Bacheloroppgave

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

Jon Ola Landgraff, Lars Markus Lerdahl, Karsten Sedal Slagstad, Aksel Digre Søbstad

Fjernstyring av kjøretøy over 5G/4G

Bacheloroppgave i Ingeniørfag, elektro
Veileder: Cuong Phu Le
Medveileder: Stig Petersen

Mai 2021

NTNU
Kunnskap for en bedre verden

Jon Ola Landgraff, Lars Markus Lerdahl, Karsten Sedal Slagstad, Aksel Digre Søbstad

Fjernstyring av kjøretøy over 5G/4G

Bacheloroppgave i Ingeniørfag, elektro
Veileder: Cuong Phu Le
Medveileder: Stig Petersen
Mai 2021

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for elektroniske systemer

NTNU
Kunnskap for en bedre verden

NTNU

Institutt for elektroniske systemer
Institutt for elkraft
Institutt for teknisk kybernetikk

Fjernstyring av kjøretøy over 5G/4G

Oppgavens tittel: Fjernstyring av kjøretøy over 5G/4G **Project title:** Remote controlled vehicle over 5G/4G	**Gitt dato:** 15.12.20 **Innlevingsdato:** 20.05.21 **Gradering:** [x] åpent [ ] lukket [ ] åpent fra _____ **Antall sider/bilag:** 185
Gruppedeltakere: Jon Ola Langraff (JL) tlf. 94033468 email: jonoland@stud.ntnu.no Lars Markus Lerdahl (LL) tlf. 92834253 email: larsmle@stud.ntnu.no Karsten Sedal Slagstad (KS) tlf. 97472034 email: karstess@stud.ntnu.no Aksel Digre Søbstad (AS) tlf. 91621286 email: akselds@stud.ntnu.no	**Veileder internt:** Cuong Phu Le (CL) tlf. 94257411 email: cuong.le@ntnu.no **Veileder eksternt:** Stig Petersen (SP) tlf. 93003112 email: stig.petersen@sintef.no
Studieretning: Elektroingeniør, Elektronikk	**Prosjektnummer:** E2118
Oppdragsgiver: Yara International ASA	**Kontaktperson hos oppdragsgiver:** Stig Myrland (SM) tlf. 90791444 email: stig.myrland@yara.com

Sammendrag

I denne rapporten undersøkes mulighetene for fjernstyring av kjøretøy over 5G/4G, der hovedfokus er utfordringer knyttet til forsinkelse i video og styringssignal. Videre vektlegges også den subjektive brukeropplevelsen ved styring av kjøretøyet, dette for å gi en totalvurdering av systemet.

Prosjektet er gitt av Yara og er en del av EU prosjektet '5G-SOLUTIONS' delprosjekt 'UC3.5: Autonomous assets & logistics for smart port'. 5G-nettet til Telenor benyttes og prosjektet er et 'proof of concept' for fjernstyring av kjøretøy over 5G/4G.

Det er benyttet en Arduino som motorkontroller, festet på et egetdesignet kretskort med en logikk-konverteringskrets. For montering på kjøretøyet er det designet en 3D-printet brakett, med monteringsmuligheter for kretskort, Raspberry Pi, kamera og antenner.

For nettverkskommunikasjon og video-prosessering er det benyttet to Raspberry Pi, en på pilot-siden og en på kjøretøyet. Bilen styres ved hjelp av ratt og pedaler og piloten mottar direktevideo fra bilens perspektiv.

Hvordan minimere videoforsinkelsen har vært en stor utfordring i prosjektet. GStreamer har vist seg å være den beste løsningen, og benyttes derfor i alle ledd av videokommunikasjon. Nettverkskoden er utviklet i C++ og motorkontrolleren er utviklet i AVR-C.

Videre er egenutviklede tester brukt for å tallfeste video- og styringssignalenes forsinkelse. Testmetoder som er brukt er ping-test, signaltest med ekstern stoppeklokke og glass-to-glass test av video.

Resultatene fra test av forsinkelse viser at overføring via 5G gir medianverdier på cirka 123 ms for video, og cirka 35 ms for styringssignal. Overføring via 4G gir noe høyere verdier, henholdsvis cirka 133 ms og cirka 39 ms.

Subjektivt oppleves det at piloten har god kontroll over kjøretøyet. En viktig grunn til dette er valg av ratt, pedaler og racingstol for fjernstyring.

Abstract

In this report, the feasibility of controlling a vehicle remotely over 5G/4G is explored. The main focus is latency in regard to video- and control signals. The subjective experience of controlling the vehicle is also discussed, in order to give a complete assessment of the system.

The project is given by Yara, and is a part of the EU-project '5G-SOLUTIONS', subproject 'UC3.5: Autonomous assets & logistics for smart port'. Telenor's 5G-network is used, and the project is a 'proof of concept' for proving the feasibility of operating vehicles remotely over 5G/4G.

An Arduino is being used as motor control unit (MCU), mounted on a self-designed PCB. For mounting on the vehicle, a 3D-printed bracket is designed. This allows for mounting of the PCB, Raspberry Pi, camera, and antennas.

Network communication and video processing is done by two Raspberry Pis, one on the pilot side and one on the vehicle. The vehicle is controlled by steering wheel and pedals, while the operator receives live video from the perspective of the vehicle.

How to minimize video latency has proved to be a challenge. GStreamer has been found to be the best solution, and is therefore used in all aspects of video communication. The network code is written in C++ and MCU software in AVR-C.

Furthermore, self-developed tests are used to benchmark the latency of the video- and control-signal. Tests used include ping test, measurements of signal latency with the help of an external microcontroller for time-measurement, and glass-to-glass test of video.

Testing show that transmission by 5G gives median latency values of approximately 125 ms in the case of video, and approximately 35 ms in the case of control signals. Transmission by 4G results in higher values, approximately 135 ms in the case of video, and 39 ms in the case of control signals.

Subjectively, there is a feeling of being in control when operating the vehicle.

Forord

Oppgaven ble gitt av Yara i samarbeid med Telenor, SINTEF og NTNU.

Stig Myrland, Process System Manager for Yara Porsgrunn, var oppdragsgiver og vår kontaktperson i Yara. Som oppdragiver stilte Stig Myrland, på vegne av Yara, med finansiering og produktspesifikasjoner til det endelige produktet.

Cuong Phu Le, Førstelektor ved institutt for elektroniske systemer NTNU, har igjennom oppgaveprosessen stilt seg tilgjengelig for gruppa som intern rådgiver, med sin brede kompetanse innen elektronikk-fagfeltet. Videre også til stor hjelp igjennom skriveprosessen.

Stig Petersen, Seniorforsker på avdelingen 'Connectivity Technologies and Platforms' ved SINTEF, var ekstern veileder og til stor hjelp tidlig i designfasen for mulige løsninger til prosjektet.

Olai Bendik Erdal, Senior Project Manager - Business Development for Telenor, var vår kontaktperson i Telenor. Olai Bendik Erdal ordnet med SIM-kort til prosjektet og bidro med informasjon rundt Telenors 4G- og 5G-nett.

Andre takk:
Takk til MAKENTNU for hjelp med 3D-print.
Takk til Elektronikk og prototypelabben ved institutt for elektroniske systemer for produksjon av kretskort og 3D-print.
Takk til Christian Frugone, avdelingsingeniør ved institutt for konstruksjonsteknikk, NTNU. Christian hjalp til med utfordringer knyttet til 3D-design og 3D-print.
Takk til Even Johan Christiansen, avdelingsingeniør ved institutt for elektroniske systemer, NTNU. Even hjalp til med utfordringer knyttet til PCB-designverktøyet CircuitMaker.
Takk til André Midtby for hjelp med grafisk design.
Takk til andre som har bidratt.

Underskrift:
20.05.2021

[illegible] Jon Ola Lantgraff Aksel Digre Søbstad

Karsten S. Slagstad

Innhold

Figurer

Tabeller

Kodelister

Ordliste

4G Fjerde generasjons mobilnett. Også kalt LTE. 4G er etterfølgeren til de tidligere teknologiene 2G (GSM) og 3G (UMTS/WCDMA). iv, 1, 3, 10, 11, 17, 66, 81, 83, 94, 102, 105, 112, 113, 115, 120, 122

5G Femte generasjons mobilnett. iv, 1–3, 10, 11, 15–18, 25, 27, 66, 79, 81, 83, 94, 102, 103, 106, 112–115, 119–122

ADC Analog-to-digital converter. Maskinvare som konverterer analoge signaler til digitale. 8, 97

API Application Programming Interface. 2, 15, 107, 116

APN Access Point Name. Gateway mellom mobilnett og internett. 17

Arduino Mikrokontroller montert på kretskort, med tilkoblinger for ekstern maskinvare og åpen kildekode. iv, 1, 5, 15, 16, 18–20, 22, 27, 59–62, 75, 84, 88, 89, 92, 97–99, 113, 116, 118

ASIC Application-specific integrated circuit. 120, 121

AT commands Tekst kommandoer for å kommunisere med modem. Også kalt Hayes command set. 81

bil-pi Raspberry Pi som er montert på bilen. 66, 69, 79, 118

bitrate Antall bits som overføres pr. sekund. 11–14, 119, 120

DAC Digital-to-analog converter. Maskinvare som konverterer digitale signaler til analoge. 8

DC Direct Current (likestrøm). 8

dekoding Prosessen med å dekomprimere video så det kan vises på skjerm. 16, 69, 107, 114–116, 120, 122

edge computing Databehandling som foregår nært 'kanten' av nettverket. I motsetning til databehandling som foregår i sentraliserte datasentre. 11

enkoding Prosessen med å komprimere video så det kan sendes over nettverk. 16, 68, 70, 114–116, 120, 122

ESC Electronic speed control. 1, 18, 19, 21, 22, 58, 59, 62, 63, 79, 116, 117

FOV Field of view. Hvor bredt kamera filmer. 24

FPGA Field-programmable gate array. 120, 121

GPIO General-purpose input/output. Fysiske tilkoblingspinner som kan brukes til kommunikasjon med eksterne enheter. 20, 84, 90–92, 113, 114

GStreamer Modulbasert programvare som brukes til blant annet video prosessering. iv, 66, 68, 95, 115, 116

H.264 Videokodeken som benyttes i prosjektet. Også kalt AVC, MPEG-4 AVC eller MPEG-4 Part 10. 11–14, 68–70, 115, 116, 120, 121

HAT Hardware Attached on Top. Når HAT nevnes her menes 5G/4G modem som er montert på Raspberry Pi. 1, 15–17, 25, 27, 79, 81, 113–115

IP core intellectual property core: En gjenbrukbar krets eller logikk eid av et selskap. 121

I²C I²C eller I2C, Inter-Integrated Circuit. Protokoll for sending av data mellom enheter, typisk mikrokontrollere og tilkoblet utstyr. Halv duplex (kun kommunikasjon i en retning om gangen). 6, 7, 116

kjernenett I denne sammenhengen: den delen av mobilnettet hvor trafikk mellom tilkoblede enheter rutes. 10, 11, 17, 83

M.2 Fysisk tilkobling for kommunikasjon- og lagringsmoduler. 17

megabit per sekund Antall megabit som blir sendt i løpet av ett sekund. 6, 7

ms Millisekund, 1/1000 sekund. 1, 3, 10, 19, 66, 90–95, 100, 112, 115, 116, 119, 120, 122

OSI Open Systems Interconnection model (OSI model). En konseptuell modell oppdelt i 7 lag, som beskriver hvordan forskjellige nettverksprotokoller fungerer sammen. 9

PCB Printed circuit board. 76

pilot-pi Raspberry Pi som står på "pilot-siden" av systemet. Er tilkoblet ratt/pedaler og skjerm. 24, 66, 69, 80, 118

PWM pulse-width modulation. Firkantpulser som simulerer et analogt signal. 2, 5, 8, 18, 19, 22, 58–63, 84, 113, 116, 117

radiolinje Trådløs punkt-til-punkt kommunikasjon ved hjelp av mikrobølger. Høy kapasitet over avstander på flere titalls kilometer. 10

radionett I denne sammenhengen: den delen av mobilnettet som kommuniserer trådløst med telefoner/tilkoblede enheter. 10

Raspberry Pi Datamaskin på størrelse med et bankkort. Kjører typisk Linux. iv, 1, 2, 11, 15–20, 22, 24, 27, 28, 57, 66, 69, 73, 76, 77, 79–81, 83, 84, 90, 94, 97, 103, 105, 107, 113–117, 119–121

RTP Real-time Transport Protocol. Protokoll for sending av lyd/video over IP. Inneholder mekanismer for å håndtere tapte pakker. RTP pakkes typisk inn i UDP. 69

SCL Serial Clock Line. En av to tilkoblinger mellom enheter som kommuniserer over I^2C protokollen. 6

SDA Serial Data Line. En av to tilkoblinger mellom enheter som kommuniserer over I^2C protokollen. 6

service Et program som starter i bakgrunnen når systemet starter opp. 81, 82

SPI Serial Peripheral Interface. Protokoll for sending av data mellom enheter, typisk mikrokontrollere og tilkoblet utstyr. Full duplex (kommunikasjon i begge retninger samtidig). 5, 7, 16, 19, 53, 57, 60–64, 77, 84, 113, 116, 118, 119

SSH Secure Shell Protocol. Protokoll for kommunikasjon over nettverk. Brukes for å kjøre kommandoer på Raspberry Pi. 66, 79, 81

TCP Transmission Control Protocol. En protokoll for kommunikasjon over IP. Inneholder mekanismer som oppdager om pakker går tapt på veien, vil sende pakker på nytt om nødvendig. 9, 115, 119, 120

UDP User Datagram Protocol. En protokoll for kommunikasjon over IP. Vil ikke sende pakker på nytt, gir ingen garanti for at pakker når mottaker. 9, 115, 119

URLLC Ultra-reliable low latency communication. 5G som er optimalisert for ultra lav forsinkelse, og høy pålitelighet. 120

videokodek En kodek er ett sett med regler for hvordan enkoding/dekoding foregår (enKOde, DEKode). 116, 120, 121

VLC VLC media player. 94, 95, 115

VPN Virtual Private Network. Teknologi for å kommunisere sikkert mellom enheter over usikre nettverk. 66, 67, 69, 73

Kapittel 1

Introduksjon

I dette prosjektet har gruppen bestående av fire tredjeårs-studenter ved Elektroingeniørstudiet på NTNU (fordypning Elektronikk) laget en prototype for fjernstyring av en radiostyrt bil over Telenors 5G-nett. Oppdragsgiver for bacheloroppgaven er Yara. For Yara er dette en del av deres bidrag til EU prosjektet '5G-SOLUTIONS'. Yaras deltagelse i EU prosjektet har sitt utspring fra 'Yara Birkeland' prosjektet hvor autonomi er en viktig faktor [1][2].

Yara er use case eiere av EU prosjektet '5G-SOLUTIONS' sitt delprosjekt 'UC3.5: Autonomous assets & logistics for smart port', hvor man undersøker hvordan 5G kan benyttes for å automatisere og effektivisere logistikken i containerhavner [3]. Denne bacheloroppgaven ser nærmere bestemt på om fjernstyring av kjøretøy vil kunne fungere som et alternativ dersom autonome systemer trenger assistanse. UC3.5 delprosjektet inngår i 'LivingLab3 Smart Cities & Ports'. Yara er også deltakere i UC1.3, UC1.5 og UC3.6 [4].

Når man kommuniserer trådløst over 4G vil det være en forsinkelse på ca. 25 ms eller mer hver gang data overføres trådløst mellom basestasjonen og den tilkoblede enheten [5]. Denne forsinkelsen vil være betydelig lavere ved bruk av 5G. Under 10 ms vil være vanlig, rundt 1 ms vil være mulig å oppnå [6].
5Gs lave forsinkelse og høye kapasitet gjør det til en godt egnet overføringskanal for store mengder sanntids-video. Video med svært lav forsinkelse er et krav for at mennesker skal kunne manuelt fjernstyre kjøretøy og andre maskiner. I dette prosjektet var målet å teste hvor godt det fungerer å fjernstyre et kjøretøy over 5G-nettet, ved hjelp av video. Oppgaven vil kunne gi et 'proof of concept', men ikke et ferdig utviklet produkt. Rapporten dokumenterer arbeidet fra start til fungerende løsning.

Innenfor et budsjett på Kr 50.000,- skal det utvikles en måte å sende styringssignaler til en bil, og direktevideo fra bilen til en skjerm hos piloten. Bilen er en ombygd radiostyrt bil hvor mottaker er byttet ut med en Raspberry Pi i tillegg til en 5G-HAT for mobildekning. Mellom styringsservo/ESC er det en Arduino som

endrer styringssignalene sendt over 5G til PWM signaler som bilen forstår. Video fra kameraet enkodes på Raspberry Pi, sendes over 5G og dekodes på en annen Raspberry Pi ved piloten. Deretter vises video på en skjerm som gir piloten tilnærmet sanntids-video av hva bilen ser. De største utfordringene vil være forsinkelsen fra piloten styrer ratt eller pedaler til endringen vises på skjermen. For stor forsinkelse vil gjøre bilen veldig vanskelig å kontrollere.

Et godt resultat vil være når piloten har følelsen av god kontroll over kjøretøyet. En forutsetning for god kontroll vil være lav total forsinkelse i systemet og en lignende følelse som ved kjøring av en personbil. Dersom man mister dekning under kjøring eller det er et brudd i kommunikasjonen må bilen kunne merke dette og stoppe selv. Dette er for å gjøre kjøringen trygg for personer i nærheten av bilen, og fordi en kollisjon i moderat fart kan føre til store materielle skader, eller alvorlige skader på eventuelle personer i sammensøtet. Sluttrapporten tar for seg all relevant informasjon tilknyttet prosjektet, og alt av tester og resultater. Den skal gi god oversikt over alt som er gjort for å nå det resultatet som er nådd og skal gi grunnlag for repetisjon av oppgaven med samme resultat.

