsker også prossesering av data vi ellers måtte ha gjort på vår side.

Koden du ser over har ikke implementer cache. "Cacheeller bufferer en måte å lagre data lokal på for å minske presset på statens vegvesen sine servere. Dette er spesielt nyttig i vårt prosjekt ettersom et bilskilt fort blir med på flere bilder når bilen først er foran kameraet. Vi utviklet en cache som holder på dataen i 1 dag. Med andre ord når vi henter informasjon om en bil er elbil eller ikke så blir dette lagret i

en dag. Den ferdige koden med implementert cach kan finnes i get_data.py"under vedlegg.

6.2.3.4 Raspberry pi klient side

Nå som modelene er trent på resten av systemet settest opp. Det vi begynner med er der bildene kommer fra, nemelig et kammera koblet opp mot en raspberry pi. Ettersom raspberry piene vi bruker ikke er raske nok til å kjøre objektgjennkjennings modellen live kjører vi denne på server siden. Raspberry pien sin jobb blir dermed enklere og kan deles inn i konkrete steg.

- Henter bildet (ta bildet fra kameraet)
- Lager et komprimert bildet og gjør det klart for sending
- Sender det komprimerte bildet til serveren og får lokasjon av bilskilt tilbake
- Sender bilskiltene fra det ordinalet bildet
- Henter bildet (begynner på nytt)

Det første steget er relativt simpelt. Det er mye fin dokumentasjon for hvordan man skal koble opp og ta bilder med et kamera ved å bruke raspberry pi. Når raspberry pien får strøm begynner med en gang og jobbe. Den starter på en loop hvor det første steget er å ta et bildet, la oss kalle dette bildet1. Bildet1 kommer rett fra kameraet og ingen prosessering er gjort på det (det har høyeste kvalitet kameraet klarer). Det første raspberry pien gjør er å lage en klone av dette bildet som er litt mindre (sklaerer det ned så bredden på bildet er 800 px). Dette gjøres fordi modellen vi har for objektgjennkjenning modellen vår ikke kan godta bilder over 800px brede så denne dataen hadde ikke blitt brukt uansett. La oss kalle dette nye bildet bildet2. Bildet2 blir så komprimert med en ekstremt sterk jpeg kompresjon. Denne kompresjonen er veldig lossy". Dette betyr at mye data går tapt under kompresjonen, men det viktigste er at objektene på bilde2 fortsatt er synlig. Etter att bildet2 har blitt send til serveren får raspberry pien svar tilbake med lokasjonen til bilskiltene. Dette gjøres ettersom bilskiltene på bildet2 somoftest har blitt uleselige etter den kraftige kompresjonen. Etter raspberry pien har fått lokasjonen på bilskiltene kan den hente ut disse lokasjonene fra bildet1 og sende dette til serveren. Dette fører til til at serveren har høykvalitet bilder av bilskiltene den så kan lese av.

6.2.3.5 Skilt-gjenkjenning server side

Når vi utviklet server siden var et viktig punkt at den skulle ha mulighet for å ståtte flere forskjellige kamera-klienter samtidig. Dette gjør vi for å optimalisere minnebruk ettersom en server da kan holde begge de nevralet nettverkene i minne konstant og kjøre bilder fra forskjellige kameraer gjennom nettverkene i paralell. Dette sparer mye prosessorkraft å gjør at hvert kamera kan sende bilder til serveren opp til 15 ganger i sekundet før det er serveren sin prosessorkraft som forsinker resten av systemet. skilt-gjenkjenning serveren sin oppgave er delt inn i to deler. Hvilke del som kjører kommer ann på hvordan data som serveren mottar.

Den første delen kjører når serveren mottar et lavkvalitet bildet fra en kamera klient, før referert til som bildet2. Hvis servere mottar et slik bildet blir den analysert av modellen som finner bilskilt. Koordinatene og størrelsen på bilskiltene blir så skalert opp så de matcher bildet1 og blir sendt tilbake til raspberry pien.

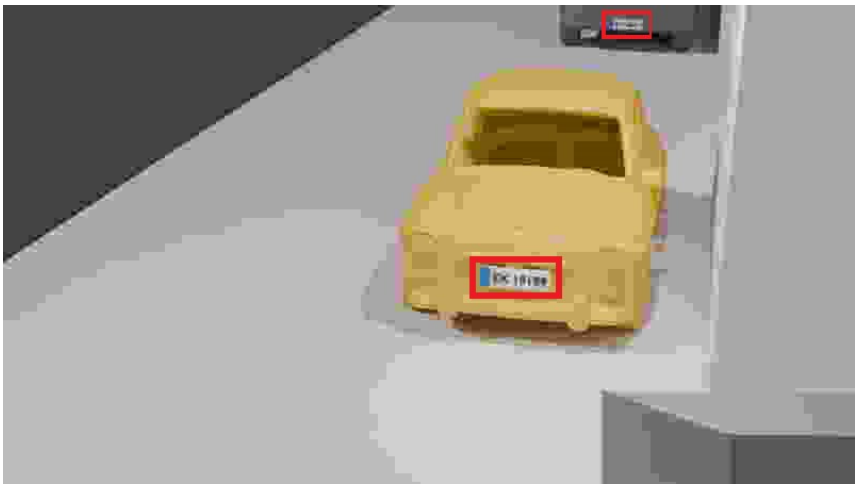
Den andre delen kjører bare når serveren mottar høykvalitet bilder som bare inneholder bilskilt. Disse bildene kommer ofte fler om gangen ettersom et bildet2 kan inneholde fler bilskilt. Disse bilskiltene blir derreter lest av. Etter det sjekker vi statens vegvesen for informasjon om bilene, og hvilke biler som er elbiler eller ikke. I vår prototype har vi valgt å bruke denne informasjonen til å visualisere informasjonen vi får fra server siden. Derhvor de høykvalitet bildene er satt på lavkvalitet bildene og en grønn farge er gitt til biler registrert på statens vegvesen som fullelektriske. Denne visualiseringen kan du se under når vi går gjennom det totalet systemet.

6.2.3.6 Full konstruksjon av bilskilt avlesnings systemet

Under ser du et eksempel på hvordan systemet funker, vi begynner med at raspberry pi (kamera modulen) tar et bildet.

Figur 16: Steg 1. kameraet tar bildet

Figur 17: Steg 2. Bildet blir komprimert og sendt



Figur 18: Steg 3. Server svarer med lokasjon av bilskilt

Figur 19: Steg 4. Raspberry svarer tilbake med bilskilt

Figur 20: Steg 5. Server mottar og analyserer bilskiltene (her visualisert all dataen serveren har mottatt, høykvalitet bildene lagt oppå lavkvalitet)

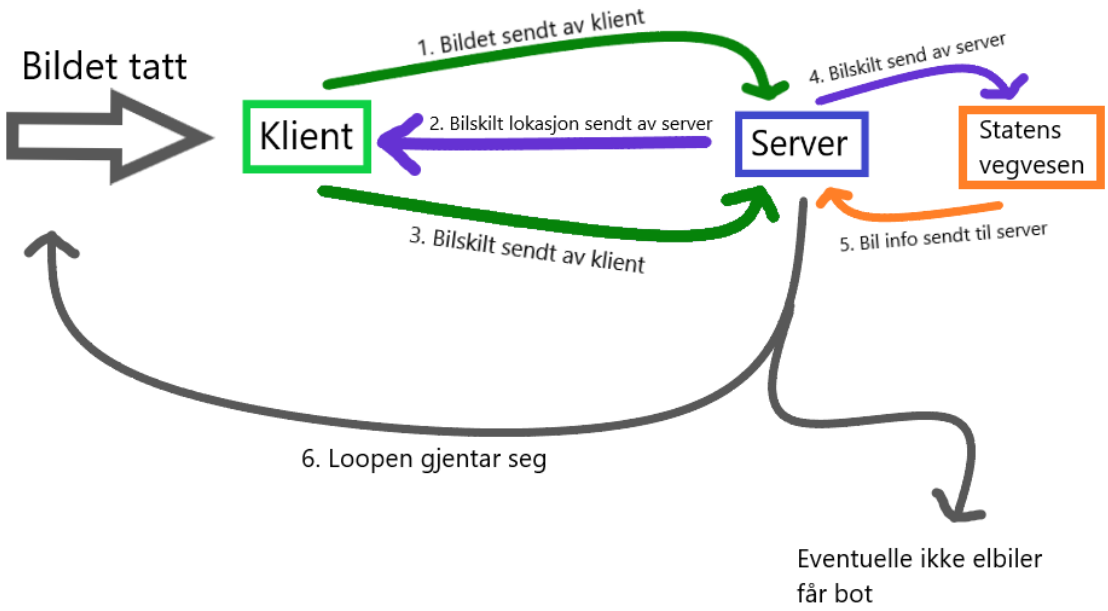
Etter disse 5 stegene har gått gjentar begynner systemet igjen fra steg 1. Under ser du utklipp tatt fra servers siden under noen av testene våre. I de tidlige fasene brukte vi demo bilen for å kunne teste systemet raskt, og med minst mulig bruk av resurser. Denne demobilen hjalp mye under utviklingen for lettvinthet og produksjonshastighet.