Rapportens struktur:

1. **Introduksjon:** En introduksjon til prosjektet og hva det går ut på.
2. **Krav:** En oversikt over mål satt før og underveis i prosjektet.
3. **Teknisk design:** Bakgrunnsteori, forklaring av maskinvare og programvare og hvorfor disse løsningene ble valgt.
4. **Prosess:** Arbeidsprosessen og verktøy som ble brukt for prosjektledelse og dokumentasjon.
5. **Implementering:** Mer tekniske detaljer brukt under oppgaven. Derunder biblioteker, språk, verktøy og API-er.
6. **Hvordan bruke løsningen:** En guide som forklarer hvordan man starter systemet, for å kunne kjøre bilen.
7. **Testing:** Detaljert beskrivelse av testmetoder brukt til å fremstille resultater.
8. **Resultater:** Fremlegg av resultater fra utførte tester.
9. **Diskusjon:** Drøfting rundt resultatene, løsningen som er valgt og selve utviklingsprosessen.
10. **Konklusjon:** Hva som er oppnådd i prosjektet. Videre eventuelle endringer som kunne vært gjort for å forbedre resultatene.

Kapittel 2

Krav til løsning

Kravene satt før prosjektets start var basert på teori lest under startfasen av prosjektet og hva oppdragsgiver ønsket å oppnå. Underveis i prosjektet ble forståelsen for hva som var mulig å oppnå klarere og noen krav/mål endret seg. Selv om den ferdige oppgaven ikke er innenfor alle kravene satt i starten kan det være et godt utviklet produkt som tilfredsstiller oppgaven gitt.

Ved prosjektets start ble det satt noen mål for hvordan den ferdige bilen skulle fungere:

- Total forsinkelse på video fra kamera til skjerm skal helst være under 100 ms over 5G.
- Total forsinkelse på styringssignaler vesentlig lavere enn videoforsinkelse.
- Bilen skal kunne styres over 4G/5G, uten visuell kontakt mellom piloten og bil.
- Bilen skal kunne fungere på både 4G og 5G, for å kunne sammenligne forsinkelse og video-stabilitet.
- Piloten skal ha følelsen av god kontroll over kjøretøyet.
- Stabil arkitektur for overføring av styrings- og videosignaler.

Målet for oppgaven er å utvikle et 'proof of concept' for et kjøretøy styrt over 5G. Kjøretøyet styres av ratt og pedaler i egen bilstol og skjerm, hvor video fra bil og styringssignaler fra pilot sendes over 5G. Bilstol, ratt og pedaler ble valgt for å gi en mest mulig virkelighetsnær kjøreopplevelse. Det skal også være mulig å bytte til 4G for å se forskjellene mellom 4G og 5G. For å gi piloten god kontroll over kjøretøyet bør total forsinkelse i systemet være under 100 ms. Mer forsinkelse enn dette kan gjøre kjøretøyet vanskelig å kontrollere. Ettersom kommunikasjonen mellom bil og pilot er over 4G/5G skal bilen kunne styres hvor som helst i landet, så lenge enten 4G eller 5G er tilgjengelig ved både pilot og bil. Et annet mål er å gi piloten følelsen av god kontroll over kjøretøyet.

Krav satt underveis i prosjektet:
For å teste forsinkelse i systemet skal en måling av videoforsinkelse kunne kjøres når bilen er stillestående, og generell test av forsinkelse i nettet skal kunne gjøres under kjøring.

Begrensninger:

- Tidsbegrensing på ti fulltidsuker og åtte deltidsuker med 40 prosent arbeidsbelastning.
- Basert på kommersielt tilgjengelig utstyr.
- Hele prosjektet har et budsjett på 50.000 kr.

Kapittel 3

Teknisk design

3.1 Bakgrunnsteori

I dette kapittelet vil det bli redegjort for teorien bak den tekniske løsningen, derunder bakgrunnsteori, maskinvare og programvare.

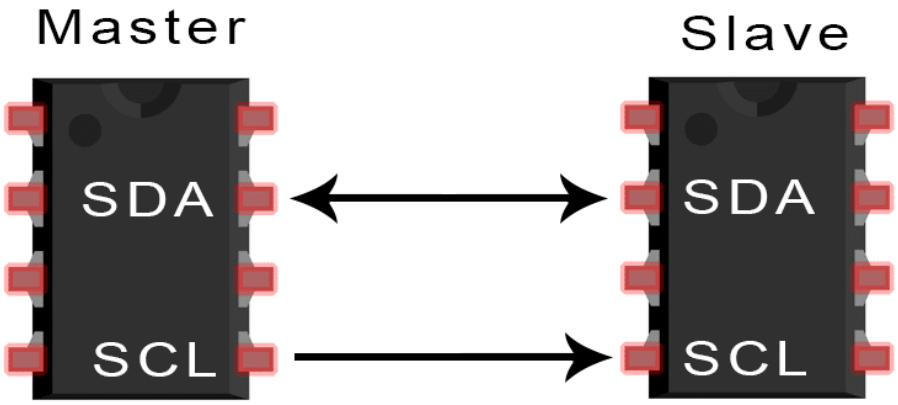
3.1.1 Mikrokontroller

Microchip Atmega328p

Brukes av blant annet Arduino Uno, som er Arduinoen brukt til testing i starten av prosjektet. Den har en 16-bit teller med 2 utganger og en 8-bit teller. 8-bit teller gir for lav oppløsning på PWM signalet til formålet og en av utgangene på 16-bit teller er utilgjengelig når Arduinoen står i SPI slave modus. Atmega328p har en 16MHz klokke [7].

Microchip Atmega4809

Brukes av Arduino Nano Every, mikrokontrolleren på det ferdige designet. Har to 16-bit tellere hvor den som skal brukes har 3 utganger. På Atmega4809 kan utgangspinnene for PWM bestemmes i kode og den er dermed veldig fleksibel. Atmega4809 bruker andre register enn Atmega328p og koden blir derfor veldig forskjellig [8].



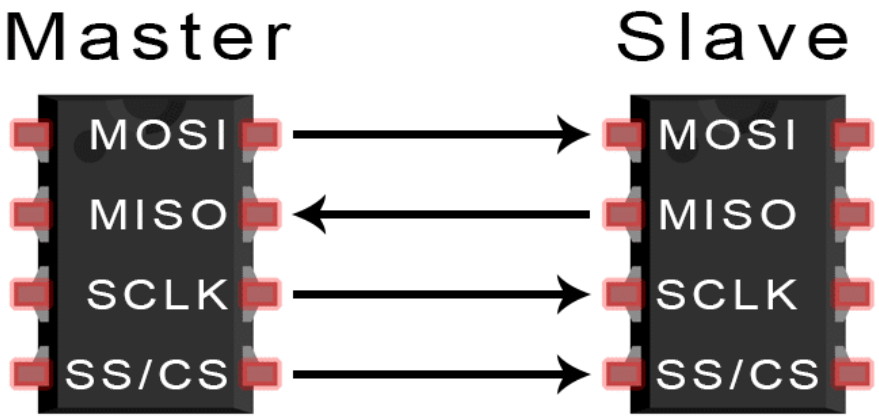
Figur 3.1: I2C kommunikasjon [9].

3.1.2 I²C/SPI

I²C:
Som vist i figur 3.1 er I²C en kommunikasjonsprotokoll som bruker to datalinjer for å kommunisere mellom en master og opptil flere slaver. Dette kan f.eks. være mellom en mikrokontroller og tilkoblet utstyr. Datalinjene er SDA (Serial Data Line) for dataoverføringen og SCL (Serial Clock Line) for klokkesignal. Det må også være en felles jord mellom master og slave. For å skille mellom slaver brukes en 7-bit adresse og maksimal hastighet er 3,4 megabit per sekund. Kommunikasjon mellom master og slave starter med at master sender en START melding til alle slaver den er koblet til. Dette gjør at slavene lytter på SDA. Deretter sendes adressen master ønsker å kommunisere med, og om adressen stemmer med adressen på slaven vil enheten sende et BEKREFT signal tilbake [9][10].

Om master skal motta data setter den RD/nWR(Read/n*Write) høy og slaven begynner å sende data etter sendt BEKREFT. I dette tilfelle må master bekrefte mottatt data mellom hver byte. Klokkesignalet bestemmer når data på SDA kan endres, mer spesifikt kan dataen kun endres når SCL er lav, men holder seg stabil når den er høy [9][10].

Dataoverføringen benytter en 'pull-up resistor' for å sette bits. For å sette en 0 settes linjen til jord og for å sette en 1 gjøres det ingenting. Dette gjør at dersom en slave sender 0 og en annen sender 1 vil ikke noe kortsluttes fordi 1 vil gå til jord og mottatt verdi blir 0. Master kan også lytte på SDA samtidig som den sender, og dersom den leser 0 når den sender 1 vet den at det er et problem en plass. Etter at kommunikasjonen mellom master og slave er ferdig sendes et STOP signal til alle slaver og de begynner å lytte etter et START signal igjen [9][10].



Figur 3.2: SPI kommunikasjon [11].

Hovedtrekket til I²C er 'clock stretching'. Dersom slaven opererer saktere enn klokkefrekvensen satt av master kan slaven holde igjen klokkesignalet. Ettersom master kun leser av data når klokken blir høy vil slaven, ved å holde klokken lav, få lenger tid til å sende data. Dette forhindrer pakketap og feil i data, men kan begrense hastigheten på overføringen betraktelig [9][10].

SPI:

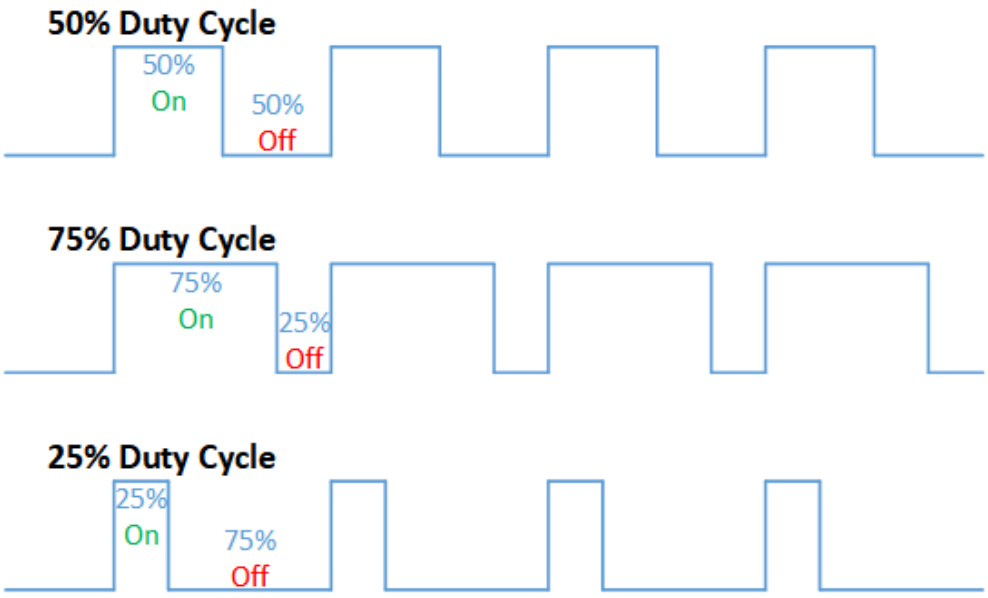
SPI bruker fire datalinjer for kommunikasjon mellom master og slave som vist i figur 3.2:

- SCLK (Serial clock) for klokkesignal
- MOSI (Master out-Slave inn) for data ut fra master og inn på slave
- MISO (Master inn-Slave out) for data inn på master og ut på slave
- SS (Slave Select) for valg av slave

Master må ha en SS pinne for hver slave som kobles til, men ved kun 1 slave er det ikke alltid nødvendig med SS. Det er fire ulike modus for dataoverføring avhengig av hvilke klokkeflanker data skal sendes på. Ettersom det er egne datalinjer for sending og mottak trenger ikke master å bekrefte sendt eller mottatt data. Master kan bare sende data uten å vite om det kommer fram, og data kan sendes og mottas samtidig. SPI er altså 'full-duplex', noe I²C ikke er. Den største fordelen med SPI er at data kan sendes så ofte som nødvendig og er kun begrenset av hvor fort programmet kan motta/sende dataene, ofte over 10 megabit per sekund. Dette kan føre til pakketap istedenfor redusert hastighet i sendingen [11][10].

3.1.3 PWM

PWM (pulse-width modulation) er en måte å styre analoge systemer med digitale signaler. Det digitale signalet simulerer et analogt signal ved hjelp av tellere og en DC kilde som enten er helt på, eller helt av. Et PWM signal vil være en firkantbølge, amplituden varierer mellom 0 og 100%. Spenningen ut bestemmes av signalets duty cycle. F.eks. vil en 10% arbeidssyklus på en 9 volts kilde gi et signal på 0.9V. Dette vises i figur 3.3. Arbeidssyklussen bestemmes av en teller og en klokkefrekvens. Telleren teller etter klokkefrekvensen og resettes til 0 når den når en toppverdi. Denne toppverdien settes for å oppnå ønsket frekvens. Det benyttes også en 'prescaler' som senker klokkefrekvensen for å senke nødvendig toppverdi så det passer i en 16-bit teller. En 16MHz klokke med ønsket frekvens på 50Hz og prescaler på 8, vil etter formelen
$\frac{Klokkefrekvens}{1000000} * \frac{periodetid\ i\ ns}{prescaler}$
ha en toppverdi på 40 000. For å bestemme hvor mye av signalet som skal være 'på' sammenlignes tellerverdien med et register. Med 10% arbeidssyklus på signalet over må den være på for verdier opp til 40 000*0.1 = 4000 og av for de resterende 90% av verdiene[12][13].



Figur 3.3: Eksempel på PWM signal [14].

En av fordelene med et PWM signal over et analogt signal er liten påvirkning fra støy. Dette er fordi signalet enten er 1 eller 0 og støy må derfor kunne endre en 1 til en 0 eller motsatt for å ha noen effekt. Det er heller ikke nødvendig med ADC eller DAC i systemet [12][13].

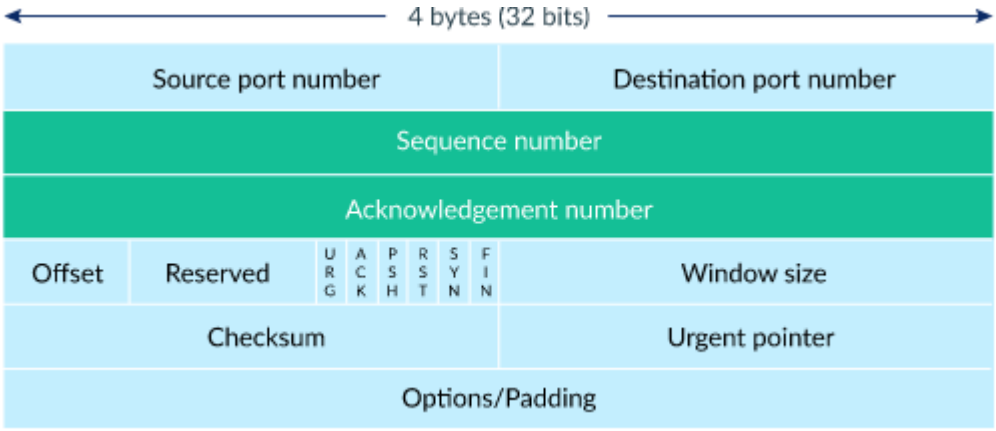
3.1.4 Nettverksprotokoller

IP - internett protokoll

Internet Protocol (IP). Protokollen som danner grunnlaget for internett. Finnes i to versjoner, IPv4 og IPv6. En IPv4 adresse er bygget opp av fire byte. IPv4 adresser brukes for ruting av trafikk mellom bil/pilot i dette prosjektet.

TCP

Transmission Control Protocol (TCP) – OSI lag 4 transport TCP er en type transportprotokoll som blir brukt for en pålitelig overføring av datapakker. TCP blir mest brukt i sammenheng med IP, da ofte referert til som TCP/IP. TCP har en rekke mekanismer for å unngå pakketap, duplikatpakker eller korrupte pakker.



Figur 3.4: Illustrasjon av en TCP-header [15].

Hvert transmisjonssegment inneholder en header som normalt er på 20 byte, den kan maksimalt bli opp til 60 byte. Denne størrelsen er avhengig av hvilke header alternativ som er flagget høy, som vist i figur 3.4 [15][16].

UDP

User Datagram Protocol – UDP – OSI modell lag 4 Transport

UDP er en transportprotokoll som blir ofte brukt sammen med IP (Internett protokoll) ofte referert til som UDP/IP. UDP har en sjekk for at dataene ikke er korrupt (data mottatt er samme data som ble sendt) [16].

3.1.5 Mobilnett

Grunnleggende om mobilnett

Mobilnettet består av flere ledd som alle må spille på lag for at man skal kunne bruke nettet som forventet. Telefoner og annet utstyr (brukerterminaler) kommuniserer trådløst med en basestasjon. Basestasjoner og antenner kalles på fagspråket for Radio Access Network (RAN), eller radionett. I Telenors mobilnett alene er det over 8000 basestasjoner [17]. Telia og andre operatører har egne basestasjoner, men utstyr fra de forskjellige operatørene kan i en del tilfeller være fysisk plassert i samme bygg/mast.

Når brukerterminalen har sendt datapakkene trådløst til basestasjonen, må datapakkene sendes videre ut i verden. Her benyttes som regel fiberoptiske kabler, eller radiolinje på steder hvor det er vanskelig å legge fiber. Trafikken fra mange forskjellige basestasjoner samles sammen i en datastrøm, og transporteres i fiberoptiske kabler til 'kjernen' i mobilnettet. Dette kalles transportnett.

I kjernen av mobilnettet sitter utstyr som blant annet sørger for å koble opp samtaler, rute IP trafikk og som holder oversikt over tilkoblede enheter. Det er dette som kalles kjernenett. I Telenors tilfelle ligger dette i datasenter i Osloregionen.