Figur 21: Serverside visualisering med demo bilen

Figur 22: Serverside visualisering med fullelektrisk elbil

- 

Figur 23: Serverside visualisering med ikke fullelektisk bil (sensurert når bildet ble hentet grunnet manglende samtykke fra eier)



Figur 24: Flytskjema over hele bilskilt systemet

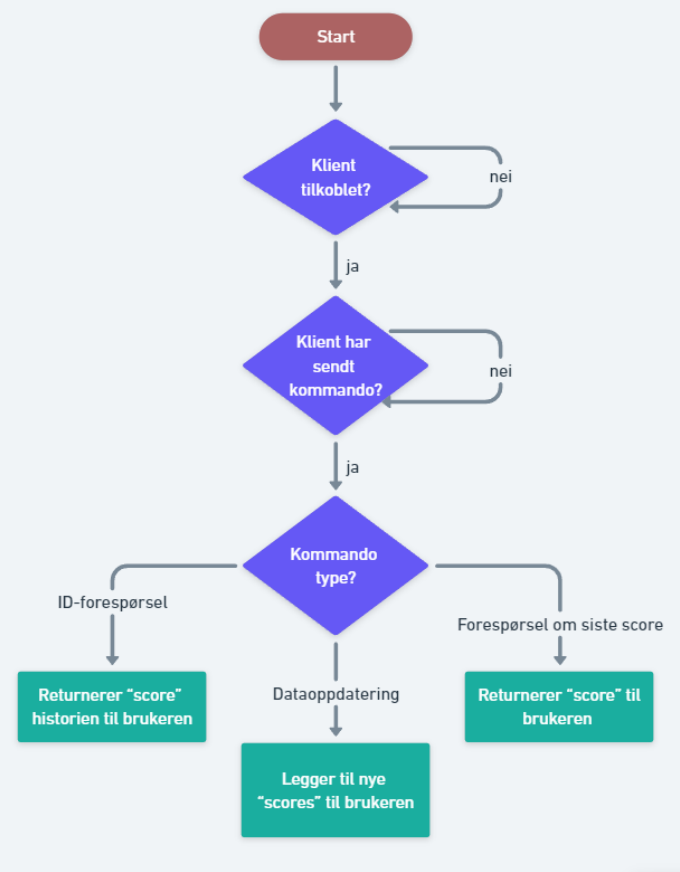
6.2.4 Utvikling av server

For å fikse kommunikasjon og overføring av data mellom de forskjellige modulene ble det utviklet en server. Denne serveren skal hovedsaklig brukes til å lagre data fra kjøremønster modulen og sende denne til ladesystem modulen når det trengs.

For å håndtere kommunikasjon mellom de ulike modulene og serveren, er det etablert et sett med forespørselstyper som en klient (en enhet koblet til serveren) kan sende. Disse forespørselstypene er lagd for å dekke behovene til smart city prosjektet, og inkluderer:

- ID-forespørsler: En klient (modul) sender en ID til serveren. Serveren bruker denne ID-en til å hente og sende tilbake alle relevante data knyttet til klienten
- Dataoppdateringer: En klient sender ny kjøremønsterdata sammen med en ID. Serveren mottar disse dataene og oppdaterer den sentrale databasen slik at informasjonen er tilgjengelig for andre moduler eller for senere analyse.
- Forespørsler om siste score: Ladesystemet kan for eksempel be om kjøremønsterdata for å avgjøre pris. Klienten sender en forespørsel om sin siste score sammen med en ID, og serveren henter og returnerer den siste scoren fra databasen. Dette gjør det mulig for ulike moduler å dele informasjon og samarbeide effektivt for å oppnå målene til smart city prosjektet.

Funksjonalitetene til serveren kan bli oppsummert med flytskjemaet i figur 25



Figur 25: Flytskjema server

Her er det meningen at kjøremønster-modulen bruker Dataoppdateringforespørselen til å sende ny oppdatert data til serveren mens lade-modulen bruker Forespørsel om siste scoreforespørselen for å få tak i scoren til brukeren. "ID-forespørselforespørselen er hovedsaklig for vedlikehold av serveren og kommer ikke til å bli tatt i bruk under normal operasjon av smart city systemet.

I tillegg til disse forespørslene er serveren designet for å være lett å utvide med flere funksjonaliteter hvis det trengs i fremtiden.

Når kjøremønster og ladesystem modulene blir implementert i den virkelige verden vil serveren trenge å prosessere og lagre store mengder data. Å arbeide med så mye data er en krevende oppgave, derfor har vi gjort flere tiltak for å optimalisere både sending av data fra kjøremønster modulen til serveren og lagring av den dataen i databasen vår.

For å optimalisere lagringen og sendingen av data på serveren vår komprimerer vi all dataen. Ved å komprimere dataen blir datamengden redusert en stor

mengde, noe som er veldig viktig for effektiv håndtering av store mengder data. Når datamengden blir redusert betyr også at det blir lettere å data fra kjøremønstermodulene til serveren som er viktig fordi vi ikke vil at brukerne skal måtte bli belastet med høyt dataforbruk på grunn av modulen.

Når systemet blir implementert, vil det være flere tusen biler som tar i bruk kjøremønstermodulen og lademodulen samtidig og sender data til serveren. For å håndtere denne store mengden samtidige forespørsler, har vi integrert multi prosessering i serveren. Dette tillater serveren å håndtere flere klienter samtidig ved å opprette en ny tråd for hver klient som kobler seg til. Hver tråd kan da uavhengig fra de andre behandle forespørslene fra sin tilhørende klient, uten å blokkere eller forsinke behandlingen av andre klienter. Multi prosessering hjelper mye når flere skal bruke systemet samtidig og det vil gjøre det enkelt for oss å skalere fra prototype som bare har et par klienter om gangen til utgivelse som kan behandle opp til flere tusen klienter om ganger.

6.3 Evaluering og test

6.3.1 Evaluering av kjøremønstermodulen

For evalueringen av kjøremønstermodulen så testet vi systemet med ekte data og passet på det oppførte seg slik som vi ønsket. Det vil si at kjørte koden og sjekket om det ble opprettet en score slik som vi hadde planlagt. Vi undersøkte også om scoren hadde oppførselen vi ønsket. Det betyr at vi sjekket om de ulike kjøremønstrene resulterte i en økning/minking av scoret slik det var ønsket. Det krevdes også en undersøkelse av kommunikasjon med de andre modulene. Scoren vi lagde for hver sjåfør skulle bli sendt med en uuid til servermodulen. Dermed var det essensielt å forsikre oss om at koden vår lyktes med å gjennomføre denne oppgaven.

6.3.2 Evaluering av lademodulen

For evaluering av lademodulen brukte teamet en kombinasjon av kvalitative og kvantitative testmetoder for å sikre en omfattende vurdering av ytelsen og påliteligheten til prototypen. Lademodulen evalueres etter en disse testpunktene:

- **Ytelsestester**
  Disse ble utført for å måle responstiden og resursforbruket til systemet. Hensikten var å vurdere hvorvidt programmet kjører i varierende nettverksmiljø.

- **Stresstester**
  For å simulere høyt trafikkvolum og belastning, ble lademodulen utsatt for kontinuerlige dataforespørsler, der hensikten var å undersøke mulige flaskehalser ved API- og serverkommunikasjon under belastning.

- **Funksjonstester**
  Disse testene fokuserte på å validere funksjonene i lademodul-programmet. Hovedfokuset var teste hvor fleksibelt programmet er for å anvende alternative API-er for strømdata, siden dette ble et problem med enkelte tidligere iterasjoner av lademodulen.

- **Brukertester**
  En kvalitativ brukertest ble utført av teammedlemmene for å simulere faktiske brukere. Hvert teammedlem testet ulike profiler med forskjellige ID-er og kjørescore for å evaluere brukeropplevelsen. Det ble undersøkt hvorvidt prisreguleringen for ulike kjøremønstre er rimelig og realistisk.

Evaluering av resultat

Ytelsestestene viser at systemet hadde akseptabel responstid på omtrent et halvt sekund for en alminnelig datamaskin og den enklere ettkortsdatamaskinen Raspberry Pi 3B. Programmet har lavt resursforbruk da det foretas kun enkle beregninger i selve programmet. Stresstest ved serverkommunikasjon avslører et problem med prototypen til serveren, da det er mulig å sende store mengder data om gangen, som ikke tas hensyn til i serverkoden. Funksjonstesten bekrefter at funksjonene i programmet opererer som forventet, og kan lett anvendes på en annen strømdata-API skulle nåværende API gå ut av drift. I siste iterasjon av lademodulen gjør programmet et enkelt API-kall ved en URL-adresse, som ikke krever et dedikert bibliotek som ble brukt i de første iterasjonene da strømprisen ble hentet via Nordpool[5]. Intern brukertest er ikke immun mot skjevheter og bias, og uten tilgang til en større brukerbase blir brukertesten mer teknisk rettet. Tilbakemelding fra teammedlemmer ansees å være objektivt nok til at det gangner utviklingen av prototypen. Problemet og forslag ble adressert i videre iterasjoner av prototypen.