Hva er nytt i 5G

Med hastigheten man får over 4G kan man i mange tilfeller sende video i sanntid. En av de store ulempene med 4G er at forsinkelsen er høyere enn om trafikken hadde gått over fiberoptiske kabler hele veien.

For å kunne styre kjøretøy og maskiner i sanntid er det svært viktig at forsinkelsen er så lav som mulig. Dette gjelder uavhengig om det er mennesker som skal styre ved hjelp av video, eller datamaskiner/algoritmer som tar beslutninger basert på video eller andre sensordata. Det er forventet at forsinkelsen i radionettet i 5G vil ligge på 10 ms eller lavere ved mange bruksområder, helt ned i <1 ms for spesielle bruksområder [6]. På 4G kan forsinkelsen være nede i 20-25 ms, men den kan også variere og være mye høyere [5][2]. Den økte kapasiteten i 5G-nettet gjør det også mulig å sende et større antall videoer samtidig.

Endringene fra 4G til 5G er ikke bare i radionettet, 5G vil på sikt også innebære en total forandring av kjernenettet. I overgangen til 5G vil det finnes to forskjellige former for 5G. Den ene kalles 'non-standalone', det betyr at man benytter 5G radionett (5G New Radio), kombinert med samme kjernenett som benyttes til 4G. Den andre formen er 'standalone', som betyr at man også benytter 5G kjernenett [18]. Ved 5G standalone vil for eksempel to enheter som er tilkoblet samme basestasjon kunne kommunisere med hverandre ved å sende trafikk kun via base-

stasjonen, eller via datasenter i nærheten (edge computing). I dagens 4G-nett går denne trafikken via kjernenettet i Osloregionen.

5G-nettet som er bygget ut ved NTNU i Trondheim er Telenor Norges kommersielle 5G-nett. Pr. april 2021 er det av typen non-standalone kjernenett, det vil si samme kjernenett som for for 4G. Selv om bilen og piloten er koblet på samme basestasjon må IP pakkene sendes via én av flere 'packet gateways' som styres av mobilnettets kjernenett. Datatrafikk fra brukerterminaler i Trondheim rutes i dagens kommersielle 5G-nett via en packet gateway i Oslo, og går tilbake samme vei. Ved Yaras anlegg på Herøya i Porsgrunn testes det standalone 5G kjernenett fra Telenor Research (5G-VINNI) [19]. I et slikt nett vil packet gateways kunne etableres lokalt på Herøya slik at datatrafikk rutes lokalt. Bilen har så langt bare blitt testet på non-standalone nettet i Trondheim.

3.1.6 Video enkoding

Om video enkoding

Råvideo fra kamera har en veldig høy bitrate. For eksempel vil en video med oppløsning på 720x480 piksler, 24 bit farger (RGB), 30 fps gi:

$$\frac{720x480x24x30}{1024^2} = 237\frac{Mb}{s}$$

Som formelen viser gir dette en bitrate på 237 megabit per sekund. Det kan fungere å ha en enkelt videostrøm med denne bitraten over 5G, men det oppstår raskt kapasitetsproblemer om det sendes et større antall videoer.
For at det skal være praktisk mulig å sende video over 5G må videoen komprimeres før den sendes. Dette gjøres ved at video enkodes før den sendes, og dekodes ved avspilling. I dette prosjektet benyttes videokodeken H.264, som er en veldig utbredt kodek. Raspberry Pi har støtte for maskinvare-akselerert enkoding/dekoding av H.264, noe som betyr at det er egen maskinvare som er designet utelukkende for å enkode/dekode video, uten at dette belaster CPU [20].

Antall farger	1280x720 30fps	1920x1080 30fps
8 (3 bit)	79 Mb/s	178 Mb/s
64 (6 bit)	158 Mb/s	356 Mb/s
512 (9 bit)	237 Mb/s	534 Mb/s
4096 (12 bit)	316 Mb/s	712 Mb/s
32K (15 bit)	396 Mb/s	890 Mb/s
262K (18 bit)	475 Mb/s	1068 Mb/s
16.7M (24 bit)	633 Mb/s	1424 Mb/s

Tabell 3.1: Råvideo, bitrate ved forskjellig oppløsning og fargedybde (RGB)

Tabell 3.1 viser bitraten til råvideo, ved forskjellig fargedybde. Det illustrerer at fargeinformasjon utgjør en stor del av bitraten.

Videokodeker som H.264 er designet for blant annet å gjøre det mulig å sende video over internett, og er dermed optimalisert for å gi god bildekvalitet ved forholdsvis lav bitrate. Bildekvalitet, bitrate og forsinkelse er parametere som henger sammen. Ved å optimalisere for god bildekvalitet og lav bitrate øker forsinkelsen.

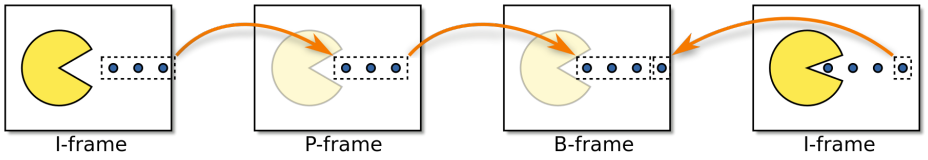
Om H.264

I H.264 enkodet video benyttes en rekke teknikker for å begrense videoens bitrate, samtidig som bildekvaliteten opprettholdes. Videoenkoding er et eget fagfelt, dette er bare en overordnet beskrivelse av hvordan enkoding fungerer.

Bevegelse Man kunne sendt video som en serie med bilder, men dette er veldig lite effektivt hvis det bare er endring i deler av bildet. Om videoen for eksempel viser en person som snakker er det ingen grunn til å stadig sende nye bilder av en statisk bakgrunn. Om bildet inneholder ett objekt som beveger seg kan det være mer effektivt å sende informasjon om retningen objektet beveger seg, istedenfor å sende helt nye bilder.

I H.264 sender man ikke alltid hele bildet, men man sender nok informasjon til at hvert enkelt bilde kan rekonstrueres hos mottaker. Det benyttes tre forskjellige typer 'frames'. I-frame, P-frame og B-frame [21]. De forskjellige typene frames inneholder forskjellig type informasjon.

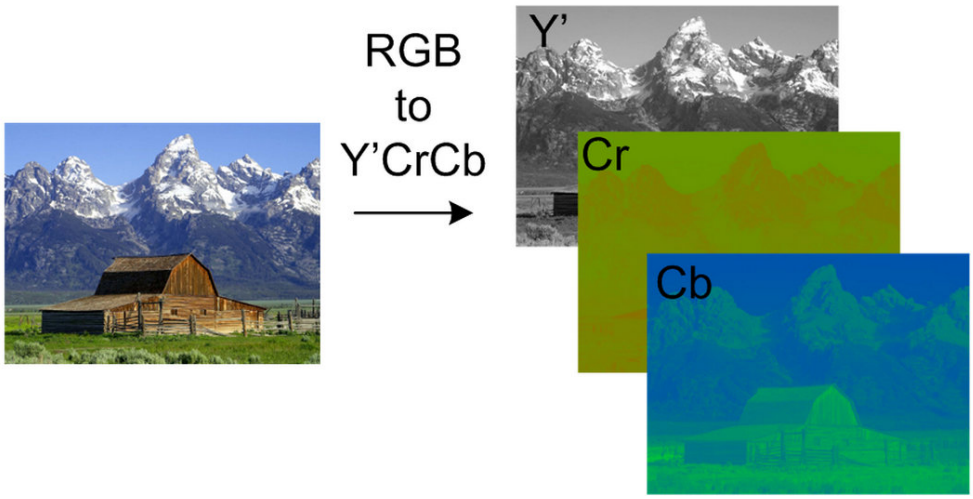
- I-frame: En I-frame inneholder all informasjonen som kreves for å tegne ett enkelt bilde på skjermen
- P-frame: P-frame inneholder endringer siden forrige I-frame
- B-frame: B-frame inneholder endringer i forhold til forrige og neste I-frame



Figur 3.5: Illustrasjon av I-frame, P-frame og B-frame [22].

Figur 3.5 illustrerer forskjellige typer frames benyttet i H.264 enkodet video. For at videodekoderen skal kunne rekonstruere ett bilde fra mottatt B-frame er man avhengig av å også ha både forrige og neste I-frame. Hvis dekoder mottar sekvensen IBBP må dekoder vente til den får I-frame fra neste sekvens, før denne sekvensens B-frames kan vises på skjermen. Dette gjør at innkommende data må mellomlagres, og medfører en ekstra forsinkelse.

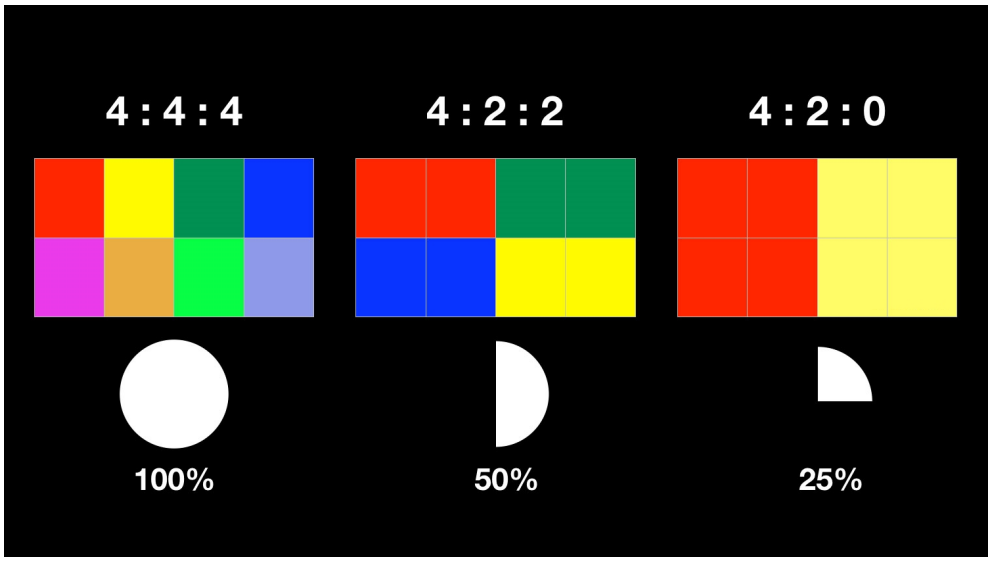
Farger Som vist i tabell 3.1 krever det høy bitrate å sende uenkodet video med nøyaktige farger. Det menneskelige øyet er veldig dårlig på å se forskjellen på små fargenyanser [23]. Dette utnytter man ved å ikke sende fargeinformasjon for alle pikslene, men istedenfor bruke samme farge på piksler som ligger ved siden av hverandre [24]. Dette kalles 'chroma subsampling'.



Figur 3.6: YCbCr farge enkoding [25].

Figur 3.6 illustrerer hvordan fargebilder kan deles opp i flere deler som hver for seg inneholder informasjon om enten lysstyrke eller farge. Måten dette gjøres på er at man splitter bildet opp i tre 'kanaler'. En Y kanal for lysstyrke (luma), og Cb og Cr kanaler for fargeinformasjon (chrominance/chroma). Y kanalen kan sendes

i full oppløsning, mens fargekanalene kan sendes i lavere oppløsning, noe som gjør at videoens bitrate blir betydelig lavere. Dette er ikke unikt for H.264 kodeken, men brukes nesten universelt [26][27].



Figur 3.7: Chroma subsampling [28].

Resultatet blir at fargen fra en enkelt piksel også blir brukt på pikslene rundt. Figur 3.7 illustrerer dette. Kakediagrammet på figuren illustrerer hvor stor andel av den opprinnelige bitraten som kreves for å overføre farge informasjon.

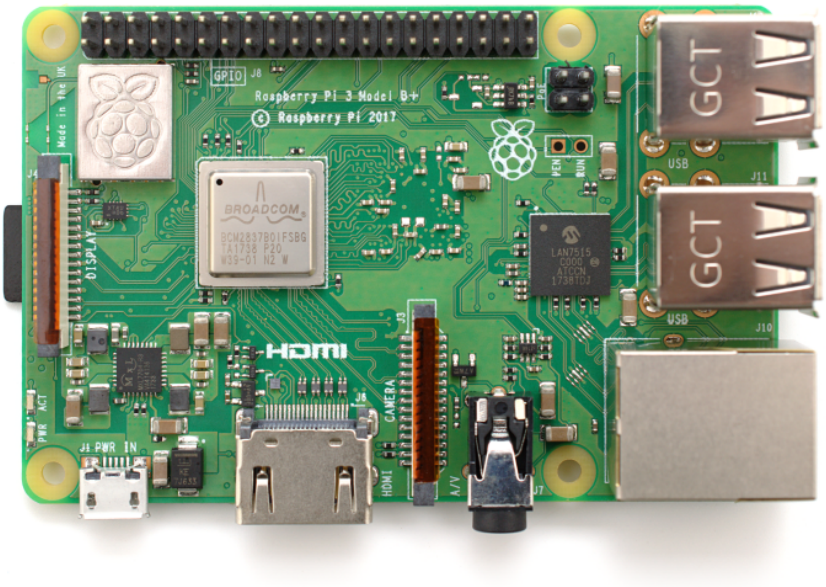
3.2 Maskinvare

Dette delkapittelet tar for seg beskrivelse av maskinvare brukt i prosjektet. Derunder Raspberry Pi, maskinvare for 5G kommunikasjon, Arduino, kretskort, kjøretøy og 3D-print.

3.2.1 Raspberry Pi

Prosjektet er bygget rundt ettkorts datamaskinen Raspberry Pi som vist i figur 3.8. Raspberry Pi er i praksis en fullt fungerende PC, i et veldig kompakt format (på størrelse med et bankkort). Raspberry Pi inneholder alt det som kreves av en enkel datamaskin, CPU, GPU, RAM, tilkoblinger (USB, Ethernet, HDMI osv.). På bilen brukes modell 4B, på pilot-siden brukes modell 3B+. Dette er på grunn av at 4B ikke støtter OpenMAX API for dekoding av video, som gir best ytelse.

Det finnes flere forskjellige operativsystemer som er modifisert for å kjøre på Raspberry Pi, her brukes Raspberry Pi OS, som er det offisielle operativsystemet fra Raspberry Pi Foundation. Raspberry Pi OS er basert på Linux distribusjonen Debian.



Figur 3.8: Raspberry Pi 3B+ [29].

Raspberry Pi-ene på bil og pilot-side fungerer som hjernen i systemet. På pilot siden er Raspberry Pi tilkoblet ratt og pedaler via USB, og skjerm over HDMI. På begge sider er Raspberry Pi tilkoblet 5G-HAT via USB.

Raspberry Pi på bilen kommuniserer med Arduino over SPI-protokollen. Kameraet er koblet til CSI-port på Raspberry Pi.

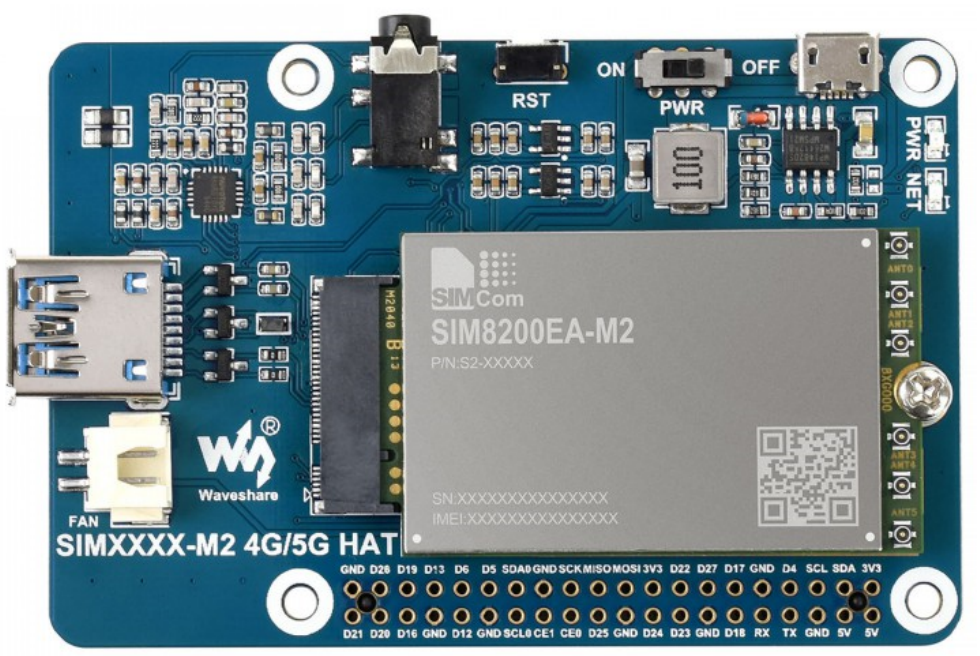
Det er flere grunner til at Raspberry Pi ble valgt som platform for prosjektet. Raspberry Pi har forholdsvis kraftig maskinvare, med støtte for maskinvareakselerert enkoding/dekoding av video, i et svært kompakt format.

Siden den første modellen av Raspberry Pi ble lansert i 2012 er det solgt over 40 millioner eksemplarer [30]. Med en slik popularitet, og millioner med aktive brukere er det i dag en relativt moden platform. Eventuelle problemer er stort sett løst for lengst, og det finnes svært mye informasjon på nettet omkring bruk av Raspberry Pi. 5G-HAT som ble benyttet er designet for å kunne monteres på Raspberry Pi, og alle gruppemedlemmene hadde kjennskap til Raspberry Pi fra før av. Dermed var Raspberry Pi et naturlig valg.

Spesifikasjoner for Raspberry Pi 3B+ og 4B ligger i vedlegg B.1 og B.2.