Den første prototypen hadde store problemer med å hente riktig strømtabell, da den brukte et eget bibliotek for å håndtere data fra Nordpool. Biblioteket kunne ikke hente ut pristabeller for Norge, og denne løsningen ble forkastet. De ble forsøkt å hente data direkte fra andre nettsider med tilgang til strømdata, men til slutt ble Strømpris API-et[6] fra Beneficial Apps brukt i stedet. Dette forenklet utviklingen av programmet betraktelig, da det ble mulig å sende en enkel forespørsel om

[5]https://www.nordpoolgroup.com/en/trading/api/
[6]https://www.hvakosterstrommen.no/strompris-api

strømtabellen for gjeldende strømregion til API-et istedenfor å skrape data direkte fra nettsiden. Dette fungerer ypperlig sammen med `geocoder` og API-et fra Kartverket[7] for å kunne levere riktig informasjon til brukeren, i tillegg til mulighet for å bruke alternative API-er i fremtiden. Evalueringen av lademodulen konkluderes med at den siste utgaven av prototypen opererer pålitelighet og effektivt og leverer korrekt strømpris etter tidspunkt og strømregion.

6.3.3 Evaluering og test av server

Evaluering og testing av serveren ble gjennomført med flere forskjellige tester for å sikre funksjonen til serveren i en rekke scenarioer. Disse testene inkluderte:

- **Stresstest**
  For å simulere høy trafikk til serveren ble det simulert flere lade- og kjøremønstermoduler som koblet seg til og sendte kommandoer til serveren samtidig.
- **Feilhåndteringstester**
  For å sikre at serveren kunne håndtere feil og uventede hendelser testet vi hvordan serveren ville håndtere ugyldige forespørsler og koblet klienter uventet fra serveren.

Resultatene av evalueringen og testingen av serveren var positive. Stresstestene viste at serveren kunne håndtere et stort antall samtidige forespørsler uten noen negative bivirkninger. Feilhåndteringstestene viste at serveren håndterer uventede hendelser, som ugyldige forespørsler eller at klienter uventet kobler fra, uten problem.

Det er viktig å merke at under disse testene ble serveren kjørt på en bærbar datamaskin, ikke den endelige serverløsningen som vil være en fysisk eller skybasert server. Dette betyr at vi ikke har fått testet hvordan serveren vil reagere på et scenario med antall samtidige klienter som den vil ha når den blir lansert da den den bærbare datamaskinen var for svak til å teste det. Imidlertidig gjør resultatene fra testene vi har fått gjort oss sikre på at serveren er godt designet, og at den vil fungere i et slikt scenario.

6.4 Iterasjoner og forbedringer

6.4.1 Forbedring av lademodulen

[7]https://www.kartverket.no/api-og-data/grensedata

I utgangspunktet ble lademodulen utviklet separat fra kjøremodulen. Den første prototypen av lademodulen hadde begrenset funksjonalitet og fleksibilitet; posisjon og strømregion måtte hardkodes inn og kommunikasjon med server var enda ikke implementert. Hensikten med denne første prototypen var bestemme hvilken API-leverandør teamet skulle bruke for å hente strømpriser. Teamet prøvde ut API-er for de ulike markedene organisert av Nord Pool; Elspot[8] og Elbas[9] for å få tilgang til strømdata. Dette virket som en god løsning i starten, til det ble oppdaget en feil med API-en som gjorde at programmet mottok feil strømtabell. Programmet var satt opp til å hente strømtabellen fra Trøndelagsregionen (`NO3`), men det var ikke mulig å hente ut noe annet enn strømtabellen for Helsinki, Finland. Dette ble løst i neste iterasjon av lademodulen, da teamet gikk over til å bruke en API fra *Beneficial Apps*. API-en er tilgjengelig på hvakosterstrommen.no. Denne API-en tilatter input for strømregionen og støtter alle strømregionene i Norge, dermed kunne teamet også implementere en annen funksjon som var foreslått under designprosessen, mulighet til å finne posisjonen og da strømregionen til brukeren automatisk. Denne funksjonen ble realisert i den siste prototypen, i tillegg til serverkommunikasjon og regulering av strømpris basert på kjøremønsteret.

6.4.2 Kjøremønster

Når vi begynte på iterasjonene begynte vi først med en protoype som var en raspberry-pi som var teipet på en 3D-printet bil. Dette gjorde vi for å teste ut vår første algoritme, hvor vi testet hvordan systemet skulle funke. Vi brukte da den 3D-printede bilen til å bevege den i et mønster som en realistisk bil hadde gjort. Deretter testet vi forskjellige typer algoritmer som skulle gi en høyere score hvis det skulle blitt detektert en lav akselerasjon og lavere score hvis det skulle blitt detektert en høy akselerasjon. Noen problemer vi fant veldig tidlig som følge av iterasjonene våres var at vi trengte et ulineært system. Dette vil si at vi ville ha et system som skulle fungere slik at hvis man skulle hatt en ekstremt dårlig score, så skal det være mye vanskeligere å få en dårligere score, enn å få bedre score. Eksempelvis hvis man hadde vært en sjåfør med 70 score, og bråbremset hadde man for eksempel gått ned til 69 poeng. Mens hvis man hadde vært en sjåfør med 20 i score, hadde man gått ned til 19,8. Dette er fordi det er mer forventet av en sjåfør med dårlig score til å gjøre slike grove feil. Når vi hadde forestilt oss hvordan algoritmen vår skulle være, gjenstod det å teste dette på ekte data for å finne ut hvordan algoritmen skulle se ut i det ferdig-produktet vårt. Dette gjorde vi ved å bruke raspberri-pi oppsettet vårt

[8]Nord Pool Elspot
[9]Nord Pool Elbas

og kjørte rundt mens vi samlet ekte akselerasjonsdata med en ekte bil. Vi kjørte rundt i omtrent 3 timer og bruke dataen vi fikk fra denne kjøreturen til å lage den endelige algoritmen i systemet vårt. Vi bestemte at en gjennomsnittlig score for alle sjåfører skulle være 70 i score. Det vil si at hvis du har under 70 i score, som kjører som en gjennomsnittlig sjåfør, vil du etter hvert ende opp med 70 i score, og på samme måte hvis du er en sjåfør med 100 i score, og kjører som en gjennomsnittlig sjåfør skulle du også ende opp med 70 i score. Vi valgte også at 70 skulle være start-scoren, altså scoren alle sjåfører i ytreringvei skulle begynne med.

6.4.3 Bilskilt avlesning

Den første interasjonen av bilskilt avlesning systemet av er demofaset for å teste at systemet kunne funke slik vi hadde tenkt det. Denne demo fasen gikk ut på teste at de nurale nettverkene fungerete slik de skulle samt at å sette opp henting og sortering av data fra statens vegvesen. Når alle delene fungerte induviduelt lagde vi den første iterasjonen av det ferdige systemet som kjørte lokalt på en laptop. I denne versjonen slapp vi steg som å sende bilder over nettet, og at scriptene kjørte lokalt gjorde det mye lettere å debugge.

Når den første iterasjonen fungerte som tenkt begynte vi å implementere en server og en klientside lignede det ferdige produktet. Dette gikk ut på å sette opp kommunikasjonen mellom server siden og klient siden, samt å få scriptene til å kjøre og samarbeide på begge sidene. Dette brakte også nye problemer i former av anderledes kamera, og uforsette forsinkelser ved WiFi kommunikasjon som hadde vidre effekter på resten av systemet. Disse problemene ble løst og vi kunne begynne på siste iterasjon.

Siste iterasjon av bilskilt avlesnings systemet hadde sterkt fokus på virkelige bruksområdet. Her implementerte vi mer mobile kamera moduler å begynte å teste systemet langs veien og på parkerte biler. Disse mobile kamera modulene var enten batteri drevet eller koblet til en annen mobil strømforsyningskilde som usb utgang i en bil. Denne iterasjonen hadde et ekstremt fokus på testing for å forsikre oss om at alle feil ble funnet og rettet opp i det ferdige systemet.

6.4.4 Server

Den første iterasjonen av serveren var en enkel Python-kode som, når den ble kjørt, tillot klienter å koble seg til. Klientene kunne enten sende inn data som ble lagret i en tekstfil eller be om data fra den samme tekstfilen. Selv om den endelige versjonen

av serveren i hovedsak utfører de samme oppgavene, har den blitt forbedret og optimalisert betydelig sammenlignet med den første iterasjonen.

En av måtene den endelige versjonen av serveren er bedre en den første versjonen er hvordan vi lagerer data på serveren. I den første iterasjonen lagrer vi alt i en enkel tekstfil, mens i den fullførte serveren bruker vi en SQLite database. Dette gjør det mye enklere å strukturere å søke i dataene. Vi lagrer også dataen mer effektivt da den endelige versjonen av serveren bruker Python sitt pickle bibliotek for å komprimere den lagrete dataen. Ved å komprimere dataen vi lagrer bruker vi mye mindre plass på å lagre data som gjør at vi kan ha mer data på like mye lagringsplass.

En annen måte den fullførte versjonen av serveren er bedre er at den har implementert multiprosessering. Multiprosessering gjør at serveren kan håndtere flere klienter samtidig, ikke bare en og en. Den gjør dette ved å opprette en trådfor hver klient som kobler seg til. Hver av de trådene kan da uavhengig fra de andre behandle vær sin klient og fullføre forespørslene deres.

Den endelige versjonen av serveren har også implementert robust feilhåndtering. Dette gjør at serveren kan fortsette å fungere selv om det oppstår feil, som for eksempel ugyldige forespørsler fra klienter eller uventede frakoblinger. Der den første versjonen ville ha stoppet å fungere i slike situasjoner, håndterer den ferdige serveren disse feilene og fortsetter å fungere uten å bli påvirket.