3.2.2 5G kommunikasjon

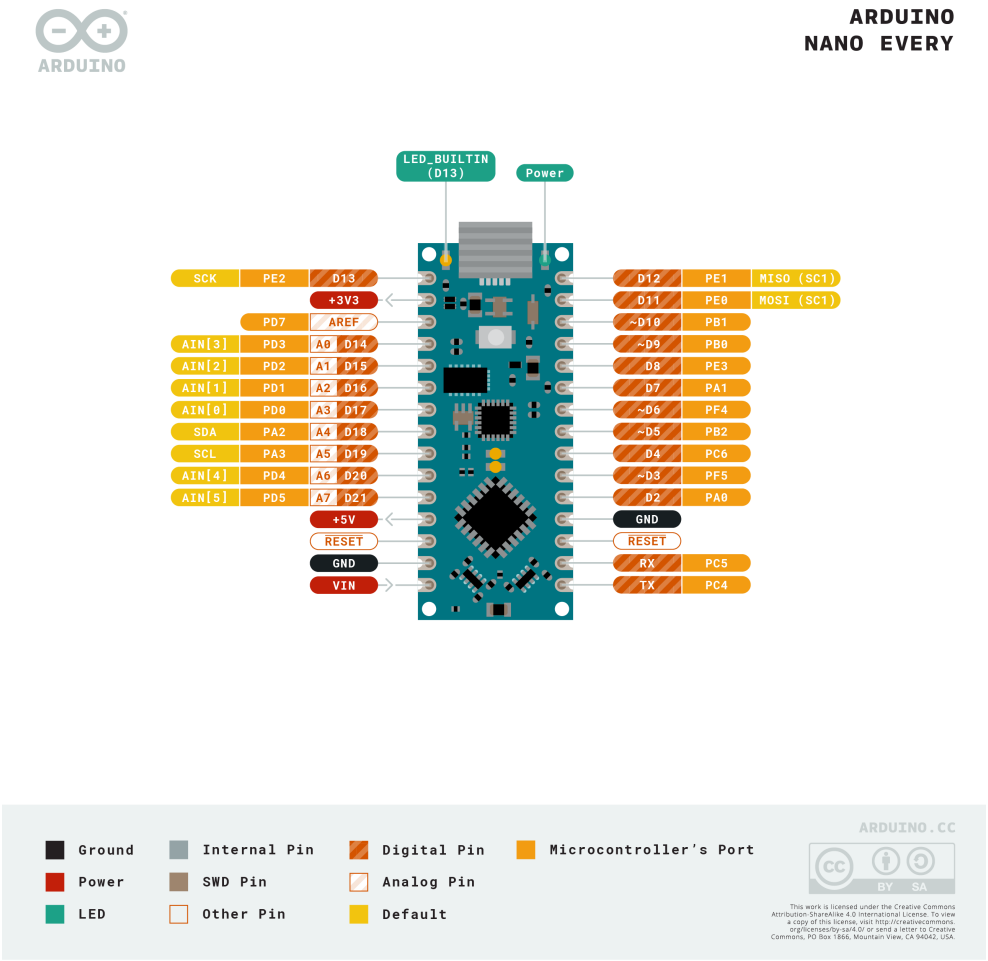
For å kommunisere over 5G/4G brukes modulen 'SIM8200EA-M2 5G HAT' fra Waveshare, som vist i figur 3.9[31]. Dette er en såkalt 'HAT' (Hardware Attached on Top) som monteres på Raspberry Pi. Modulen består av ett kretskort med USB port for tilkobling til Raspberry Pi/PC, og M.2 spor som 5G modulen monteres i. Modulen fra Waveshare kommer med 'SIM8200EA-M2' inkludert. Denne modulen fra SIMCom er selve modemet som kommuniserer med 5G/4G basestasjonen.



Figur 3.9: 5G-HAT [31].

Kommunikasjonen mellom Raspberry Pi og 5G modulen foregår over USB. For å koble Raspberry Pi til mobilnettet kjøres et skript fra Waveshare. Ved kjøring av skriptet oppgis APN og SIM kortets PIN-kode som parameter. Korrekt APN må benyttes for at Raspberry Pi skal bli tildelt fast, offentlig IPv4 adresse. Fast IP er nødvendig for at Raspberry Pi-ene skal kunne kommunisere direkte seg imellom (via kjernenett).

3.2.3 Arduino



Figur 3.10: Arduino Nano Every [32].

For styring av bilen skal det brukes en Arduino. Arduino er en programmerbar mikrokontroller som kan brukes til styring av sensorer, aktivering av dioder eller i dette tilfelle, styre hastighet og retning på en motor. Arduinoen er bedre egnet for dette bruksområdet enn en Raspberry Pi fordi Raspberry Pi kun har èn PWM utgang tilgjengelig som standard. Gruppen er også godt kjent med PWM på Arduino og det lar pien fokusere på sending av data over 5G. Arduinoen er i tillegg bedre sikret mot høy spenning fra ESC om det skulle bli et problem. Signalet fra ratt/pedaler er en 8 bit digital verdi, men ESC/styringsservo krever et PWM signal. Derfor må det være et ledd som gjør denne konverteringen. Alle på gruppen har jobbet med Arduino før, noe som gjør den til det naturlige valget for mikrokontroller.

I den ferdige bilen brukes en Arduino Nano Every med 16/20 MHz klokke og tre PWM utganger på TCA0, som vist i figur 3.10. TCA0 er en intern teller på Arduinoen som ittereres med 16/20MHz klokke. Ved hjelp av denne telleren kan det lages et PWM signal med valgfri frekvens og duty cycle som vist i teoridelen. Med 16 bit teller er det lett å oppnå en periodetid på 10ms (100Hz) som både ESC og styringsservo krever for normal drift. Det gir også mange flere verdier enn ved en 8 bit teller, noe som gir mer finjusterte utgangsverdier. Den har tre utganger på 16 bit teller og hvilke utganger som hører til kan endres i kode. Dette gir god fleksibilitet ved design av andre deler til bilen. Koden til Arduinoen er hovedsaklig skrevet i AVR-C noe som gjør at oppgaven ikke blir låst til bibliotek[8].

3.2.4 Motorstyring/Servostyring

En Arduino Nano Every på bilen får styringssignaler fra Raspberry Pi over SPI, og avgjør hva piloten har endret på og med hvor mye. Deretter bruker Arduinoen mottatte verdier for å generere et PWM signal og bestemme på hvilken port PWM signalet skal ut på. Dette signalet sendes til ESC som er en XL-5 fra TRAXXAS og en 2075 servo fra TRAXXAS[33][34]. En bil med disse komponentene ble valgt fordi det finnes mye støttelitteratur og de er lett tilgjengelige om noe skulle bli ødelagt under testing. ESC på bilen sender tar inn styringssignal fra bilen og kontrollerer spenningen inn på motoren og dermed hastighet og retning. ESC gir også 6V til servoen.

3.2.5 Kretsdesign

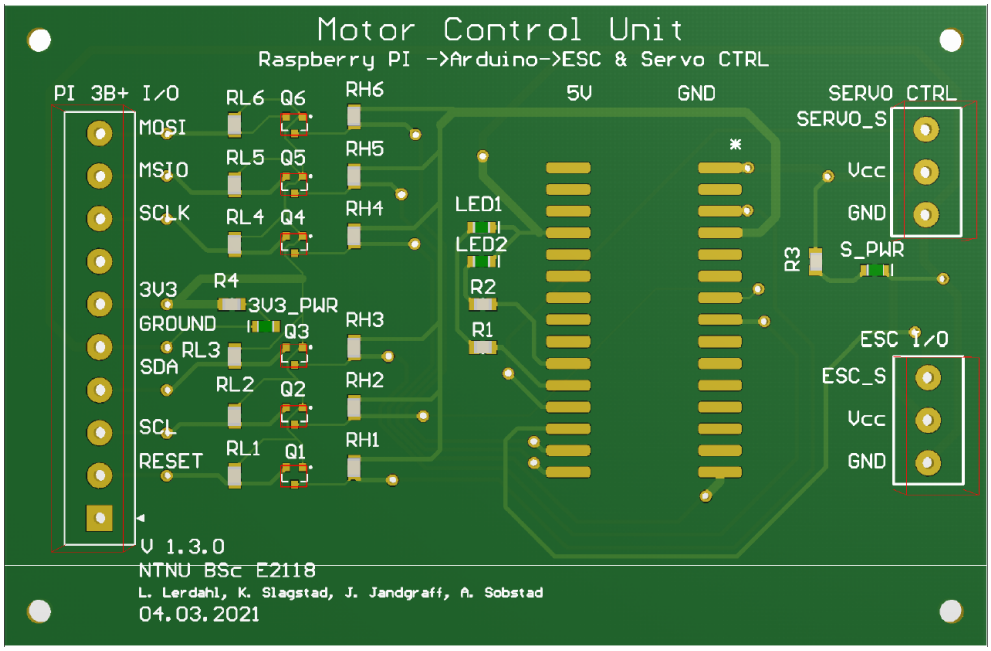
For kommunikasjon mellom Raspberry Pi og Arduino kreves det logikk-konvertering som gjør om fra 5V til 3,3V logikk. Dette er fordi Arduino bruker 5V logikk og Raspberry Pi bruker 3,3V. 5V inn på Raspberry Pi kan føre til skade på komponenter [32][35]. For å implementere logikk-konvertering, og videre ha robuste koblinger til ESC og styringsservo ble ett eget kretskort designet.

Kortet ble designet for å kunne produseres lokalt på Elektronikk- og Prototype-labben ved Institutt for elektroniske systemer. Dette medførte noen begrensninger i designet [36]. Derunder:

Produksjon med Mekanisk CNC-fres hos Elektronikk- og Prototype-labben

- To-lags kort.
- Ingen loddestopplag eller silketrykk.
- Minimum baneavstand: 0,21mm.
- Minimum drill/hole size: 0,9mm.
- Kobbertykkelse: 35um.
- Håndlodding av komponenter.

3.2.6 Kretskort-utlegg



Figur 3.11: PCB-utlegg - Fremside.

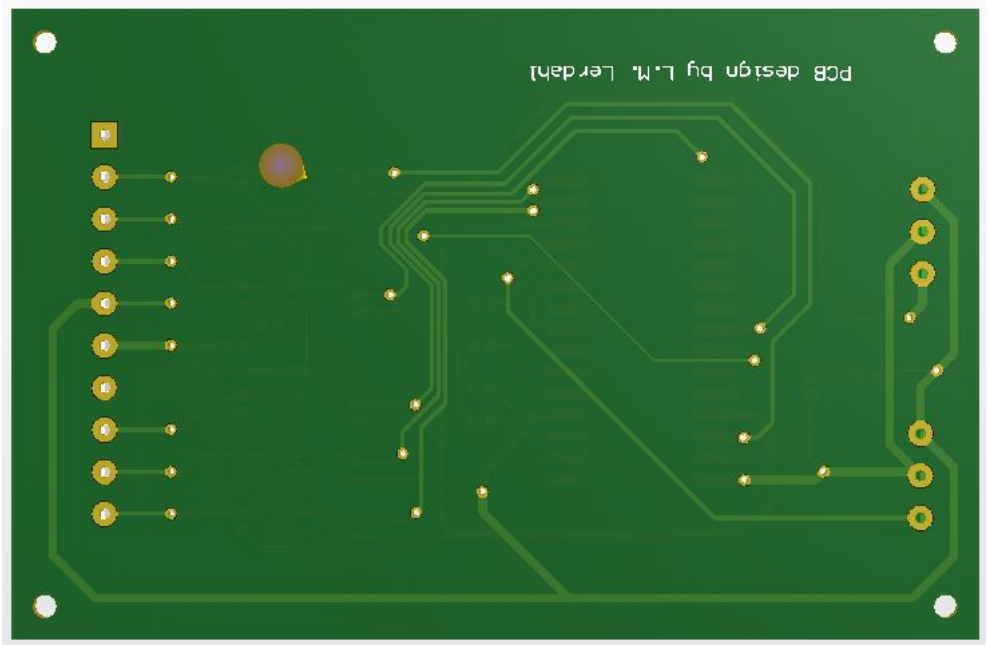
Kretskort-utlegget beskrevet fra venstre til høyre. Innganger på venstre side av kretskortet med mulighet for montering av skruklemmer eller andre innfestninger med 5mm avstand mellom pinner og pinnestørrelse under 1,4mm.
Disse inngangene kobles de tilhørende GPIO-pinnene på Raspberry Pi.

Logikk-konverteringskrets

Neste del av kretsen er logikk-konveteringen, som er realisert ved bruk av to $10k\Omega$ SMD-motstander og en N-Channel MOSFET, av typen BSS138 i SMD-størrelse fra ON Semiconductor. Logikk-konverteringen er basert på 'open hardware' fra Sparkfun og designet etter deres referanse-design. Konverteringskretsen er av bi-directional type, som vil si at signaler kan gå begge veier å konverteres til riktig tilhørende nivå[37].
Lav-siden av kretsen, altså den siden som er koblet opp til Raspberry Pi med 3,3V logikk har også tilførselen sin fra Raspberry Pi, for å sikre riktig spenningsnivå og holde sidene separate. Høy-siden får tilførselen sin fra Arduinoen med 5V logikk. På kortet er det en konverteringskrets per signalbane.

Signal og tilførselsbaner

Signalbanene går videre fra logikk-konverteringen til undersiden av kortet ved hjelp av vias.



Figur 3.12: PCB-utlegg - Bakside.

Banene går da på undersiden av kortet frem til neste via, hvor de så går opp til oversiden av kortet og kobles de til tilhørende terminalene.
På kortet er det brukt to forskjellige banebredder. En banebredde for signaler og en for tilførselsbaner. Førstnevnete er dimmensjonert til 0,254mm, eller 10 mil, som samsvarer med generelle anbefalinger for signalbaner [38]. Tilførselsbanene er er dimensjonert etter største strømgang, som i dette tilfellet går via Vcc-banen fra ESC-tilførselen til servo-utgangen. Dette strømtrekket ble målt med multimeter til 1A.

For å kalkulere banebredde benyttes kalkulator hos 7pcb[39]. Parametrene brukt er basert på kobbertykkelse oppgitt av prototypelabben, samt tommelfinger-regel for 'temp rise' eller temperaturstigning på norsk [36][39].

$$Current = 1A$$

$$CuThickness = 35\mu m$$

$$Temprise = 10C$$

$$Ambienttemp = 25C$$

$$Conductorlength = 5mm$$

$$Peakvoltage = 7.4V$$

Som ga følgende resultat:

$$Requiredtracewidth = 0.78mm$$

Nødvendig banebredde, 'requierd trace width' kalkuleres til 0,78mm.
tilførselsbanene settes til 1,016mm, med en liten margin over kalkulert banebredde.

Arduino Nano Every

Alle signalbanene er videre koblet til de tilhørende inngangene på Arduinoen. Arduinoen har utganger av 'castellated' type som gjør at kortet kan loddes fast på terminaler, uten gjennomgående ben.
Videre får Arduinoen tilførselspenning fra ESC via inngangen Vcc. Vcc fungere som tilførsel til både Arduino og servo. PWM-signalene fra Arduinoen som kontrollerer ESC og servo er koblet til vær av sine utganger på høyre side av kortet. Den totale signalgangen på kortet går da fra høyre til venstre.

Indikatorer

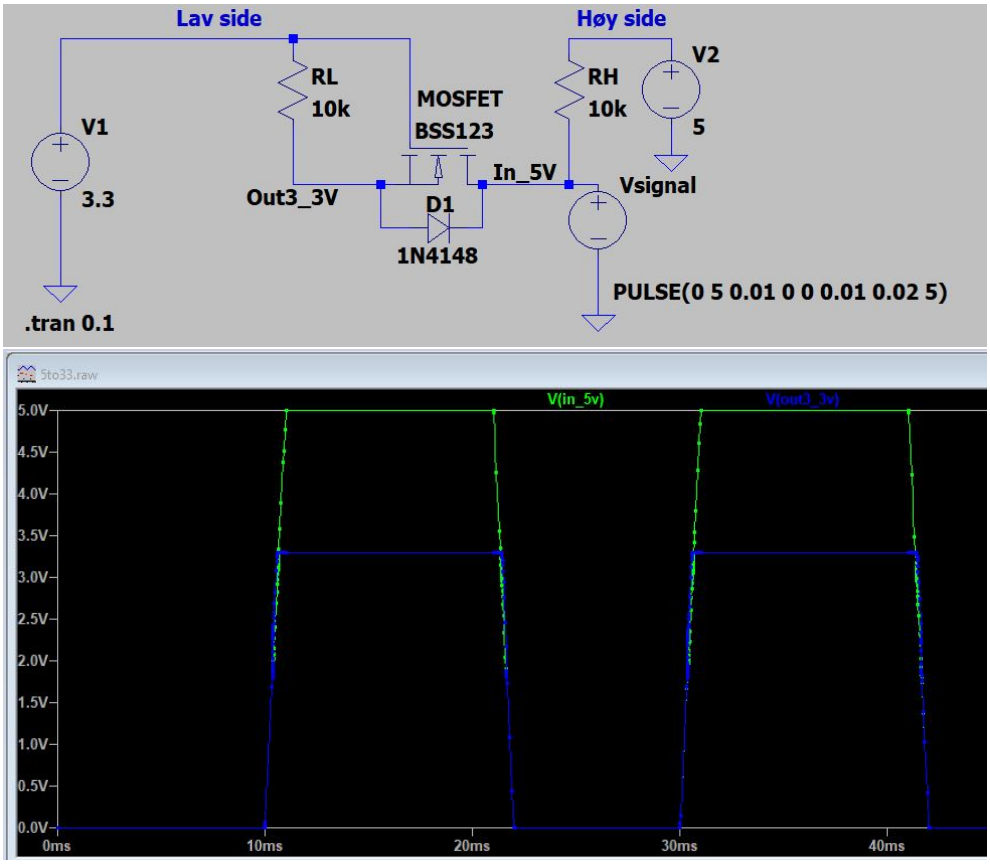
Utover indikator-dioder som er montert på Arduino Nano Every, har utlegget fire andre indikator-dioder i SMD-størrelse. 3V3_PWR, som er koblet til 3,3V banen og indikerer om tilførsel fra Raspberry Pi er i drift. S_PWR har samme funksjon, men indikerer tilførsel fra ESC. De siste to lysdiodene er koblet til digital-utganger på Arduinoen og kan kontrolleres med programkode.