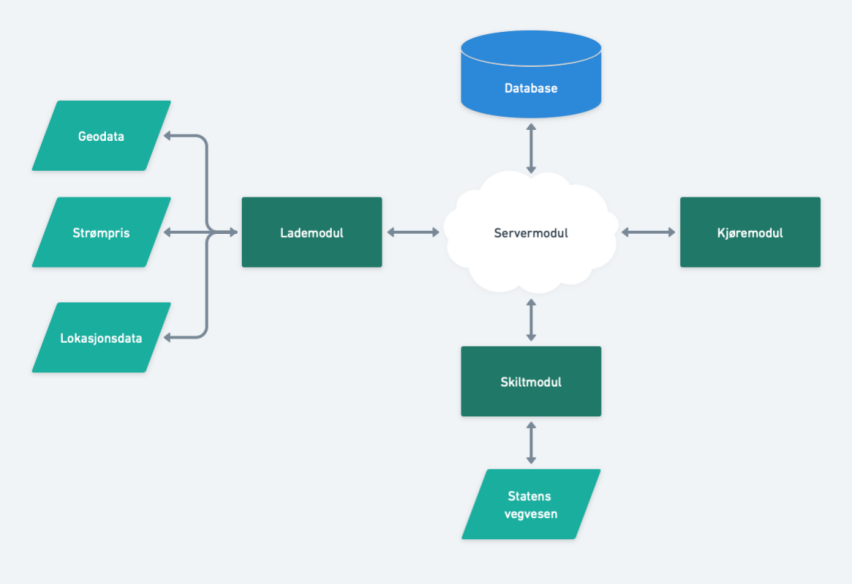
6.5 Ferdigstilling av prototypen

Under ferdigstillingen av prosjektet ble de forskjellige modulene, som hadde blitt utviklet hver for seg, integrert i et sammenhengende system. Å integrere lade-modulen, kjøremønster-modulen og serveren deres til et sammenhengende system gikk ganske problemfritt, da samhandling mellom modulene var en sentral del av systemets funksjonalitet og derfor ble prioritert under utviklingen. Dette gjorde at modulene enkelt kunne kommunisere med hverandre og utveksle data som nødvendig. Integrering av skiltgjenkjenning i resten av systemet var litt mer utfordrende, da samarbeid mellom moduler ikke er like sentralt for denne modulens funksjonalitet. Til tross for dette klarte gruppen å løse problemet gjennom godt samarbeid og kommunikasjon. Resultatet av dette samarbeidet er en sammenhengende, fungerende prototype som effektivt løser smartby-utfordringen i Ytre Ringvei.

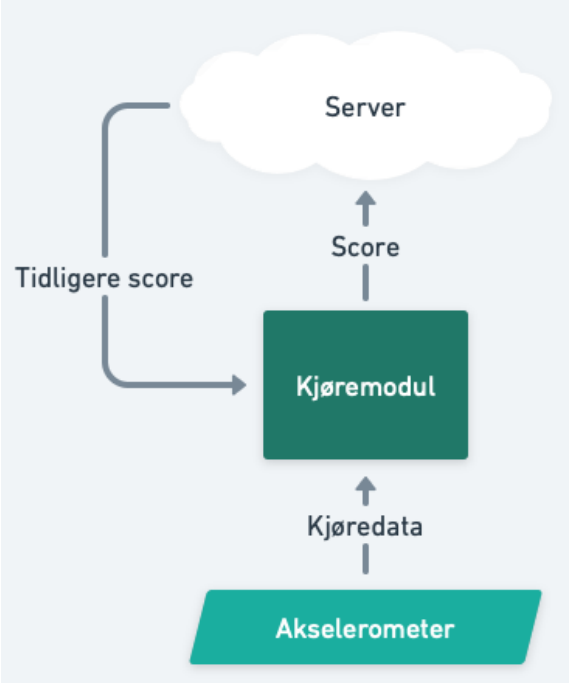
7 | Teknisk dokumentasjon

7.1 Systemarkitektur

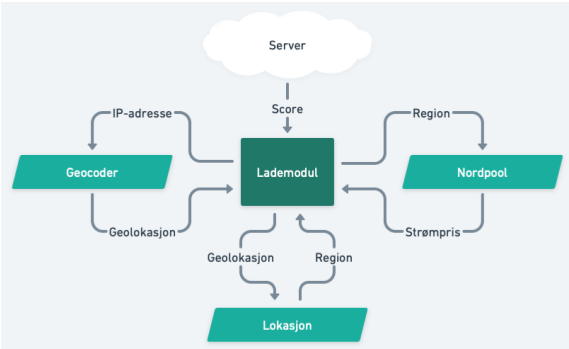
Nedenfor vises flere flytskjemaer som dokumenterer hvordan løsningene funker. Dette inkluderer et flytskjema for hvordan hele systemet funker, figur 26, i tillegg til flytskjemaer for hvert delsystem, figur 27-30.



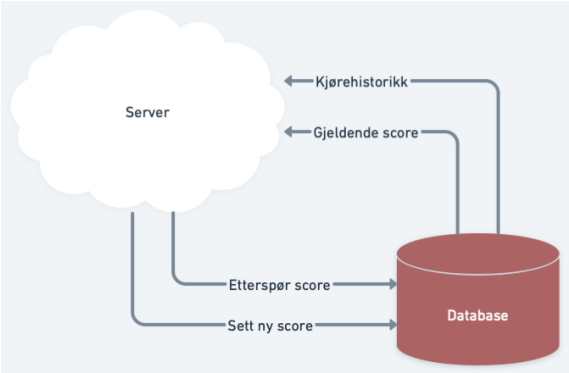
Figur 26: Bilde som viser programflyt på et modulnivå



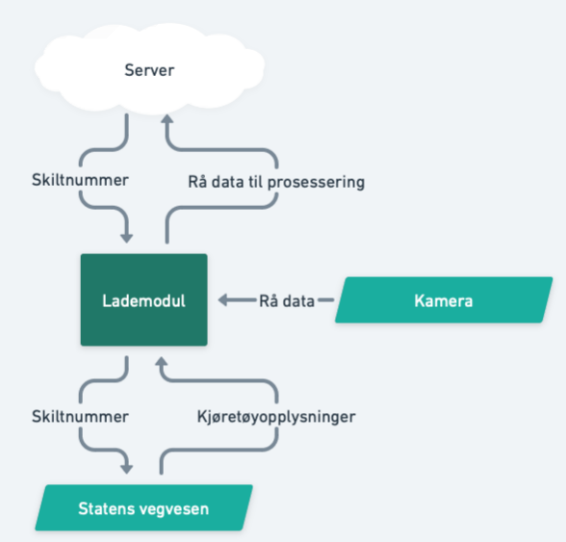
Figur 27: Flytdiagram for arkitektur og kommunikasjon mellom server og kjøremodul.



Figur 28: Flytdiagram for ladesystemets arkitektur og kommunikasjon med server.



Figur 29: Flytdiagram for kommunikasjon mellom server og database.



Figur 30: Flytdiagram for kommunikasjon mellom server og skiltgjenkjenningsmodul.

7.2 Implementering av maskinvare

For prototypen ble det valgt at det bare skulle brukes Raspberry Pi-er for databehandling nær brukeren. Dette ble bestemt på grunn av at dette ga tilgang til å bruke programmeringsspråket Python, som ble ansett som betydelig lettere å prototype i enn de andre tilgjengelige alternativene. Derimot fører dette til større kostnader enn nødvendig. Derfor planlegges det at det nær det fullførte produktet blir gjort en vurdering av hvilken mikrokontroller som er nødvendig for de ulike løsningene, og at de ulike løsningene implementeres med hensyn til disse mikrokontrollerne. Det antas at det dermed bare burde være nødvendig med en kraftigere datamaskin som en Raspberry Pi for datasyns-modulen, mens de andre modulene potensilt bare trenger en svakere datamaskin som en enkel mikrokontroller.

For å kunne realisere de ulike modulene er det nødvendig med en rekke ulike komponenter. For Kjøremønster-modulen var det nødvendig å ta benytte seg av en metode for å samle inn data for akselerasjon. Dette kravet blir oppfylt av en Sense-Hat-komponent som enkelt kan legges til på Raspberry Pi-en som blir brukt. Denne komponenten inkluderer en IMU[10] som har sensorer som blant annet måler akselerasjon. Videre kan denne sensoren enkelt aksesseres ved hjelp av innebygde funksjoner som gjør det trivielt å arbeide med sensoren. For det fullførte produktet vil derimot en mer dedikert sensor være billigere og mer ønskelig, i tillegg til at Sense-Hat-en potensielt sett kan være vanskelig å integrere med noe annet enn en Raspberry Pi.

[10]Inertial Measurement Unit

For modulen for ladesystemet blir det i prototypen brukt en enkel relé for å kunne bestemme hvorvidt en tenkt bil skal lade eller ikke. Dette gjør det enkelt og billig å iterere over ulike design for ladesystemet, men er ikke sofistikert nok til å kunne brukes til det fullførte produktet. Der vil det være nødvendig å ta i bruk komponenter som både kan regulere strømflyten, i tillegg til å kunne kommunisere med elbilen for å kunne samhandle om mengden strøm som må å tilføres.