3.2.7 Skjematisk fremstilling av kretsen

Skjematisk fremstilling av kretsen ligger som vedlegg til denne rapporten. Se vedlegg I.

3.2.8 Simulering og verifikasjon før produksjon

Før produksjon ble logikk-konverteringen simulert og testet i LT-spice for å se at ønsket funksjonalitet ble oppnådd.



Figur 3.13: Simuleringskrets - LT-spice.

Som det fremgår av figur 3.13 bekreftes funksjonaliteten av målingene gjort i punkt in_5V markert i grønt og Out3_3V markert i blått på grafen. Vsignal genererer en firkant-puls på 5V, dette for å se at signalet er blitt konvertert på 3,3V utgangen, Out_3V. Tilsvarende simulering ble også gjort ved 3,3V til 5V. Også dette ga ønsket nivå-konvertering.

Videre ble også en 'design rule check' utført i Circuitmaker [40]. Denne baserer seg på predefinert og egendefinerte regler, som i dette tilfellet er justert i henhold til produksjonsspesifikasjonene til prototypelabben [36]. Design rule ligger som vedlegg til denne rapporten. Se vedlegg I.

3.2.9 Menneskelig grensesnitt

Logitech G29 ratt og pedaler

For styring av kjøretøyet ble Logitech G29 ratt og pedaler valgt. Dette var for å gi følelsen av å faktisk kjøre bilen. Logitech G29 kommuniserer over USB, noe som gjør det enkelt å koble til en Raspberry Pi og har driverstøtte for Linux [41].

Racingstol Playseat Challenge

Det er brukt en racingstol. Dette gir gode monteringspunkt for ratt og pedaler og følelen av å sitte i en bil. Valget falt på akkurat denne fordi den var sammenleggbar og dermed lettere å transportere [42].

3.2.10 Kamera og skjerm

Fordi prosjektet bygger på Raspberry Pi ble opprinnelig det offisielle Raspberry Pi kameraet (Camera Module V2) valgt [43]. For å minimere forsinkelsen fra kamera til skjerm benyttes teknikker som lav oppløsning og høy bildefrekvens. Dette gjør at FOV blir lav, og det blir vanskeligere enn nødvendig å styre bilen. Derfor ble kameraet byttet ut med 'Raspberry Pi High Quality Camera', som har feste for å montere vidvinkel linse [44]. I dette kameraet sitter det en Sony IMX477 sensor på 12,3 megapiksler. Montert på kameraet sitter en linse med 165° FOV. Raspberry Pi kommuniserer med kameraet over CSI-2 standarden. Tilkobling skjer med CSI kabel.

På pilot-siden brukes en standard PC skjerm, tilkoblet med HDMI kabel. Skjermen som benyttes er en 27" 'Benq GL2780' som støtter oppløsning opptil 1920x1080 (16:9 format). Video som sendes kan ikke sendes i 1920x1080 oppløsning, da hadde forsinkelsen blitt uakseptabelt høy. Om video som sendes har lavere oppløsning enn skjermen må video i utgangspunktet skaleres opp for å vises i fullskjerm. Oppskalering av video gir ekstra forsinkelse. Derfor er pilot-pi stilt inn til å gi ut et videosignal med en oppløsning på 720x480 piksler. Videoen som sendes har samme oppløsning for å matche dette. Ved å benytte samme oppløsning på skjerm og video fyller bildet hele skjermen fra topp til bunn uten at man trenger å skalere opp video.

3.2.11 Bil

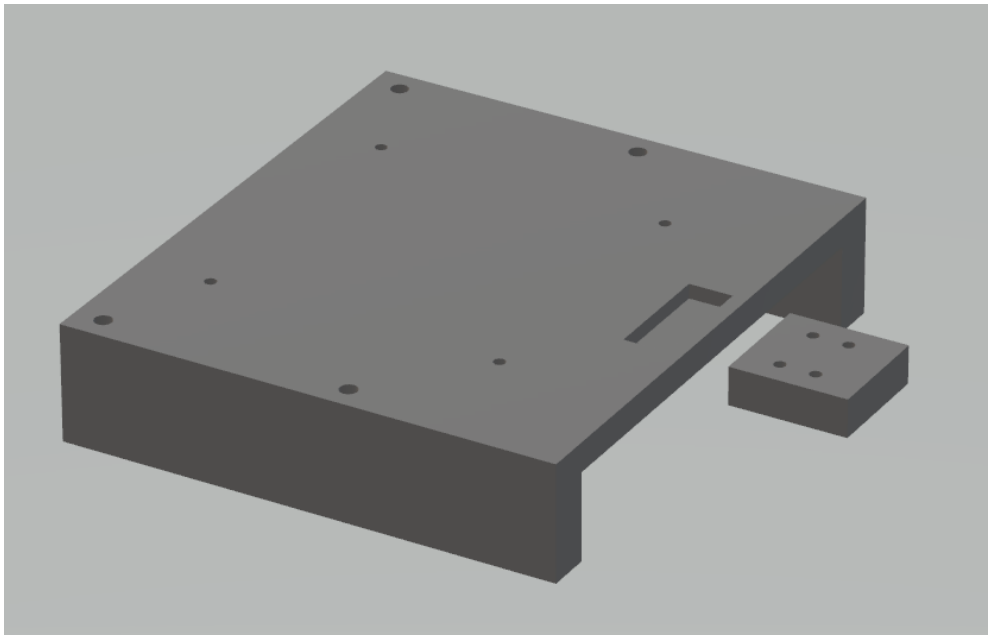
Figur 3.14: Bilen brukt under prosjektet [45].

Bilen som ble valgt er en 'Traxxas Slash 1/10 2WD Black RTR' og vises i figur 3.14. Denne ble valgt fordi den kun har tohjulstrekk og dermed er det færre bevegelige deler. Dette betyr færre deler som kan bli ødelagte under kjøring. Den er også ganske stor, noe som gjør det lett å få plass til å montere alle komponentene.

3.2.12 3D-print

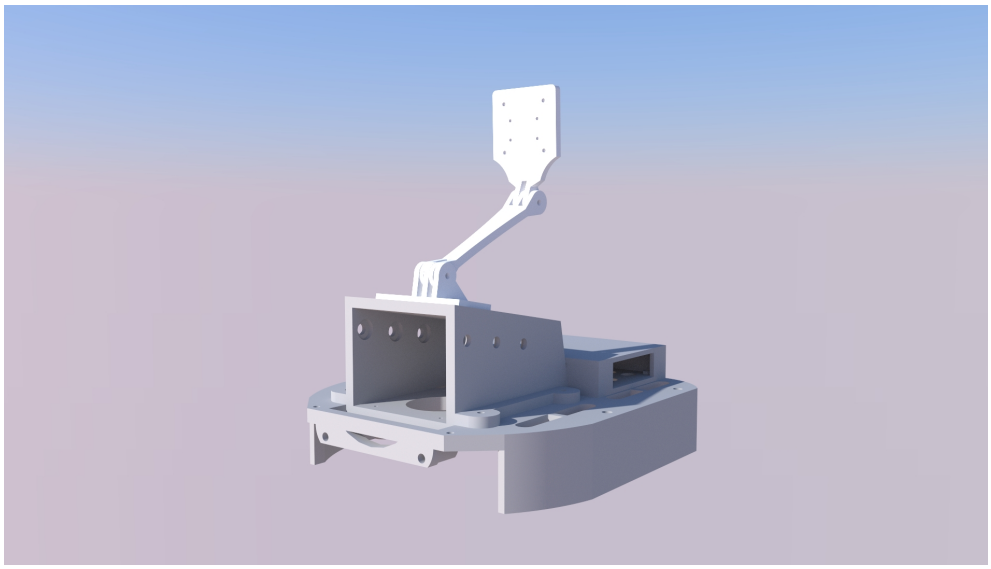
For montering av hardware på bilen falt valget på 3D-print. Dette gir en forholdsvis lett brakket som alt kan monteres på, men også beskyttelse dersom bilen krasjer. Det er også en billig løsning ettersom NTNU har 3D-printere tilgjengelig for studenter med kurs. Hovedfokuset ved 3D-print er å kunne feste alle komponentene med skruer for å forhindre bevegelse under kjøring. Dette var spesielt viktig for kameraet som trengte et solid monteringspunkt for minst mulig risting under kjøringen.

På bilen var det allerede 4 hull som kunne brukes som feste til brakketten. I første revisjon av 3D-print ble holderen som kom med 5G-HAT festet direkte. Dette er ikke en optimal løsning fordi den tar mye unødvendig plass og er laget av tynn plastikk med plastskruer. Kameraet ble montert fremme på brakketten for å gi god sikt. Første revisjon ble laget så simpel som mulig og skulle egentlig bare brukes til testing underveis i oppgaven.



Figur 3.15: 3D-print, revisjon 1.

Første revisjon av 3D-print vises i figur 3.15. Problemet med den denne var plassering av kameraet. Med kameraet montert i front på bilen var det vanskeligere enn nødvendig å styre og dårlig sikt rundt bilen.



Figur 3.16: 3D-print, revisjon 2.

I revisjon 2, som vises i figur 3.16, ble kameraet flytte opp og bak. Med mer av bilen i synsvinkelen til piloten skal det bli lettere å styre og få et overblikk over ting

rundt bilen. Holderen til 5G-HAT ble også fjernet og byttet ut med 3D-printede deler som er sterkere og mer plasseffektive. Koblingsbrett med Arduino og logikk-konverteringskrets ble byttet ut med egetdesignet kretskort, dermed er det laget monteringsmuligheter for dette. Det er også en vifte foran på bilen som skal kjøle ned Raspberry Pi og 5G-HAT.

Dekselet over 5G-HAT og Raspberry Pi beskytter delene, retter luftstrømmen på en effektiv måte og fungerer som monteringspunkt for antennene. Kretskortet har et lokk over seg som beskytter det og kameraarmen er justerbar for å kunne justere kameravinkel. Hele 3D-print er også formet mer som bilen for å se bedre ut, men også for å holde batterier og andre ting som ligger under bedre på plass. Dette designet fungerer mye bedre som en ferdig løsning, men lave toleranser mellom 3D-print og komponenter gjør den vanskelig å koble opp og ned.

3.3 Programvare

3.3.1 Rattavlesning

For å kontrollere kjøretøyet benyttes et Logitech G29 ratt. Dette rattet bruker USB-grensesnitt for input, med god støtte i Linux ved bruk av joystick API[46] .

Headerfil for avlesning av G29-ratt

For å kunne lese av dataen fra rattet benyttes C++ på Linux, nærmere bestemt ved bruk av en Raspberry Pi, igjennom USB-grensesnittet. I Linux skrives data fra USB-HID enheter til virtuelle filer, for å kunne handtere og bruke dataen[47]. Avlesing av disse virtuelle filene kan gjøres på flere måter, og i dette tilfellet benyttes C++. Avlesning er gjort ved hjelp av en egenutviklet headerfil, skrevet i C++ basert på Linux-dokumentasjon for avlesning av joystick-enheter.

Implementering av i kode headerfilen

For å implementere headerfilen i kode må en først og fremst legge headerfilen tilgjengelig for C++-koden som skal skrives. På Raspberry Pi
legges filen G29_wheel_input.h inn i header-mappen på Raspberry Pi.

Headerfilens oppbygning

Headerfilen er bygget opp av en klasse som tar seg av kommunikasjon med rattet med funksjoner som gjør nødvendige verdi-konverteringer. Denne klassens struktur knyttes til ett objekt som arver klassens struktur inne i koden hvor headerfilen er inkludert.

Includes
Standard Linux-biblioteker brukt i headerfilen. Dette er predefinerte headerfiler som gjør det mulig å bruke Linux-funksjonalitet i C++ [48].

```
/###Burde få inn ifdef for å sjekke om disse er definerte fra før, i
    koden header-filen brukes i./
#include <stdio.h> //
#include <stdlib.h> //calloc for å lage 0-arrays.
#include <fcntl.h> //For å sette avlesningsmodus
#include <unistd.h> //read() for avlesning av filer
#include <sys/ioctl.h> // IO - kontroll input/output control
#include <linux/joystick.h> //MÅ LASTES NED PÅ PI F R BRUK
#include <iostream> //cout for outputs til terminal
#include <math.h>//pow for å finne kvadratrot.
#include <typeinfo>
#include<unistd.h>//usleep delay.
```

Kodeliste 3.1: Rattavlesningskode - inkluderte biblioteker.

Kodeliste 3.1 viser de inkluderte GNU/Linux-bibliotekene brukt i headerfilen. Se eget vedlegg for utdypende forklaring av hvert enkelt bibliotek.

Definerte variabler

Hele rattdata-koden er strukturert i en klasse. Alle definisjoner, utenom selve defineringen av klassen er inne i klassen.

Public variabler og klasse-deklarasjon

```
class JoystickInput
{
public: / Public variabler. /
        #define JOY_DEV "/dev/input/js0" //Filbanen peker på default
            Linux USB-inndata for joysticks. jsx.
        / RATT-CODES definert /
        #define WHEEL  0                        //evdev code for
            rattrotasjon
        #define CLUTCH  1                       //evdev code for
            clutchpedal
        #define ACCELERATOR 2           //evdev code for gasspedal/
            akseleratorpedal
        #define BRAKE 3                         //evdev code for
            bremsepedal
        #define FIRST 5                         //evdev code for ukjent
             verdi som kommer ved oppstart.
        #define MINUS_BTN_CODE 20       //evdev code for knappetrykk
            minus.
        #define PLUSS_BTN_CODE 19       //evdev code for knappetrykk
            pluss.
        int wheel_bit_resolution = 16; //Oppløsning på ratt og pedaltrå
            kk
        int wheel_deg_max = 860;        //860 graders rotasjon.
            Logitech oppgir 900grader.
        int wheel_range = 180;          //180 graders utslag default.
            Hvor mye av rattets rotasjonsområde vi velger å bruker
```

Kodeliste 3.2: Rattavlesningskode - Public variabler

Kodeliste 3.2 viser deklarasjon av public variabler, samt definisjon av selve klassen. Public variabler er tilgjengelige også utenfor klassen og kan leses og skrives til. Dette er for å kunne redefinere de forskjellige variablene i c++-koden som bruker headerfilen om det skulle være behov, uten å måtte endre header-filen.

Private variabler

Private variabler brukt i klassen.

```
private:
// Diverse variabler


unsigned int microsecond = 1000000; //ett sekund i mirosekunder. brukes
     i usleep.
int oldRange,newRange; //deklarerer oldrange og newRange til senere
    bruk i map-funksjonen.
int m_value,m_code; // Deklarerer  variabler for utlesning av data.
    Return-verdi fra funksjonene eventCode og eventValue.
// Diverse END
```

```
//Wheel values
    :

 int m_wheel_max = (pow(2, 16) - 1) / 2;//fordi rattet går fra -32xxx
    til 32xxx
 int m_wheel_center = 0;// Gammel type
 int m_wheel_inc_deg = (m_wheel_max 2 / wheel_deg_max); //kalkulerer
    antall grader rotasjon pr int-inkrement
 int m_wheel_range_int = m_wheel_inc_deg    wheel_range;          //
    Kalkulerer range i skalert til returnverdi fra ratt.
 / finner maksimalutslag i vær retning basert på wheel_range_int /
 int m_Newwheel_min = m_wheel_center - m_wheel_range_int;
 int m_Newwheel_max = m_wheel_center + m_wheel_range_int; //Setter nye
    max og min med default 360 graders total rotasjon
//Wheel END



//Variabler som brukes i avlesning av fil. Satt opp slik som i
    linuxdokumentasjonen til joystick.h              /
int joy_fd,    axis = NULL, num_of_axis = 0, num_of_buttons = 0, x; //
    Deklarerer flere variabler som int. og axis som en pointer-int
char  button = NULL, name_of_joystick[80]; //Deklaerer char-pointer for
    knapper og joystick-navn.
struct js_event js; // lager en struckt med event. js_event arver
    struckt-strukturen til js.
//Avlesningsvariabler END
```

Kodeliste 3.3: Rattavlesningskode - Private variabler

Kodeliste 3.3 viser de definisjon av alle private variabler bruk i headerfilen. Private variabler er i motsetning til public variabler, ikke tilgjengelige utenfor klassen. Dette er variabler som brukes og håndteres av funksjonene i klassen. Her ligger variabler som i utgangspunktet enten er statiske eller som kalkuleres utifra gitte verdier fra enten public-variabel, eller fra data returnert fra rattet.

Funksjoner inne i klassen JoystickInput

Inne i klassen er både public og private funksjoner deklarert, med lik tilgjengelighet slik som variablene.

```
public:

int setup(void) //Setup joystick/ratt
void setWheelRange(int deg_angel) //For å redefinere ratt-utslag
void readEvent(void) //Les av ny data.
int eventValue() //returnerer value knyttet til kode
int eventCode() //returnerer kode knyttet til value.
```

Public-funksjonene beskrives mer i detalj i sitt eget avsnitt. Alle funksjonene over er klassifisert som public, og derfor tilgjengelig utenfor klassen. Det gjør at funksjonene kan kalles ved bruk av klassen.

```
Private:

int map(int oldMax, int oldMin, int newMax,int newMin) //Generell re-
    map funksjon
void Wheel_rotation(void) // Håndtering og remapping av ratt-rotasjon
    til grader.
Pedal_Push8bit(void) //Håndtering og remapping av pedal-data til 8-bit:
     0-255
```

Kodeliste 3.4: Rattavlesningskode - Private funksjoner.

Kodeliste 3.4 viser alle de definerte private funksjonene brukt i klassen. Private-funksjonene blir også beskrevet mer i detalj i eget avsnitt. Alle funksjonene over er klassifisert som private, og er ikke tilgjengelig utenfor klassen. Dette er funksjoner som kun brukes internt i klassen.

Bruk og beskrivelse av funksjoner

I denne delen av kapittelet vil klassens private - og public-funksjoner bli forklart i detalj. Public funksjoner

Public funksjoner, altså funksjoner som er tilgjengelig utenfor klassen. Disse kan kalles på når objektet er blitt definert.

```
/ Definerer objekt av klassen JoystickInput /
JoystickInput G29; // Definerer objektet G29.
```

Kodeliste 3.5: Rattavlesningskode - Eksempel på definisjon av objekt.

I kodeliste 3.5 deklareres objektet og kan nå kalle på de forskjellige public-funksjonene inne i klassen JoystickInput.

setup(void)

```
/ Kaller på setup-funksjonen definert inne i klassen /
G29.setup(); //Setter opp rattet. Denne kjøres engang.
```

Kodeliste 3.6: Rattavlesningskode - kaller setup().

I kodeliste 3.6 kalles setup-funksjonen og kjøres. Denne funksjonen definerer diverse variabler og åpner den virtuelle filen som returnerer inndata fra rattet koblet til med USB.

```
int setup(void)
{
        //Vi definerer joy_fd som open (js0,read only) og sjekker om
            den er -1 for å se om fila ble åpnet eller ikke.
        if ((joy_fd = open(JOY_DEV, O_RDONLY)) == -1) //JOY_DEV er
            filbane til js0 i linux virituelle filsystem. O_RDONLY=
            read only. hvis return er -1, ble ikke fila åpnet.
```

```
        {
                //std::cout<<("Couldn't open joystick. Check USB
                    connection")<<std::endl;
                std::cout<<("Kunne ikke åpne joystick-datafil. Sjekk
                    USB-tilkoblingen!")<<std::endl;
                return -1;
        }
        //Ved suksess av åpning av fil:
        ioctl(joy_fd, JSIOCGAXES, &num_of_axis);
                        //joy_fd er filen js0 åpnet.. JSIOCGAXES
            returnerer antall akser inn i num_of_axis
        ioctl(joy_fd, JSIOCGBUTTONS, &num_of_buttons);
                //JSIOCGBUTTONS returnerer antall buttons inn i
            num_of_buttons.
        ioctl(joy_fd, JSIOCGNAME(80), &name_of_joystick);
                //JSIOGNAME returerner identifier string,
            enhetsinformasjon/navn på enhet
        axis = (int *) calloc(num_of_axis, sizeof(int));
                //calloc (size_t num, size_t size); Allokerer
            minneplass til arrayet axis
        button = (char *) calloc(num_of_buttons, sizeof(char));   //
            calloc:Allocates a block of memory for an array of num
            elements, each of them size bytes long, and initializes all
             its bits to zero.

        / Skriving til terminalvinduet. /
        //std::cout <<"Joystick detected:"<<name_of_joystick<<std::endl
            ;
        std::cout <<"Joystick detected:"<<name_of_joystick<<std::endl;
              //skriver ut til terminal: navn på joystick.
        //std::cout<<"Axis:"<<num_of_axis<<std::endl;
        std::cout<<"Axis:"<<num_of_axis<<std::endl;
                                            //skriver ut til terminal:
            Antall akser.
        //std::cout<<"Buttons:"<<num_of_buttons<<std::endl;
        std::cout<<"Buttons:"<<num_of_buttons<<std::endl;
                                  //skriver ut til terminal: Antall
            knapper.
        fcntl(joy_fd, F_SETFL, O_NONBLOCK);      / Setter avlesningen
            til ikke blokkerende modus /
        return 1; //Setup returnerer 1 ved suksess

}
```

Kodeliste 3.7: Rattavlesningskode - Setup-funksjonen.

Kodeliste 3.7 viser hvordan setupfunksjonen er oppbygd. Funksjonen definerer joy_fd som open(JOY_DEV,ORDONLY), dette vil lese av filbanen JOY_DEV som peker på USB-enheten, som i dette tilfellet er Logitech-rattet som ønskes avlest, videre sjekkes det hva denne returnerer.
Hvis denne returnerer -1 betyr det at filen ikke kunne åpnes og en output på skjermen ved hjelp av cout gir beskjed om dette. Hvis den returnerer noe annet enn -1 betyr det at filen ble åpnet og er klar til videre håndtering av dataen som nå er

lastet inn i joy_fd.

Ioctl tar tre argumenter. Argument en, joy_fd er filen som skal håndteres, argument to, JSIOCGAXES, JSIOCGBUTTONS eller JSIOCGNAME gir informasjon om hvilken variabel fra filen en ønsker returnert.
Det tredje argumentet, num_of_axis, num_of_buttons eller name_of_joystick er variabelen en ønsker at informasjonen skal lastes inn i. Dette gjøres for hver enkelt variabel, først num_axis, så num_buttons og til sist num_of_buttons. Videre brukes denne ny informasjonen til å lage to arrays med lengden av antall akser og knapper. Akser som array av ints, og button som en array of chars.

Videre brukes disse variablene til å lage to arrays ved hjelp av calloc. Calloc(lengde på arrayet, størrelse på hver plassering). Calloc genererer en array med gitt lengde og størrelse på hver enkelt plassering i arrayet, og det reserveres derfor plass i minnet til senere bruk. Videre blir alle disse verdiene satt til null[49]. Num_axis brukes til lengde på arrayet og sizeof(int) brukes til å bestemme hvor stor hvert enkelt element i listen skal være. Det samme gjøres med num_buttons, men i dette tilfellet holder det med elementstørrelse på sizeof(char).

Nest sist i setup-funksjonen printes informasjon om hvilken joystick som er koblet til og hvor mange akser og knapper denne har. Dette printes til skjermen med funksjonen cout, som er standard output-funksjon innebygget i C++ [50].
Avslutningsvis i setup-funksjonen settes avlesning av joy_fd til «NON-BLOCKING mode» ved bruk av fcntl, som vil si at avlesningsmodusen til read-funksjonen som brukes til å lese inn data ikke blokkerer videre kjøring om det ikke er data [51].

readEvent(void)

```
/ Kaller på read-funksjonen definert inne i klassen /
G29.readEvent (); //Setter opp rattet. Denne kjøres engang.
```

Kodeliste 3.8: Rattavlesningskode - kaller på readEvent-funksjonen.

I kodeliste 3.8 kalles readevent-funksjonen og kjøres. Denne funksjonen leser inn nyeste data fra rattet og gjør denne dataen tilgjengelig for videre håndtering. Funksjonen har også en forsinkelse for å bremse ned hvor ofte en kan lese av data, og dermed spare CPU-kraft. Denne funksjonen legges inn i hoved-løkken av koden hvor headerfilen blir brukt, dette er for å kontinuerlig avlese data fra rattet og få nyeste data inn.

```
void readEvent(void)
{
        //Pollingrate på USB-HID er 1000Hz. Nyquist-samplingsrate blir
            da 2000Hz, eller hvert 0.5ms.
        usleep(0.0005   microsecond);//0.5ms delay for å avlaste CPU.
            Kan byttes ut med millis-timer for å unngå blocking, men
            ikke behov.
        read(joy_fd, &js, sizeof(struct js_event)); //leser in fra
            joy_fd inn i structen js som har arvet struktur fra
            js_event fra joystick.h.
        //std::cout<<("  \r")<<std::endl; //Setter print pointer til
            start igjen. skriver over printen fra forrige.
        fflush(stdout); //Flusher buffer for cout.

}
```

Kodeliste 3.9: Rattavlesningskode - readEvent-funksjonen.

Funksjonen beskrevet i kodeliste 3.9 begynner med en kort forsinkelse på 0.5 glsms ved å bruke den innebygde Linux-funksjonen usleep fra unistd-biblioteket. Etter denne korte pausen i koden leses så joy_fd som tidligere ble definert og åpnet av setup-funksjonen av ved å bruke funksjonen read() fra unistd-biblioteket. Read er en innebygget Linux-funksjon og brukes for å lese av filer. Funksjonen tar tre argumenter, hvor første argument er filen, bufferen-variabelen en laster dataene inn i og siste argument er bufferstørrelse på variabelen dataene lastes inn i.

Dataene blir i dette tilfellet lastet inn i js som tidligere nevnt i variabel-avsnittet er en struct som har arvet sin struktur av js_event fra joystick.h biblioteket. Js-structen har nå den nyeste dataen fra rattet lastet inn i seg, med strukturen som nevnt i joystick.h-delen innledningsvis.

eventValue()

```
/ Kaller på eventvalue-funksjonen definert inne i klassen /
value=G29. eventValue (); //returnerer verdi fra enten rattrotasjon
    eller pedaltråkk
```

Kodeliste 3.10: Rattavlesningskode - Kaller på eventValue-funksjonen.

I kodeliste 3.10 kalles eventvalue-funksjonen. Denne funksjonen returnerer verdi-informasjon om siste registrerte hendelse *event*. Den returnerte verdien lastes så inn i en variabel, i dette tilfellet value.

```
int eventValue()
{
        //Kanskje laste inn i m_value her, istedenfor i alle under.
            Ingen utslag på kjøring, men færre linjer kode.
        switch (js.type & ~JS_EVENT_INIT)
        {
        case JS_EVENT_AXIS: //Ved case aksedata: dynamisk data. feks
            ratt eller pedal.
                m_value = js.value;//laster js.value inn i privat
                    variabel m_value, som er tilgjengelig for hele
                    klassen.

                if (js.number == FIRST)//Håndterer utventet verdi FIRST
                    =5.
                {
                        //do nothing...
                }
                //js.number er rattdata.
                if (js.number == WHEEL)
                {
                        Wheel_rotation();
                        //m_value konverteres ned til 8bit. 0- 180.
                        //std::cout<<"[WHEEL]:"<<std::endl;
                }
                //js.number er clutchpedal-data
                if (js.number == CLUTCH)
                {
                        Pedal_Push8bit();
                        //m_value konverteres ned til 8bit. 0- 255.
                        //std::cout<<"[CLUTCH]:"<<std::endl;
                }
                //js.number er Gasspedal-data
                if (js.number == ACCELERATOR)
                {
                        Pedal_Push8bit(); //remapper verdi
                        //m_value konverteres ned til 8bit. 0- 255.
                        //std::cout<<"[ACCELERATOR]:"<<std::endl;
                }
                //js.number er bremsepedal-data
                if (js.number == BRAKE)
                {
                        Pedal_Push8bit();
```

```
                        //std::cout<<"[BRAKE]:"<<std::endl;
                        //m_value konverteres ned til 8bit. 0- 255.
                }
                break; //Avslutter case.
        case JS_EVENT_BUTTON: //ved case knappetrykk.
                //Hvis js.number er ett av de to definerte
                    knappetrykkene.
                if ((js.number==MINUS_BTN_CODE) || (js.number==
                    PLUSS_BTN_CODE))
                {
                        m_value = js.value; //laster verdi inn i
                            m_value.
                }
                else{m_value = m_value;} //ignorer verdi fra knapper
                    som ikke er definert.
                break; //Avslutter case.
        }
        return m_value;  //returnerer value.
}
```

Kodeliste 3.11: Rattavlesningskode - eventValue-funksjonen.

Verdiene filtreres inne i funksjonen og skiller mellom data fra rattrotasjon og pedaltråkk. Returnerer en 8-bit unsigned int, 0-255 for pedaltråkk og 8-bit unsigned int, 0-180 grader for rattrotasjon.
Funksjonen eventValue som er beskrevet i kodeliste 3.11 skiller mellom knappetrykk, pedaltråkk og rattrotasjon. I første omgang lastes js.value inn i en privat variabel, som senere blir manipulert basert på hvilken type input som avleses.

Det brukes en switch-case for å skille mellom dynamisk akse-data, derunder pedaltråkk og hjulrotasjon, eller boolsk knappetrykk. Variabelen som brukes i switch er js.type og JS_EVENT_INIT.
Sistnevnte brukes i utgangspunktet til å skille mellom ekte og syntetiske events, men i dette tilfellet er det ikke behov for å skille mellom og derfor brukes en &, logisk-and og en ̃bitwise kompliment, i henhold til Linux-dokumentasjonen [46]. Dette vil gjøre at kun js.type brukes i switch-case, samtidig som en ignorerer å skille mellom syntetisk og ekte events.

Js.type returnerer to mulige verdier, enten JS_EVENT_BUTTON som er definert i joystick.h som 0x01, eller JS_EVENT_AXIS som er derfinert som 0x02. Derfor brukes JS_EVENT_BUTTON og JS_EVENT_AXIS som case, for å skille mellom type data. Hvis case JS_EVENT_AXIS utløses betyr det at dataen som kan leses fra js.value er av typen signed 16bit int.

Videre brukes en if-setning med js.number til å bestemme hvilken type akse-input som skal håndteres. Derunder Wheel, Clutch, Accelerator eller brake. Alle disse er definert som public variabler,tallverdier som er mappet til de forskjellige funksjonalitetene på rattet.

Eksempler på dette er js.number 0 for rattrotasjon, 2 for gasspedal også videre. I tilfellet if(js.number ==Wheel) kjøres den private funksjonen Wheel_rotation, som forklares i detalj under "Wheel_rotation()".

Kort forklart tar denne inn den 16-bit signed-variabelen m_value å mapper den om fra området -32767 til 32767, over til området 0 til 180.
I de neste tilfellene av den nøstede if-setningen håndterer pedaltråkk, altså tilfeller der dataen er fra pedalene. Også disse skal mappes om og legges til ett nytt område. Dette gjøres med funksjonen Pedal_Push8Bit(), som også skal forklares i mer detalj senere i kapittelet.
Kortfattet forklart mapper funksjonen om verdiene i m_value å mapper den om fra området -32767 til 32767, over til området 0 til 255.

Switch-caset har en siste case, hvor knappetrykk håndteres. I dette caset sjekkes det først om js.number er innenfor de predefinerte knappene; minusknapp og plussknapp. Dette med en enkel if-setning med en or-operator ||. Hvis if-setningen blir utløst lastes js.value direkte inn i m_value og blir returnert av funksjonen eventValue(). I tilfellet der knappen ikke er en av de definerte, filtreres denne bort i else-setningen ved å sette m_value lik seg selv, og dermed forrige verdi, slik at ingen ny data-return fra eventValue() står uendret.

int eventCode()

```
/ Kaller på eventCode-funksjonen definert inne i klassen /
code=G29. eventCode (); //returnerer verdi fra enten rattrotasjon eller
    pedaltråkk
```

Kodeliste 3.12: Rattavlesningskode - Kaller på eventCode-funksjonen.

I denne kodelisten, kodeliste 3.12 kalles funksjonen eventCode(). Denne funksjonen returnerer en int som identifiserer hvilken knapp eller akse-verdien tilhører.

```
int eventCode()
{
        switch (js.type & ~JS_EVENT_INIT)
        {
        case JS_EVENT_AXIS: //Ved dynamisk data. feks ratt eller pedal.
            s
                m_code = js.number;

                return m_code; //returnerer code.
                break; //Avslutter case.

        case JS_EVENT_BUTTON: //Ved Boolsk data. Knappetrykk
        if ((js.number==MINUS_BTN_CODE) || (js.number==PLUSS_BTN_CODE))
        {
                m_code = js.number; //Laster inn knappeidentifikator.
                    Enten MINUS_BTN_CODE) || (js.number==PLUSS_BTN_CODE
        }
```

```
        else{m_code = m_code;}//ignorer code fra knapper som ikke er
            definert.
                return m_code; //returnerer code.
                break; //Avslutter case.
        }
}
```

Kodeliste 3.13: Rattavlesningskode - eventValue-funksjonen.

Inne i funksjonen eventCode() som er beskrevet i kodeliste 3.13 kjøres først en switch-case, likt som i eventValue(). Dette for å identifisere om eventet er ett knappetrykk eller en akse-verdi.
Ved case JS_EVENT_AXIS, altså at koden er knyttet til en akse-verdi lastes js.number inn i den private variablen m_code, videre returneres m_code.
Ved caset JS_EVENT_BUTTON, altså at koden er knyttet til en knappeverdi kjøres ett filter. Dette filteret sjekker om knappetrykket er en av de definerte knappetrykk.

Filteret realiseres ved hjelp av en if-setning som sjekker om knappetrykket er den definerte minus-knappen eller den definerte pluss-knappen. Hvis knappetrykket ikke er en av de to definerte knappene, settes m_code til den foregående verdien, slik at ingen endring på utgangen (return) blir registrert og udefinerte knappetrykk blir kastet.