Modulen som innebærer datasyn krever tilgang til en sensor som har mulighet til å hente inn bildedata. Siden det bare er krav om å kunne identifisere og lese av et bilskilt på relativt kort distanse er det antatt at de aller fleste kameramodulene burde være tilstrekkelige. For prototypen har det dermed blitt brukt en enkel kameramodul som var lett tilgjengelig. På grunn av at det ferdige produktet vil være utsatt for vær og vind vil det derimot være et større krav om en kameramodul som er robust, noe som vil føre til økte kostnader for denne løsningen.

For prototypen av servermodulen ble det bestemt å dele opp serveroppgavene i to. Den første delen tar for seg kommunikasjon mellom modulene, mens den andre tar for seg databehandlingen som trengs for å kunne gjennomføre skiltgjenkjenning. Dette ble gjort for å lettere kunne iterere og finne ut av hvilke løsninger som funket best. Dette gjorde også det tekniske kravet for serveren lavere, siden den dataintensive skiltgjenkjenningen kunne bli separert fra resten. Dette førte til at det under prototypingen kunne bli brukt mindre kraftige bærbare datamaskiner, som de prosjektet allerede ble utviklet med. Denne løsningen vil dermed ikke funke for det fullførte prosjektet. Da vil det bli nødvendig med en skalerbar server som er kraftig nok til å kunne behandle de mange oppdateringene av kjøremønster, spørringer av nevnt kjøremønster og mye dataintensiv prosessering av skiltgjenkjenning. Dette vil, hvis det blir bestemt å ta i bruk en ny og dedikert serverløsning, ha store oppstartskostnader. Det foreslås dermed å bruke en allerede eksisterende skytjeneste for å kunne redusere oppstartskostnadene mot en økt kostnad for å kunne operere servermodulen. Dette vil også bedre oppfylle ønsket om en skalerbar løsning, siden en bare trenger å spørre om flere ressurser fra den eksisterende skytjenesten når dette trengs.

7.3 Programvareutvikling

For å skrive programvaren som ble brukt til de ulike modulene ble det brukt utviklingsverktøyene Visual Studio Code og PyCharm. Dette er fordi disse utviklingsverktøyene tilbyr kraftige verktøy som gjør det lettere å utvikle programvaren. Blant annet gir utviklingsverktøyene raskt relevant dokumentasjon om biblioteker relevant

til programvaren samtidig som disse bibliotekene blir brukt. Dette fører dermed til økt effektivitet når programvaren utvikles sånn at programvaren kan bli ferdig raskere og bidrar til at det gjøres færre feil under utviklingen. I tillegg til dette tillater utviklingsverktøyene noe interaksjon mellom seg selv, noe som åpner opp muligheten for at utviklingsverktøyene kan velges mellom etter eget ønske og tidligere bruk av utviklingsverktøy.

7.3.1 Utvikling og dokumentasjon til lademodulen

Programvareutvikling for lademodul beskrevet i seksjon 6.2.2. Dokumentasjonen til lademodulen er vist i tabell 1 under.

Tabell 1: Dokumentasjon til lademodulen.

API	
Kommuneinfo	API fra Kartverket for å hente gjeldende fylke for gitte koordinater.
Strømpris API	API fra Beneficial Apps for henting av strømdata. Data hentes fra ENTSO-E Transparency Platform.
Biblioteker	
`request`	For behandling av HTTP-forespørsler til API.
`socket`	For nettverkstilkobling og serverkommunikasjon.
`pickle`	Til serialisering av data som sendes til server.
`datetime`	For behandling av tid og dato.
`geocoder`	Til å finne brukerens geolokasjon basert på IP-adresse.
`tqdm`	For visning av fremdriftslinjer for simulert ladesituasjon.
Funksjoner	
`get_driver_score`	Returnerer kjørescore fra server tilhørende brukerens ID.
`calculate_price_adjustment`	Beregner prisjusteringen basert på den aktuell kjørescore og ideell kjørescore.
`get_county`	Denne funksjonen bruker IP-adresse til å finne brukerens geografiske lokasjon og returnerer fylkesnavnet.
`get_region_code`	Denne funksjonen returnerer regionkoden basert på fylkesnavnet.
`get_price`	Denne funksjonen henter strømprisen for en gitt region og justerer den basert på førerscore.
`simulate_charging`	Denne funksjonen simulerer lading av et kjøretøy basert på gjeldende strømpris og batterinivå.
`print_adjusted_prices`	Denne funksjonen skriver ut de justerte prisene basert på et gitt score-intervall.

7.3.2 Server

For å utvikle serveren ble det tatt i bruk programmeringsspråket Python. Python ble valgt på grunn av at det er enkelt å forstå språk sammen med at det har et stort utvalg av biblioteker som gjør det enkelt å programmere i.

For å komprimere dataen som blir sendt fra kjøremønster modulen gjør vi det om fra en score fra 1 til 100 til en score fra 0 til 1 ved å dele på 100. Vi gjør så om datatypen scoren er lagret som fra float 32 til float 16. Denne prosessen gjør at dataen tar opp mye mindre plass og gjør at brukerne av kjøremodulen må bruke mindre data på å sende alt til serveren. Ulempen ved å gå igjennom denne prosessen er at scoren"vil bli endret litt fordi float 16 er mindre nøyaktig enn float 32, men denne endringen er så liten at den ikke vil ha noe merkbar påvirkning av det ferdige produktet.

Når dataen har nådd fram til serveren bruker vi Python-biblioteket pickle til å komprimere dataen. pickle komprimerer dataen ved å serialisere den, som vil si at den konverterer Python-objekter til en strøm av bytes. Denne strømmen av bytes tar mindre plass enn det originale Python-objektet. Den komprimerte dataen blir lagret i en SQLite database. SQLite er en enkel database som ikke trenger en egen serverprosess for å kjøre og lagrer all dataen i en enkel fil.

For å kunne sende data fra kjøremønster-modulen til serveren og fra serveren til lade-modulen har vi tatt i bruk Python-biblioteket socket. socket biblioteket lar deg opprette og bruke netverkssokketer, som er endepunktene for kommunikasjon over internett. Det gjør at så lenge lade-modulene eller kjøremønster-modulene har IP-adressen til serveren kan de opprette kommunikasjon og sende kommandoer til den. En fordel med denne løsningen er at vi ikke trenger å endre koden til serveren eller databasen når vi legger til nye brukere i systemet. Det eneste kravet er at serveren har tilstrekkelig prosesseringskapasitet til å håndtere den økte trafikken. Dette gjør systemet skalerbart og enkelt å vedlikeholde.

For at serveren skal klare å håndtere flere klienter om gangen har vi integrert threading i serveren. Threading lar serveren opprette en ny tråd for hver klient som kobler seg til. Hver tråd kan da uavhengig av de andre behandle forespørslene fra sin tilhørende klient, uten å blokkere eller forsinke behandlingen av andre klienter. Multiprosessering hjelper mye når flere skal bruke systemet samtidig, og det vil gjøre det enkelt å skalere fra en prototype som bare har et par klienter om gangen til en utgivelse som kan behandle opptil flere tusen klienter om gangen.

Under prototypingen ble det brukt to bærbare datamaskiner og en raspberry pi for å simulere kommunikasjon mellom kjøremønster-modul, server og lade-modul.

Der kjøremønster-modulen ble kjørt av raspberrry pi'en og server og lade-modulen ble kjørt på de bærbare datamaskinene. Dette gjorde det mulig å teste kommunikasjonen og datautvekslingen mellom du ulike modulene. Det er viktig å bemerke seg at en bærbar datamaskin ikke er kraftig nok til å håndtere den store mengden data og simultane forespørsler som den endelige serveren vil motta når prosjektet blir tatt i bruk. Derfor planlegges det å bruke en annen serverløsning da. Denne løsningen vil enten være en fysisk server eller skybasert en, men koden som brukes vil fortsatt være den samme som blir brukt i prototypen. Løsningene vi brukte for kjøremønster-modulen og lade-modulen trenger ikke å endre seg når vi skalerer opp.Dette er fordi mengden data modulene behandler ikke endrer seg med antall brukere, siden de blir kjørt hos hver enkelt bruker.

7.4 Kommunikasjon og sikkerhet

På grunn av at det ble bestemt å utvikle flere ulike løsninger som måtte kommunisere med hverandre ble det bestemt at det også måtte bli tatt i bruk en eller annen form for server for å koble alt sammen. Videre ble det bestemt at de allerede eksisterende løsningene ikke passet godt nok til de modulene vi planla. Dermed ble valget å utvikle en egenlagd server hvor kommunikasjonen mellom modulene og serveren også kunne bli optimalisert for det bruksscenarioet vi trengte serveren til.

For å sikre sikkerheten til systemet ble det til prototypen bestemt å generere en UUID[11] for hver bil, sånn at informasjonen som genereres blir anonym. Dette passer på at selv om informasjonen blir lekket kan informasjonen fortsatt ikke brukes til noe nyttig av aktører med dårlige hensikter. På grunn av at det er tiltenkt at prototypen ikke skal bli brukt til sluttbrukere er det dermed tenkt at denne sikkerheten er godt nok for prototypen. Dette vil derimot antakeligvis ikke være godt nok til et ferdig prosjekt. Her vil det nok være nødvendig med validering av forespørsler eller liknende for å forhindre at informasjon kommer på avveie. Derimot er det viktig å passe på at de sikkerhetstiltakene som blir valgt ikke gjør det alt for vanskelig for nye prosjekter å integrere seg med de eksisterende løsningene.

8 | Resultat og evaluering

Etter ferdigstilling av prototypene har teamet testet og prøvekjørt løsningen. Den endelige prototypen tilbyr effektive tilnærminger til å realisere kommunens nye vedtak,

[11] Universally unique identifier

i tråd med etablerte prinsipper innen designtenking, etikk og bærekraft. Kvantiseringen av kjørevaner via kjøremønstergjenkjenning viser seg å være en god tilnærming for å gjenkjenne godt og dårlig kjøremønster. Programmet for kjøremønstergjenkjenning er velfungerende, men generelt. Det erkjennes at det er mulig å designe et mer komplekst system som tar hensyn til andre aspekter ved bykjøring; start/stopp-overvåkning, unødvendig tomgang, evne til å gjenkjenne forbikjøringer, etc. Det er mulighet for utvidelse til dette i programmet. Programmet er testet i realistiske situasjoner. Dette fører til at resultatene som er produsert under prototypingen kan anses som pålitelige med det som kan forventes av programmet under produksjon. Hvis det blir bestemt å implementere et mer komplekst system kan derimot denne innsamlede kjøreinformasjonen være manglende for tilleggsfunksjonene. Det vil dermed kreves en ny runde med innsamling av data for å vite hvor effektivt systemet er.

For skiltgjenkjenningsmodulen tyder de innhentede testresultatene at modulen godt møter vedtaket satt av kommunen. Dette er fordi de to ulike løsningene i modulen, stasjonære og mobile skiltgjenkjennings-stasjoner, gir et bredt grunnlag til å hindre at ikke-elektriske biler kjører inn mot ytre ringvei i tillegg til å fange ikke-elektriske biler som allerede er innenfor ytre ringvei. Her gir hver stasjon god og pålitelig data på hver bil som blir kontrollert, samtidig som de to ulike løsningene gjør systemet robust nok til å kunne takle situasjoner hvor deler av systemet midlertidig slutter å funke. I eventuelle scenarioer hvor kommunen kan ønske å gjøre tillegg til modulen, for eksempel ved å legge til unntak til vedtaket, regnes det med at modulen lett kan modifiseres til å også gjennomføre dette via endringer i programvare. Modulen vil dermed godt kunne dekke eventuelle mangler som skulle komme til lys, i tillegg til at de eventuelle forandringene med stor sannsynlighet vil kunne valideres med informasjonen som allerede er innsamlet.

Med tanke på lademodulen har det blitt utviklet en god helhetlig tilnærming til å løse vedtaket som er gitt av kommunen. Her har reell posisjonsdata blitt brukt til å kunne teste hvorvidt det er mulig å kunne regulere strøm, noe som har blitt vist at funker på en god og pålitelig måte. Her vil det også være mulig å enkelt legge til ekstra funksjoner basert på hva som ses som nødvendig med minimalt arbeid i programvare. Det samme gjelder også for servermodulen som er produsert. Denne dekker nødvendigvis ikke noen krav fra kommunen, men oppfyller prosjektets mål godt ved å tilby en god og pålitelig måte å kommunisere på.

9 | Bærekraft og samfunnsmessig påvirkning

Gjennom prosjektet ble det bemerket flere punkter som påvirker de bærekraftige påvirkningene prosjektet har. Først og fremst er det at byttet fra bensinbiler til elbiler kommer til å føre til en betydelig reduksjon i årlig karbonutslipp, på grunn av at framdriftskraften til bilene kan produseres med fornybar energi og ikke med fossilt brennstoff. Dette valget vil også, på grunn av at elektrifisering av biler krever at helt nye biler produseres, føre til at utslippene i et kort tidsperspektiv vil øke kraftig siden produksjon av spesielt elbiler er veldig kostbart (Miljødirektoratet 2022). I tillegg vil byttet fra bensinbiler til elbiler føre til et problem hvor fortsatt fungerende bensinbiler må vrakes, som dermed fører til at unødige materielle kostnader i tillegg til de faste produksjonskostnadene for bensinbilene ikke kan bli fordelt utover like mange kilometer kjørt som de ellers kunne gjort hvis den kjørte så lenge som mulig. Disse fordelene og ulempene kan derimot bli motvirket noe hvis det blir bestemt å tillate noen ikke-elektriske biler til å kjøre innenfor ytre ringvei i en forlenget periode.

Det er også kommet fram at prosjektet kommer til å ende opp med å ha en tydelig påvirkning for samfunnet. For enkeltpersoner fører blant annet kjøremønstermodulen til at det lettere er mulig å få informasjon som hjelper dem til å ta mer bærekraftige valg i tillegg til at dette også kan brukes for å få besparelser på kostnader som strømbruk. Derimot vil det at løsningen som er foreslått også inkluderer straffer for førere som ikke er bærekraftige føre til at mange personer kan få et veldig negativt bilde av løsningen, noe som kan føre til vanskeligheter med å innføre vedtakene. I tillegg vil modulene antakeligvis kreve flere ekstra komponenter for å kunne fungere, noe som kan føre til vanskeligheter når noen av disse stopper å funke. Videre kan det også oppstå problemer hvis enkeltpersoner prøver å finne smutthull for å kunne oppnå insentiver eller unngå straffer som de egentlig skulle mottatt. Dette vil dermed føre til økte unødvendige kostnader, samtidig som at andre brukere kan miste tilliten til systemet når noen oppnår noe de ikke har fortjent.

Et annet viktig aspekt med prosjektet er hvordan det bidrar til en potensielt stor økning i overvåkning av befolkningen. Dette inkluderer både skiltgjenkjenningen, hvor blant annet informasjon om hvilke regioner enkeltpersoner kjører i kan samles, men også kjøremønstermodulen, hvor informasjon om hvordan og kanskje mer nøyaktig hvor enkeltpersoner kjører kan bli innhentet. Med tanke på bærekraft

vil disse opplysningene kunne hjelpe kommunen til å finne ut av hvilke grupper som fører til størst strømbruk, noe som kan brukes til å bedre velge nye prosjekter for mer bærekraft. I tillegg kan kjøreinformasjon hjelpe politiet med å ta personer som driver med lovbrudd. Derimot vil den økte overvåkningen minske privatlivet til enkeltpersoner, hvor alt de gjør i bilen potensielt kan bli overvåket og kritisert. Dette vil under rammene av prosjektet føre til at enkeltpersoner som blant annet kjører av andre grunner enn essensielle nødvendigheter kan bli straffet. Videre kan senere prosjekter også definere flere måter av uønsket kjøring, noe som kan redusere bevegelsesfriheten til enkeltpersoner drastisk.

10 | Etikk og lovgivning

Bruken av KI i programvare for bildegjenkjenning, spesielt automatisk nummerskiltgjenkjenning (ALPR[12]) i et Smart City-miljø, medfører en rekke etiske implikasjoner som krever nøye vurdering; implikasjonene omfatter personvernhensyn, potensiell bias, datasikkerhet og balanse mellom offentlig sikkerhet og siviles rettigheter.

Et klassisk etisk problem ved overvåkning og bruk av KI er invasjon av privatliv. ALPR-systemer kan fange og lagre store mengder data, inkludert bevegelse og posisjon til kjøretøy og deres passasjerer. Kapasiteten til masseovervåkning kan bekymre innbyggere og trafikanter over hvorvidt personvernsrettighetene overholdes; innsamlet data kan brukes til å spore individers bevegelser over tid; et pressende sikkerhetsproblem dersom data misbrukes eller kommer på avveie. Datasikkerhet er derfor et seriøst problem i Smart City-prosjekter. Data som ikke er tilstrekkelig beskyttet er sårbart for hacking og uatorisert tilgang. Overvåkningssystemet håndterer posisjonsdata som kan utnyttes til ondsinnede formål, som forfølgelse. Det kreves robuste sikkerhetstiltak og klare retningslinjer for datalagring, -behandling og eierskap for å motvirke disse risikoene (A. Armijo 2020).

Hvordan skal balansen mellom offentlig sikkerhet og sivile rettigheter opprettholdes? ALPR-teknologi kan betydelig forbedre offentlig sikkerhet ved å hjelpe til med å pågripe kriminelle, finne savnede personer og gjenfinne stjålne kjøretøy, men utgjør også en trussel mot sivile rettigheter uten forsvarlig regulering. Utfordringen ligger i å finne en balanse mellom å utnytte teknologien for det offentlige gode og beskytte individuelle rettigheter. Overdreven avhengighet av ALPR for mindre lovbrudd eller inntektsgenerering gjennom bøter kan føre til offentlig mistillit og motstand. For å adressere disse etiske problemstillingene er det avgjørende å utvikle

[12]Engelsk: *Automatic License Plate Recognition.*

et omfattende etisk rammeverk og reguleringsretningslinjer for bruk av teknologien. Dette omfatter:

- Åpenhet og klar kommunikasjon: Informere brukere hvordan data behandles og følger lovverket for personvern og databehandling.
- Ansvarlighet: Opprette interne mekanismer for å holde personell ansvarlig ved misbruk eller sikkerhetsbrudd.
- Håndtering av skjevheter og bias: Regelmessige revisjoner og oppdateringer av KI-modell; trening og evaluering på nye datasett.
- Databeskyttelse: Sterke datasikkerhetsprotokoller for å beskytte mot uautorisert tilgang og brudd.
- Offentlig engasjement: Involvering av innbyggere i diskusjoner om utplassering og bruk av teknologien for å bygge tillit og sikre at teknologien gagner samfunnet.