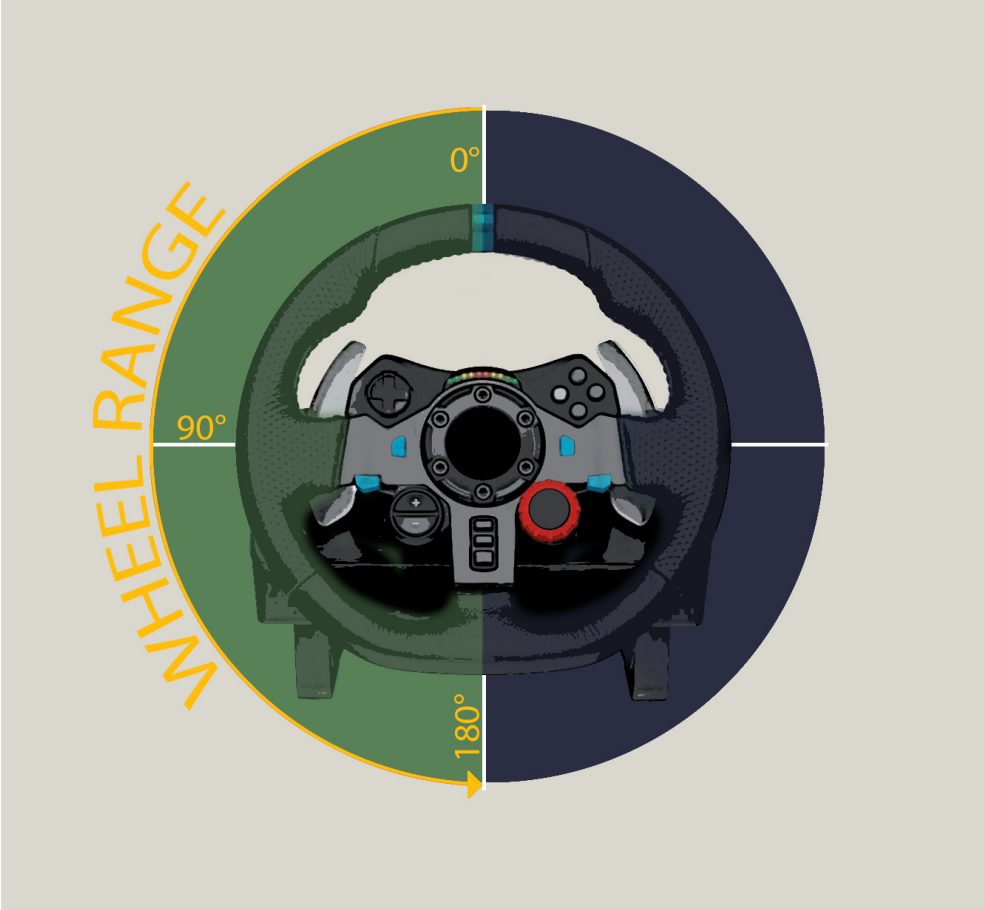
setWheelRange(int deg_angel)

```
/ Kaller på setWheelRange-funksjonen definert inne i klassen /
G29. setWheelRange (180); //Setter rattutslaget i antall grader hver
    vei fra midten til 180grader.
```

Kodeliste 3.14: Rattavlesningskode - Kaller på setWheelRange-funksjonen.

I kodeliste 3.14 over, kalles funksjonen setWheelRange som er definert inne i klassen. Denne funksjonen setter den private variabelen wheel_range.
Wheel_range-variabelen bestemmer hvor mye av rattutslaget en skal bruke fra midtstilt posisjon.

I tilfellet med bruk av Logitech G29 er det totale rattutslaget 900 grader, det vil igjen si 450 grader fra midstilt posisjon [41]. Her er det blitt valgt å bruke kun 180 grader, som betyr 180 grader utslag til hver side. setWheelRange blir satt til 180 grader.



Figur 3.17: Illustrasjon av rattutslaget.

Figur 3.17 illustrerer rattutslaget som settes med funksjonen setWheelRange.

```
void setWheelRange(int deg_angel)
        {
                if (deg_angel <= wheel_deg_max)
                {
                        wheel_range = deg_angel;
                        std::cout <<"New range set:"<<wheel_range<<std
                            ::endl;
                }
                else
                {
                        std::cout << "Please select a valid range...
                            (0-450). range set to default 180deg."<<std
                            ::endl;
                        wheel_range = 180;
                }
        }
```

Kodeliste 3.15: Rattavlesningskode - setWheelRange-funksjonen.

Funksjonen setWheelRange beskrevet i kodeliste 3.15 tar ett argument, deg_angel, som er antall grader utslag til en side. Videre kontrolleres det at deg_angel ikke er større enn det maksimale rattutslaget på 450 grader.
Hvis gitt argument er innenfor spesifikasjonene vil ny wheel_range bli satt og cout skriver i terminalvinduet at ny grense er satt. Hvis verdien derimot er utenfor spesifikasjonene, vil verdien wheel_range settes til default på 180grader og en feilmelding skrives til terminalvinduet.

Private funksjoner
De privatefunksjonene som er blitt brukt i funksjonene forklart tidligere i kapittelet, forklart mer i detalj.

void Wheel_rotation(void)
Den private funksjonen wheel_rotation(void) brukes for å mappe om verdier fra 16-bit unsigned int, til grader 0 til 180. Videre settes det også opp begrensing for hvor mye av rattutslaget skal brukes.

```
if (js.number == Wheel)
{
        Wheel_rotation();
        std::cout<<"[Wheel]:"<<std::endl;
}
```

Kodeliste 3.16: Rattavlesningskode - Kaller på wheel_rotation-funksjonen.

I eventValue()-funksjonen som ble forklart tidligere i kapittelet kalles funksjonen Wheel_rotation() hvis js.number svarer til Wheel, slik som vist over i kodeliste 3.16.

```
void Wheel_rotation(void)
        {
                if (m_value < m_Newwheel_min) // m_Newwheel_min er en
                    negativ verdi.
                {
                        m_value = m_Newwheel_min;
                        std::cout<<"[MIN]"<<std::endl;
                }
                if (m_value > m_Newwheel_max)
                {
                        m_value = m_Newwheel_max;
                        std::cout<<"[MAX]"<<std::endl;
                }
                map(m_Newwheel_max, m_Newwheel_min, 180, 0); //Mapper
                    om verdiene til grader. 0 til 180 grader.
        }
```

Kodeliste 3.17: Rattavlesningskode - wheel_rotation-funksjonen.

Inne i Wheel_rotation-funksjonen som er beskrevet i kodeliste 3.17, sjekkes det om avlest value er innen for de satt minimums grensene og maksimumsgrensene som er definert på bakgrunn av hvor stort rattutslag som er definert. Hvis m_value er under minimumsverdien m_Newwheel_min, settes m_value til denne verdien, slik at alle verdier under minimumsgrensen ignoreres.
Det samme gjelder for verdier over maksverdien m_Newwheel_max settes m_value til maksverdi, slik at alle verdier over grensen blir ignorert. Dette er fortsatt 16bit unsigned verdier, men området går ikke lengre fra -32767 til 32757, men er redefinert fra m_NEwwheel_min til m_Newwheel_max.

Disse maks og min-verdiene er utledet matematisk og definert under de private variablene. Disse min og maks-verdiene kan også redefineres ved bruk av setWheelRange-funksjonen nevnt tidligere i kapittelet. Videre skrives det også melding til terminalen hvis rotasjonen har nådd min og maksverdiene.
Til slutt i funksjonen kalles map-funksjonen som i dette tilfellet mapper om verdiene fra maks og minverdiene og til 0-180.
Dette gjør at det definerte området rattetutslaget er satt til, kan styre hele utslaget på servomotoren som styrer fremhjulene på bilen. map-funksjonen er også forklart i detalj senere i dette del-kapittelet.

void Pedal_Push8Bit(void)
Pedal_Push8Bit() er også en privat funksjon brukt for remapping av verdier fra pedelene.

```
if (js.number == Accelerator)
        {
            Pedal_Push8bit(); //remapper verdi
            std::cout<<"[Accelerator]:"<<std::endl;
        }
```

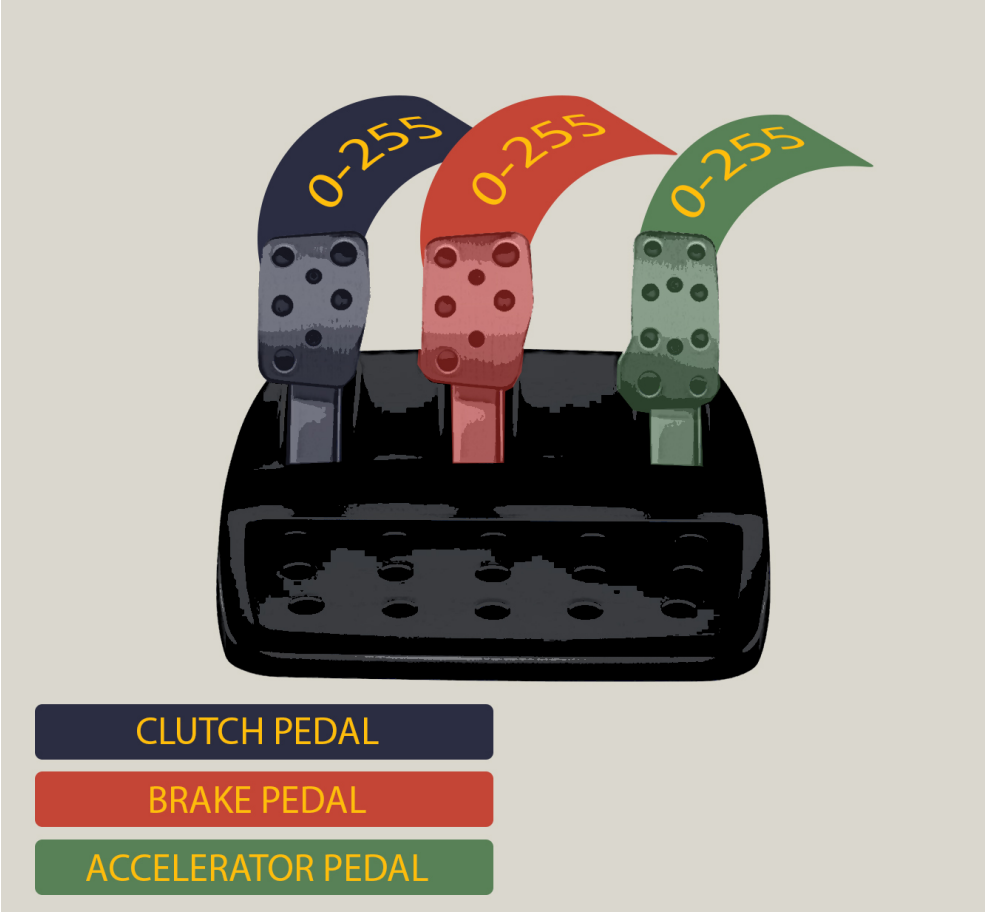
Kodeliste 3.18: Rattavlesningskode - Kaller på Pedal_push8bit-funksjonen.

Funksjonen Pedal_Push8Bit() blir kalt på hvis js.number svarer til en av de tre predefinerte pedalverdiene. Altså Accelerator, Clutch eller Brake, som vist i kodesnippen over viser tilfellet der js.number svarer til gasspedalen som i koden er definert som Accelerator.

```
void Pedal_Push8bit(void)       //Changing range of pedal press. er til
    remapping av pedaltråkk
       {
               map(-32767, 32767, 255, 0); //Mapper om data fra 16-bit
                    signed til 8-bit unsigned.
       }
```

Kodeliste 3.19: Rattavlesningskode - Pedal_push8bit-funksjonen.

Pedal_Push8Bit()-funksjonen beskrevet i kodeliste 3.19, kaller på map-funksjonen og definerer ny range for m_value. map endrer m_value fra å bruke området -32767 til 32767 over til å bruke 8-bit unsigned størrelse på dataen med området 0 til 255. Utover å kalle på map-funksjonen er det ikke mer som skjer inne i funksjonen.



Figur 3.18: Illustrasjon av pedalutslag.

Figur 3.18 illustrerer pedalutslaget etter at Pedal_push8Bit()-funksjonen har mappet om verdiene til 0-255.

int map()
Funksjonen map brukes til å remappe verdier fra ett område til ett annet. I dette tilfellet vil det si at området skaleres ned fra -32767 - 32767, ned til 0 - 255.

```
int map(int oldMax, int oldMin, int newMax,int newMin)
        {
                //Remapping av verdier:
                oldRange = (oldMax - oldMin);
                newRange = (newMax - newMin);
                m_value = (((m_value - oldMin)   newRange / oldRange) +
                     newMin);
                return m_value;
        }
```

Kodeliste 3.20: Rattavlesningskode - map-funksjonen.

Inne i funksjonen kartlegges først det opprinnelige området ved å ta differansen mellom maksimal og minimal verdi. Videre opprettes et nytt område, newRange som tar differansen mellom de nye grensene som blir definert. I tilfellet for ratt 0 til 180 og i tilfelle for pedaldata 0-255. Videre kalkuleres så ny verdi basert på disse.

$$oldRange = oldMax - oldMin$$

$$newRange = newMax - newMin$$

$$newValue = (oldValue - oldMin) * \frac{newRange}{oldRange} + newMin$$

I formelen over brukes newValue og oldValue for å tydeliggjøre variablene. I kode brukes kun m_value både for oldValue og newValue, da m_value blir avledet fra seg selv.
I kalkulasjonen av newValue beregnes først differansen mellom oldValue og oldMin, før denne multipliseres med forholdstallet mellom newRange og oldRange. Deretter legges dette sammen med den nye minimumsverdien. newValue vil da altså alltid holde seg innenfor de nye definerte grensene.

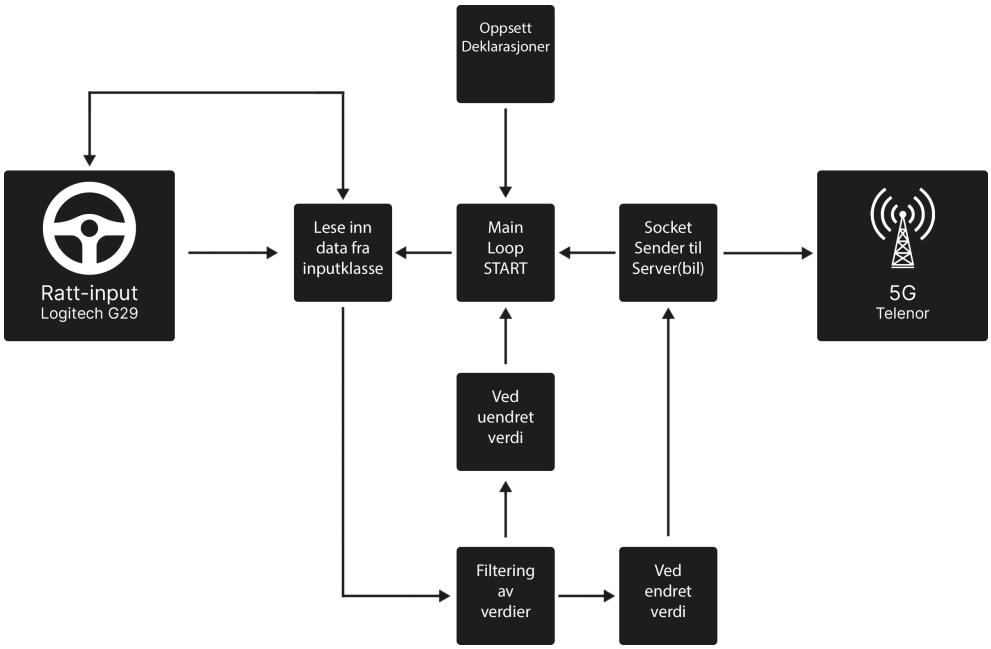
Til slutt returneres så m_value. Return fra denne funksjonen blir for ordensskyld ikke brukt. Dette fordi m_value er definert inne i klassen, som gjør at den er tilgjengelig for alle funksjonene og derfor redefinert i hele koden hvis en endrer m_value inne i en av funksjonene i klassen.

3.3.2 Nettverkskode for styresignal

Oppgaven til nettverksprogrammet er å avlese verdier fra brukerens input, altså ratt/pedal, og sende denne informasjonen videre til tjeneren som igjen sender dette serielt til motorkontrolleren ved bruk av SPI. Programmet er skrevet i c/c++ og det baserer seg på Linux sine innebygde funksjoner for nettverksprogrammering. Dermed er nettverkskoden ikke avhengig av eksterne biblioteker. Denne løsningen er basert på lærings-videoen: [52].
Github lenke klient: [53] fra læringsvideo.
Github lenke tjener: [54] fra læringsvideo.

Klient

Klienten er en løsning som baserer seg på bruk av standardbibliotek i Linux. I oppsettet defineres det først hvordan oppkoblingen skal skje.
Oppgaven baserer seg på TCP/IPv4. TCP og IPv4 bruker navnene SOCK_STREAM for TCP og AF_NET for IPv4. Deretter opprettes et objekt for bruk av input-klassen. Klienten kobler seg opp til tjeneren, og går inn i en uendelig løkke der den vil lese av fersk data fra rattet, og sjekke om disse er forandret fra forrige avlesning i filteret. Om det er nye verdier sendes disse til tjeneren. Programmet vil deretter begynne denne prosessen på nytt. Figur 3.19 viser klient-kodens arkitektur.



Figur 3.19: Diagram Klient-arkitektur.