Dette er eksempler på etiske implikasjoner og problemstillinger ved bruk av KI i Smart City-miljøer, disse må tas hensyn til og håndteres med omtanke. Ved å implementere robuste etiske retningslinjer og reguleringsrammeverk, er det mulig å utnytte fordelene med denne teknologien og samtidig beskytte individuelle rettigheter og opprettholde offentlig tillit.

11 | Konklusjon og anbefalinger

Gjennom prosjektarbeidet har teamet tilegnet seg nye og spennende erfaringer om hvordan det er å utvikle større, sammenhengende løsninger. Teamet har satt seg grundig inn i de mest anvendte design- og prosjektmetodikkene fra et teknologisk ståsted, og erfart at selv etter uendelige designiterasjoner er det til syvende og sist alltid noe som kan forbedres eller bygges videre på. Angående anvendelse av den ferdige løsningen erkjenner teamet at det fortsatt er en prototype og er underlagt realistiske begrensninger. Av disse er sikkerhet det mest betydelige, da ytterligere sikkerhetstiltak bør vurderes i et ferdig produkt av denne størrelsen.

Etter en analyse av den ferdige prototypen, kan det konkluderes med at prosjektet i stor grad har oppfylt de definerte behovene og ambisjonene til Ytre Ringvei; i den forstand at vi har skapt incentiver for bærekraftige endringer i byen ved bruk av modulene vi introduserer i prosjektet vårt.

Resurser og litteratur

A. Armijo, B. Korinek (2020). *Drivers Beware: The Ethics of Law Enforcement Technology*. Accessed: 2024-05-30.

docparser (2024). *Text$_C$aptcha$_b$reaker*. Accessed: 24-06-06.

Dubber, Markus D., Frank Pasquale og Sunit Das (jul. 2020). *The Oxford Handbook of Ethics of AI*. Oxford University Press. ISBN: 9780190067397. DOI: 10.1093/oxfordhb/9780190067397.001.0001. URL: https://doi.org/10.1093/oxfordhb/9780190067397.001.0001.

Midjo, Arne (2024a). *Designtenkning er også en mentalitet*. Accessed: 24-06-06.

— (2024b). *Designtenkning-cheat-sheet*. Accessed: 24-06-06.

— (2024c). *Hva er designtenking?* Accessed: 24-06-06.

— (2024d). *Komme i gang råd 1: Empatifasen i designtenkning*. Accessed: 24-06-06.

Miljødirektoratet (2022). «Klimagassutslipp fra elbiler». I: Accessed: 2024-06-5.

roboflow (2023). *License Plate Recognition Computer Vision Project*. Accessed: 24-06-06.

Schneider, J. (2017). *Understanding Design Thinking, Lean, and Agile*. O'Reilly Media, Inc. ISBN: 9781491980477.

ultralytics (2024). *v8.2.0 - YOLOv8-World and YOLOv9-C/E Models*. Accessed: 24-06-06.

Merknad: KI-verktøy er brukt til språkvask.

Vedlegg

A Endringslogg

29.mars Evaluering og valg av moduler

4. april Startet på modulene

10. mai Brukertest av modulene

24. mai Startet på prosjektrapporten

1. juni Ferdigstilling med modulene

7. juni Ferdigstilling med prosjektrapporten

B Pythonkode for lademodul

```python
import requests
import socket
import pickle
import datetime
import geocoder
from time import sleep
from tqdm import tqdm

def get_driver_score(my_id):
    s = socket.socket()
    print("Socket created")

    # Connect to the server
    s.connect(('12.52.10.50', 12345))
    print("Connected")

    data = ["ask", my_id]

    # Send id to server
    s.send(pickle.dumps(data))
    print("Data sent")
    score = s.recv(1024).decode()
    s.close()
    return int(float(score) * 100)

def calculate_price_adjustment(current_score, ideal_score):
    if current_score >= ideal_score:
        return 1.0  # no price adjustment
    elif current_score < ideal_score:
        price_adjustment = 1.0 + (ideal_score - current_score) / 100
        return price_adjustment


def get_county():
    try:
        # Get coordinates
        g = geocoder.ip('me')
        lat = g.latlng[0] # Latitude
        lon = g.latlng[1] # Longitude
```

```
40
41
42          # Use coordinates to fetch county from API
43          url = f"https://api.kartverket.no/kommuneinfo/v1/punktfiltrer=
            ↪  fylkesnavn&nord={lat}&ost={lon}&koordsys=4258"
44          response = requests.get(url)
45          response.raise_for_status()  # Raise an exception for HTTP errors
46          data = response.json()
47          if "fylkesnavn" in data:
48              county = data["fylkesnavn"]
49              return county
50          else:
51              return "County not found"
52      except (requests.RequestException, ValueError) as e:
53          print(f"Error occurred while fetching county: {e}")
54          return "County not found"
55
56
57  def get_region_code(county):
58      # Nord Pool Spot region codes
59      region_codes = {
60          "Nordland": "NO1",
61          "Troms og Finnmark": "NO1",
62          "Agder": "NO2",
63          "Telemark": "NO2",
64          "Vestfold": "NO2",
65          "Buskerud": "NO2",
66          "Trøndelag": "NO3",
67          "Møre og Romsdal": "NO3",
68          "Hordaland": "NO4",
69          "Sogn og Fjordane": "NO4",
70          "Rogaland": "NO4",
71          "Oslo": "NO5",
72          "Akershus": "NO5",
73          "Østfold": "NO5",
74          "Hedmark": "NO5",
75          "Oppland": "NO5"
76      }
77      return region_codes.get(county, "Unknown")
78
79
80  def get_price(region_code, score, ideal_score):
81      try:
82          # Get current date
83          year = datetime.datetime.now().strftime("%Y")
84          month = datetime.datetime.now().strftime("%m")
85          day = datetime.datetime.now().strftime("%d")
86
87          # Fetch spot prices from API
88          url =
            ↪  f"https://www.hvakosterstrommen.no/api/v1/prices/{year}/{month}-{day}_{region_code}.json"
89          response = requests.get(url)
90          response.raise_for_status()  # Raise an exception for HTTP errors
91          price_table = response.json()
92
93          now = datetime.datetime.now().hour
94
95          # Find current price from timetable
96          for time_period in price_table:
97              start_hour = int(time_period['time_start'].split('T')[1][:2])
98              end_hour = int(time_period['time_end'].split('T')[1][:2])
99
100             if start_hour <= now < end_hour:
101                 price = time_period['NOK_per_kWh']
102
103                 # Adjust price according to driving score
104                 adjustment = calculate_price_adjustment(score, ideal_score)
105                 return price * adjustment * 1.25  # Add mva
106
107         return None
108     except (requests.RequestException, ValueError) as e:
109         print(f"Error occurred while fetching price: {e}")
110         return None
```

```
111
112
113 def simulate_charging(region_code, score, ideal_score, max_price, battery_level,
    ↪ max_battery_level):
114     price = get_price(region_code, score, ideal_score)
115     print("-" * 70)
116     print(f"Your location is {get_county()}, region {get_region_code(get_county())}.")
117
118     if price is not None:
119         if price <= max_price:
120             print(f"Current price is {round(price, 2)} kr/kWh. Max price is set to {max_price}
                ↪ kr/kWh.")
121             print("Charging vehicle...")
122             print("-" * 70)
123
124             for level in tqdm(range(battery_level, max_battery_level)):
125                 sleep(level / max_battery_level)
126
127             print("Vehicle is fully charged!")
128         else:
129             print(f"Current price, {round(price, 2)} kr/kWh exceeds max price {round(max_price,
                ↪ 2)} kr/kWh!")
130     else:
131         print("Price is 'None'.")
132
133 def print_adjusted_prices(region, ideal_score):
134     for score in range(0, ideal_score):
135         print(f"Score: {score}\t\t\tPrice: {round(get_price(region, score, ideal_score),
            ↪ 2)}kr/kWh.")
136     print(f"Score: {ideal_score} - 100\t\tPrice: {round(get_price(region, ideal_score,
        ↪ ideal_score), 2)}kr/kWh.")
137
138 def main():
139     ideal_score = 70
140
141     my_id = 'f6188363-d31b-42e0-8ce6-b6e02e539c28'
142     region = get_region_code(get_county())
143
144     score = get_driver_score(my_id)
145
146     print(f"Score is {score}. Base price is {round(get_price(region, ideal_score, ideal_score),
        ↪ 2)},
147     you're paying {round(get_price(region, score, ideal_score), 2)} kr/kWh.")
148
149     # Compare prices for good and bad driving
150     #print_adjusted_prices(region, ideal_score)
151
152     # Simulate a charging situation
153     #simulate_charging(region, ideal_score, ideal_score, max_price=5, battery_level=0,
        ↪ max_battery_level=50)
154
155 if __name__ == "__main__":
156     main()
157
```

C Kommunikasjon mellom lademodul og server

```
1  import socket
2  import pickle
3  import time
4
5  s = socket.socket()
6  print("Socket created")
7
8  port = 12345
9  s.connect(('12.52.10.50', port))
10 print("Connected")
```

```
my_id = "2221b187-fbd0-4978-acc2-9e51fe8b0ae9"

time.sleep(0.5)

data = ["ask", my_id]
s.send(pickle.dumps(data))
print("data sent")
print (s.recv(1024).decode())

s.close()
```

D Skilt gjenkjenning klient kode

```
import cv2
from PIL import Image
from io import BytesIO
import numpy as np
import socket
import struct
import pickle

# Create a socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
#client_socket.connect(('elektratemp.ignorelist.com', 4333))
client_socket.connect(('localhost', 8485))
connection = client_socket.makefile('rwb')

cap = cv2.VideoCapture(0)

try:
    while True:
        ret, frame = cap.read()
        #print(frame.shape)
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        pil_img = Image.fromarray(frame)
        pil_img = pil_img.resize((800, 608))
        buffer = BytesIO()
        pil_img.save(buffer, "JPEG", quality=5)
        data = buffer.getvalue()

        # Write the length of the capture to the stream and flush to ensure it actually gets sent
        connection.write(struct.pack('<L', len(data)))
        connection.flush()

        # Send image frame
        connection.write(data)
        connection.flush()

        # Receive the length of the pickled list
        list_len = struct.unpack('<L', connection.read(struct.calcsize('<L')))[0]
        pickled_list = connection.read(list_len)
        my_list = pickle.loads(pickled_list)

        print(my_list)
        for item in my_list:
                buffer = BytesIO()
                pil_img = pil_img.crop((item[0], item[1], item[2], item[3]))
                pil_img.save(buffer, "JPEG", quality=95)
                data = buffer.getvalue()

                connection.write(struct.pack('<L', len(data)))
                connection.flush()

                connection.write(data)
                connection.flush()


```

```
finally:
    connection.close()
    client_socket.close()
```

E Skilt gjenkjenning server kode

```
import cv2
import numpy as np
import socket
import struct
from io import BytesIO
import pickle
from ultralytics import YOLO
from PIL import Image
from torchvision import transforms as T
from text_module import get_text
from get_data import get_sign

model = YOLO('best.pt')

# Create a socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(('localhost', 8485))
server_socket.listen(0)

# Accept a single connection
connection = server_socket.accept()[0].makefile('rwb')

def get_sqares(img):
    pil_img = Image.fromarray(img)

    transform2 = T.ToTensor()
    image_tensor = transform2(img).unsqueeze(0)
    results = model(image_tensor)

    return results[0].boxes.data.round().int().tolist()
try:
    while True:
        # Read the length of the image as a 32-bit unsigned int. If the length is zero, quit the loop
        image_len = struct.unpack('<L', connection.read(struct.calcsize('<L')))[0]
        if not image_len:
            break

        # Construct a stream to hold the image data and read the image data from the connection
        image_stream = BytesIO()
        image_stream.write(connection.read(image_len))

        # Rewind the stream, open it as an image with PIL and do some processing on it
        image_stream.seek(0)
        data = np.frombuffer(image_stream.getvalue(), dtype=np.uint8)
        img = cv2.imdecode(data, 1)

        my_list = get_sqares(img)

        # Pickle the list
        pickled_list = pickle.dumps(my_list)

        # Send the pickled list
        connection.write(struct.pack('<L', len(pickled_list)))
        connection.flush()
        connection.write(pickled_list)
        connection.flush()

        for item in my_list:
```

```
            image_len = struct.unpack('<L', connection.read(struct.calcsize('<L')))[0]
            if not image_len:
                break
            image_stream = BytesIO()
            image_stream.write(connection.read(image_len))
            image_stream.seek(0)
            data = np.frombuffer(image_stream.getvalue(), dtype=np.uint8)
            img_part = cv2.imdecode(data, 1)

            #if get_sign(get_text(Image.fromarray(img_part))):
            #img_part[:, :, 1] = 255
            #else:
            img_part[:, :, 2] = 255 #red
            print(img_part.shape)
            img[item[1]:item[1]+img_part.shape[0],item[0]:item[0]+img_part.shape[1]] = img_part

        cv2.imshow('ImageWindow', img)
        cv2.waitKey(1)


finally:
    connection.close()
    server_socket.close()

```

F Pyhton kode for henting av informasjon fra statens vegvesen (get_data.py)

```
import json
import requests
import time

headers={'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.0.0
cach = {}

def get_sign(what):
    if cach.get(what, (0,0))[0] > time.time()-60*60*24:
        print("loaded from cach;", cach[what][1])
        return cach[what][1]

    url = "https://kjoretoyoppslag.atlas.vegvesen.no/ws/no/vegvesen/kjoretoy/kjoretoyoppslag/v1/oppslag/raw/"  + what
    data = json.loads(requests.get(url.strip(), headers=headers, timeout=10).text)
    data = data['kjoretoy']['godkjenning']['tekniskGodkjenning']['tekniskeData']['motorOgDrivverk']['motor']

    elektrisk = True
    for motorer in data:
        for item in motorer['drivstoff']:
            if item['drivstoffKode']['kodeNavn'] != "Elektrisk":
                elektrisk = False

    cach[what] = (time.time(), elektrisk)

    print(elektrisk)
    return elektrisk

if __name__== "__main__":
    get_sign("EV64761")
    get_sign("EV64761")
    get_sign("KH68603")
    get_sign("EV6476i")
```

G Pyhton kode for klient kjøremønster

```
import time
from sense_hat import SenseHat
import pickle
import numpy as np
import socket


#SETUP
try:
    with open("id.config", "r") as f:
        my_id, cscore = pickle.load(f)
except:
    import uuid
    my_id = str(uuid.uuid4())
    cscore = 70
    with open("id.config", "w") as f:
        pickle.dump([my_id, c], f)
        f.write(my_id)


sense = SenseHat()
s = socket.socket()
port = 12345
s.connect(('10.22.52.50', port))



def p(*a):
    return sum(list(map(lambda x: x**2,a)))

c = []
avg = [0]*5
start = time.time()
ss = True

scores = []
while True:
    a = sense.get_accelerometer_raw()
    s = p(a['x'],a['y'],a['z'])

    c.append(s)

    if time.time() > start + 0.4:
            s = abs((sum(c)/len(c)) - 0.981**2)
            avg.pop(0)
            avg.append(s)

            if ss:
                sense.clear((255,0,0))
                if sum(avg)/len(avg) > 0.02:
                    ss = False

            else:
                sense.clear((0,255,0))
                if sum(avg)/len(avg)  < 0.015:
                    ss = True
                else:
                    foo = avg[len(avg//2)+1]
                    cscore += max(-0.03, foo**((cscore+130)/200) - 0.1)/10
                    scores.append(cscore)

                    if len(scores) == 100:
                        sense.clear((0,0,255))
                        data = ['kj', np.array(scores, dtype=np.float32),my_id] #Gjør om float1s fra float 32 eller 64 ti
                        s.send(pickle.dumps(data))
                        print("data, sendt")
                        print (s.recv(1024).decode())
```

```
68                    scores = []
```

H Brukeranalyse med personas

Figurer

Tabeller

A | Bidragserklæring

Tabell 3: Bidragserkæring over utført arbeid

Seksjon	Utført av: Ali Amlashi	Andreas Lindeman	Ove Ljosland	Emil Hallingstad	Mathias Schiøtz
Innledning				✓	
Prosjektmetodikk	✓		✓		
Designtenking og -utvikling	✓		✓		
Empati	✓		✓		
Definisjon	✓				
Idéutvikling	✓		✓		
Prototyping	✓				
Test og iterasjon	✓				
Integrering av utviklingsmetodikk	✓				
Brukerinteraksjon og -orientering	✓		✓		
Empati og brukerundersøkelser			✓		
Definisjon av brukerbehov			✓		
Idégenerering og sortering					
Prototyping og brukertesting	✓		✓		
Ferdigstilling og bruk av løsningen					✓
Prototypeutvikling	✓	✓	✓	✓	✓
Valg av prototypekonsepter	✓	✓	✓	✓	✓
Utvikling og konstruksjon av prototype	✓	✓	✓		✓
Evaluering og test	✓	✓	✓		✓
Iterasjoner og forbedringer	✓	✓	✓		✓
Ferdigstilling av prototypen		✓	✓		✓
Teknisk dokumentasjon		✓	✓	✓	✓
Systemarkitektur		✓	✓	✓	✓
Implementering av maskinvare		✓	✓	✓	✓
Programvareutvikling		✓	✓		✓
Kommunikasjon og sikkerhet				✓	
Resultat			✓	✓	
Bærekraft og samfunnsmessig påvirkning				✓	
Etikk og lovgivning			✓		
Kritisk refleksjon			✓		
Konklusjon og anbefalinger	✓		✓		
Tilhørende refleksjonsnotat			✓		
Illustrasjoner og figurer	✓	✓	✓		✓
Utvikling av prototypemodulene	✓	✓	✓		✓
Kjøremodul	✓	✓			
Kameramodul		✓			
Lademodul			✓		
Servermodul					✓