Oppsett deklarasjoner

```
44: int main(int argc, char  argv[])
45: {
46:       char addr[20] = "78.158.241.23";
47:       int kom;
48:       int PORT = 54000;
49:       int data[2];
50:       int sock = 0;
```

Kodeliste 3.21: Klientkode - oppsetts-deklarasjoner a

Del a, kodeliste 3.21:
Linje 46: Her settes IPv4 adressen den skal koble opp til. Som standard brukes 78.158.241.23 som er IP-adressen tilhørende SIM-kortet på tjeneren. Denne adressen kan bli endret selv ved bruk av argumentet -i etterfulgt av ønsket adresse.
Linje 49: Allokerer en int, som vil lagre hvilke port nummer som blir brukt, standard blir den satt til 54000. Denne porten ligger utenfor det 'reserverte' området for TCP/IP porter: 0-1023 [55]. det er mulig å spesifisere en annen port ved å gi programmet argument -p etterfulgt av ønsket nummer.
Linje 49: Allokerer en int vektor med to elementer. Dette vil være den datavektoren som verdier fra ratt/pedaler blir lastet inn i og sent til tjeneren.
Linje 50: Oppretter selve socket variabelen. Sockets i Linux lagres i en 32-bit integer.

```
55: while ((kom = getopt(argc, argv, "i:p:")) != -1)
    {
57:   switch (kom)
      {
59:     case 'i':
60:         strcpy(addr, optarg);
61:         std::cout << "[Klient] Ip adresse gitt: " << addr << endl;
62:         break;
63:       case 'p':
64:         PORT = atoi(optarg);
65:         std::cout <<"[Klient] port gitt: " << PORT << endl;
66:         break;
67:       default:
68:         std::cout <<"[Klient] ingen argumenter port: " << PORT << "
    ip: " << addr << endl;
        }
 }
```

Kodeliste 3.22: Klientkode - oppsetts-deklarasjoner b

Del b, kodeliste 3.22:
Denne koden snuttens funksjon er å lese av argumentene som blir gitt oppstart. eksempel: *./main -p 100 -i 192.168.0.50*
Her vil mål ip adressen bli satt til 192.168.0.50 og port vil bli satt til 100
Linje 55: En løkke blir opprettet som går igjennom switch casen nedenfor, så lenge argumentet er sant. getopt funksjonen blir brukt denne funksjonen tar inn argc, argv fra main og bruker -i og -p som mulige argumenter [56].
Linje 59-62: Om argumentet -i er gitt blir funksjonen strcpy brukt for å kopiere input strengen etter -i inn i addr variablen [57].
Linje 63-66: Om argumentet -p get gitt blir funksjonen atoi brukt for å gjøre om input strengen til en integer mer riktig verdi, lagrer der etter verdien inn i PORT variablen.
Linje 67-68: Om ingen argument blir gitt brukes standard verdiene, disse printes i kommando vinduet.

```
74: JoystickInput G29; //Lager ett objekt av G29_wheel_input.h klassen
78: G29.setup();
79: G29.setWheelRange(180);
80: G29.readEvent(); //Oppdaterer rattverdier.
```

Kodeliste 3.23: Klientkode - oppsetts-deklarasjoner c

Del c, kodeliste 3.23:
Opprettet ett objekt fra joystickInput klassen fra headerfilen. kjører setup funksjonen og oppdaterer verdier, for mer info om dette se seksjonen om programvare for input fra ratt/pedal.

```
53: struct sockaddr_in serv_addr;
93: serv_addr.sin_family = AF_INET;
94: serv_addr.sin_port = htons(PORT);
```

Kodeliste 3.24: Klientkode - oppsetts-deklarasjoner d

Del d, kodeliste 3.24:
Linje 53: Oppretter en struct av typen sockaddr_in. Denne strukturen har fire undervariabler. Av de brukes disse tre:

1. sin_family en short variablen som inneholder hvilken IP protokoll som skal brukes. (IPv4 eller IPv6)
2. sin_port en unsigned short variabel som inneholder port nummer.
3. sin_addr"inneholder IP adressen.

[58]
Linje 93: Setter sin_family til AF_INET det vil si IPv4
Linje 94: Setter sin_port til den definerte portverdien fra del a. htons funksjonen blir brukt for å konvertere dataen til standard form for nettverkstransmisjon [59]. Dette blir gjort for å unngå endian problemer. Endian problemer kommer av at ikke alle CPUer leser av data på samme måte [60].

```
87: if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
   {
89:    std::cout << ("\n␣[Klient]\t␣Kunne␣ikke␣opprette␣socket␣ERROR!␣\n
    ") << ::endl;

90:    return -1;
   }
```

Kodeliste 3.25: Klientkode - oppsetts-dekla rasjoner e

Del e, kodeliste 3.25:
linje 87:
Oppretter en ikke-bundet TCP/IP socket ved bruk av funksjonen socket() [61].. Socket() funksjonen tar argumentene AF_INET og SOCK_STREAM. AF_INET for IPv4 og SOCK_STREAM for TCP.
Socket() funksjonen returnerer -1 ved feil, derfor blir den opprettet i en if funksjon som sjekker om retur verdien er mindre en null.
Linje 89-90: Om socket() funksjonen gir tilbake verdien -1 vil det printes en feilmelding i terminalen og programmet termineres ved bruk av return -1.

```
97: if (inet_pton(AF_INET, addr, &serv_addr.sin_addr) <= 0)
{
99:    std::cout << ("\n[Klient]\t␣inet_pton␣feilet␣sjekk␣om␣IP␣adressen
    ␣er␣korrekt!\n") << ::endl;
100:    return -1;
}
```

Kodeliste 3.26: Klientkode - oppsetts-deklarasjoner f

Del f, kodeliste 3.26:
Linje 97: bruker inet_pton() funksjonen som tar IP versjon, input adresse, output-adresse [62].
funksjonen brukes for å konvertere den gitte ip adressen fra tekst til en binærverdi.
Fordi funksjonen returnerer adressen i binær form i sin tredje parameter,
brukes funksjonens egne return for error håndtering.
Her brukes en if settning for å sjekke at verdien er større eller lik 0,
da feilmeldinger har verdier under 1 fra denne funksjonen.
Linje 99 & 100: Om inet_pton() funksjonen gir tilbake verdier under 0 vil det printes en feilmelding i terminalen og programmet termineres ved bruk av return -1;

```
103: if (connect(sock, (struct sockaddr )&serv_addr, sizeof(serv_addr))
     < 0)
{
105:    std::cout << ("\n[Klient]\t␣Oppkobling␣feilet!␣Avsluetter...␣\n"
    ) << ::endl;
106:    return -1;
}
```

Kodeliste 3.27: Klientkode - oppsetts-deklarasjoner g

Del g, kodeliste 3.27:
Linje 103: Her brukes connect funksjonen for å binde socketen som ble opprettet i del d) til tjeneren [63]. Dette vil si at det etableres en oppkobling der det fritt kan sendes data senere.
Linje:105 & 106: Om connect() funksjonen returnerer en verdi mindre en 0 vil det printes en feilmelding i terminalen og programmet termineres ved bruk av return -1 [63];

Lese av data fra ratt/pedal

```
127: G29.readEvent(); //Oppdaterer ratt/pedal-verdier.
128: data[0] = G29.eventValue(); //leser inn fra ratt/pedal
129: data[1] = G29.eventCode(); //leser inn fra ratt/pedal
```

Kodeliste 3.28: Klientkode - Lese av data fra ratt/pedal

Disse funksjonene kan man lese mer om under rattavlesnings seksjonen.
Kodeliste 3.28:
linje 127: Oppdaterer de interne verdiene på input objektet.
linje 128: Leser av verdien på eventValue() og lagrer det i data vektoren posisjon 0. Denne verdien inneholder informasjon om verdien på eksempel, ratt eller pedal.
linje 129: Leser av fra eventCode() og laster det inn i datavektoren posisjon 1. Denne verdien inneholder informasjonen om hvilke input det er. For eksempel ratt eller pedal.

Filter

```
5:  #define BRAKE_CODE 3
6:  #define ACCELERATION_CODE 2
7:  #define WHEEL_CODE 0
8:  #define CLUTCH_CODE 1
9:  #define MINUS_BTN_CODE 20
10: #define PLUSS_BTN_CODE 19
11: #define accInk 10// Ikke i bruk.
12: #define wheelInk  5
```

Kodeliste 3.29: Klientkode - Filtervariabler

Variabler, kodeliste 3.29:
Her knyttes verdiene til datatypene som kommer i fra ratt/pedal. Alle variablene i stor skrift svarer til forskjellig code events. AccInk og wheelInk er variabler for inkrementfiltrering. Disse brukes for å kutte ned på hvor ofte endring sendes, i tilfellet wheelInk sendes det for eksempel verdier for hver 5 økning eller reduksjon.

```
134: if (data[0]!=preData || ( (data[1] == MINUS_BTN_CODE) || (data[1] == PLUSS_BTN_CODE)
    ))
//For sending av knappetrykk
}
```

Kodeliste 3.30: Klientkode - Filter a

del a, kodeliste 3.30:
Filter b realiseres ved bruk av en if-setning. I if-setningen sjekkes det først om verdien i data[0] er lik den foregående verdien dette med mellomlagringsvariablen preData. Dette for å unngå å sende samme verdi flere ganger. Videre sjekkes det om data[1] som inneholder informasjon om hvilken type verdien er lastet inn i data[0]. Stemmer data[1] med kode for minus-knappetrykk eller pluss-knappetrykk, og data[1] ikke er like foregående verdi settes if-setningen til sann og data sendes. Til slutt settes oppdateres preData til siste sendte verdi.

```
/ PEDAL-FILTER /
139: if (data[1] == BRAKE_CODE ||
140: data[1] == ACCELERATION_CODE ||
141: data[1] == CLUTCH_CODE)
142: {
143: if ((data[0]>preData+accInk)||(data[0]<preData-accInk)
144: ||( (data[0]==0)||(data[0]>252) ) )
145: //Send alltid 0-verdi minst 1 gang. PEDAL
146: //send toppverdi minst engang. PEDAL
147:    {
148: send(sock,&data, 2 sizeof(int), 0); //sender arrayet data.
149: preData = data[0];
150: //sammenligningsvariabel. Brukes til å sammenligne ny og gammel ratt_value.
151: preMillis= (std::chrono::system_clock::now().time_since_epoch()/ std::chrono::
    milliseconds(1));

            }
    }
```

Kodeliste 3.31: Klientkode - Filter b

del b, kodeliste 3.31:
I dette filteret filtreres pedaltråkk. If-setningen på linje 139-142 sjekker om verdien i data[1] svarer til en av de tre predefinerte pedaltråkkene. Hvis denne if-setningen blir sann, går koden videre inn i en ny if-setning. Denne if-setningen sjekker om data[0] som inneholder eventValue, har endret seg mer enn terskelverdien accInk, eller om dataen er enten 0 eller over 252. De siste to er der for å alltid forsikre at bunn og toppverdi alltid blir sendt. Spesielt viktig er bunnverdien, da denne gjør at bilen stopper.

```
162: if (data[1] == WHEEL_CODE)
163: {
164:  if((data[1]==0)||          //Send bunnverdi ratt
165:  (data[0]==180) ||          //send toppverdi ratt
166:  (data[0]==90)  || //send midtstilt ratt.
167:  ((data[0]>80 && data[0] < 100 )) //send alle verdier 10 over eller under midtstilt
168:   )
169:  {
170:  send(sock,&data, 2 sizeof(int), 0); //sender arrayet data.
171:  preData = data[0];
172:  //sammenligningsvariabel. Brukes til å sammenligne ny og gammel ratt_value.
173:  preMillis= (std::chrono::system_clock::now().time_since_epoch() /
174:  std::chrono::milliseconds(1));
175:  }
176: }
```

Kodeliste 3.32: Klientkode - Filter c

del c, kodeliste 3.32:
Likt som ved pedaltråkk, filtreres også data knyttet til rotasjon av rattet.
Første og fremst sjekkes det om dataen faktisk er knyttet til ratt-rotasjon ved å se på data[0]. Hvis denne if-setningen blir utløst går programmet videre til en

ny if-setning på linje 164-167. Denne if-setningen tar seg av selve filtreringen av verdiene. Her sjekkes det om verdien er enten er bunnverdi 0, toppverdi 180 eller midtstilt 90. Utover dette slippes også verdier mellom 80 og 100 alltid igjennom. Dette for å ha finjusterings-verdier når rattet er nært midtstilt posisjon. Verdier utenfor område til de nevnte områdene sendes kun hvis verdien er økt med mer enn inkrement-terskelen wheelInk.

Sending av data

```
148: send(sock,&data, 2 sizeof(int), 0);
170: send(sock,&data, 2 sizeof(int), 0);
185: send(sock,&data, 2 sizeof(int), 0);
204: send(sock,&data, 2 sizeof(int), 0);
```

Kodeliste 3.33: Klientkode - Sending av data

kodeliste 3.33:
send() funksjonen sender datane til tjeneren, første argument gir man socketen som skal brukes, andre argument gir man datane. Her må det spesifiseres størrelsen på datane som skal sendes. Det skal sendes to ints derfor spesifiseres det 2*sizeof(int) som er 64 bit [64].

Ettersending av data

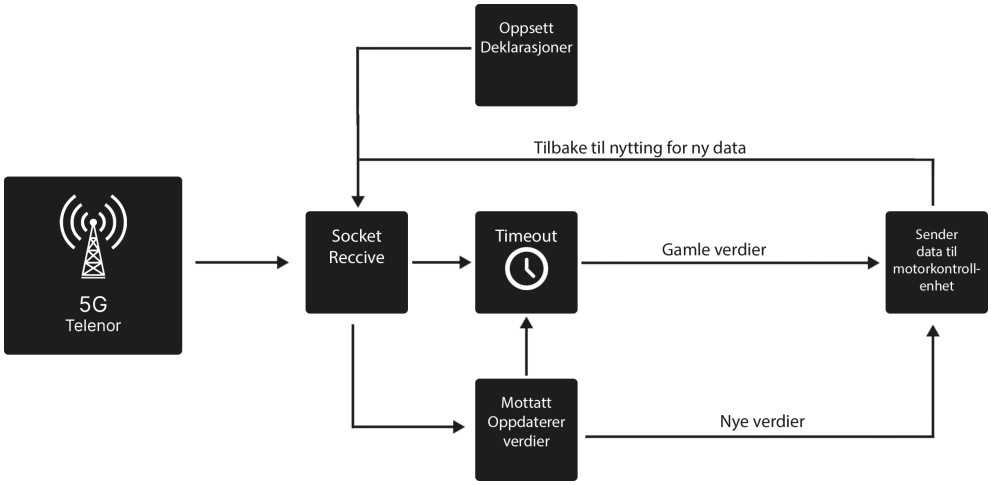
```
delta_t = ( (std::chrono::system_clock::now().time_since_epoch()/std::chrono::
    milliseconds(1))-preMillis);
if(delta_t >30)
 {
  send(sock,&data, 2 sizeof(int), 0); //sender arrayet data.
    preMillis
  (std::chrono::system_clock::now().time_since_epoch() /
  std::chrono::milliseconds(1));
 }
```

Kodeliste 3.34: Klientkode - Ettersending av data

kodeliste 3.34:
Denne funksjonen har blitt implementert i løsningen for å sende gammel data hvert 30ms om ingen ny informasjon kommer i fra piloten. Dette er for at fail-safenpå bilen skal kunne fungere. fail-safenslår inn om ingen data har blitt motatt etter 100ms. Sending av data hvert 30ms realiseres ved bruk av chrono-biblioteket [65]. Premillis settes til 'current-millis' som utledes fra chrono-funksjonen. Dette er tiden gått siden Unix epoch[65]. Denne går inn i delta_t-beregningen før if-setningen, hvor den sammenlignes med current-millis igjen. Hvis delta_t da er større enn 30ms, vil if-setningen utløses, og ferskest data lastet inn i data[0] og data[1] sendes igjen med send-funksjonen.

Tjener

Tjeneren er basert på Linux sine innebygde funksjoner og man kan se skjematikk av løsningen i figur 3.20. Det er ikke avhengig av eksterne biblioteker, som i oppsettet for klienten. Det benytter seg av TCP/IP. Det blir også definert en signal interrupt som setter stop variabelen "høyfor å avslutte main løkken for ordentlig avslutning av programmet der socketen blir lukket. For at dette skal fungere settes timeout på 10 sekunder slik at programmet ikke henger på receive funksjonen mer en det gitte tidsrommet. Når klienten er oppkoblet tar tjeneren i mot data og sender dette videre til motorkontrolleren via SPI. Man kan også gi tjeneren argumenter når man starter programmet som på klienten. -i for å endre ip adressen, -d for port. Dette fungerer på samme måte. Figur 3.19 viser tjenerkodens arkitektur.



Figur 3.20: Diagram tjener arkitektur.

Oppsett deklarasjoner

```
84: struct timeval timeout; //Timeout for ctrl+c
85: timeout.tv_sec = 10;
86: timeout.tv_usec = 0;
```

Kodeliste 3.35: Tjenerkode - Oppsett deklarasjoner a

Del a, kodeliste 3.35:
timeval struct fra sys/time.h
[66]

```
89: int lytteSock;
90: if ((lytteSock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
92:   cout << ("\n [Klient]\t Kunne ikke opprette socket ERROR!\n") << endl;
93:   return -1;
    }
```

Kodeliste 3.36: Tjenerkode - Oppsett deklarasjoner b

Del b, kodeliste 3.36:
Oppretting av lytte socket, denne vil håndtere nye oppkoblinger og binde de til hver sin unike socket. Ettersom løsningen kun støtter en klient vil denne kun bli brukt en gang. Om socket returnerer med verdi -1 avslutter programmet. [61]

```
97:  sockaddr_in serv_addr;
98:  serv_addr.sin_family = AF_INET;
99:  serv_addr.sin_port = htons(PORT);

100: inet_pton(AF_INET, "0.0.0.0", &serv_addr.sin_addr);
```

Kodeliste 3.37: Tjenerkode - Oppsett deklarasjoner c

Del c, kodeliste 3.37:
Linje 97-99: blir brukt på samme måte som forklart i:
klient, oppsetts-deklarasjoner, del c.
hint.sin_family = AF_INET; for å bruke IPv4 [58].
hint.sin_port = htons(54000); for å lytte på port 54000, htons() for å unngå endian feil [59].
Linje 100: inet_pton funksjonen brukes for å konvertere IPv4 adressen fra tekststreng til binærdata. Som i underkapittelet:
klient, oppsetts-deklarasjoner, del e.
Bruker IPv4 adressen: '0.0.0.0' for å lytte på alle nettverksgrensesnitt hos tjeneren. Dette kan byttes ut med en bestemt adresse for å for eksempel kun lytte på mobilnett grensesnitt (wwan0) eller kablet ethernet (eth0) ved å bruke deres gitte adresse [62].

```
103:   bind(lytteSock, (sockaddr)&serv_addr, sizeof(serv_addr));
104:   cout << "Tjener lytter..." << endl;
```

Kodeliste 3.38: Tjenerkode - Oppsett deklarasjoner d

Del d, kodeliste 3.38:
Binder listening socketen til verdiene gitt i **Del c** [67]

```
110:    sockaddr_in klient;
111:    socklen_t klientSize = sizeof(klient);

113:    int klientSock = accept(lytteSock, (sockaddr )&klient, &klientSize);

115:    char host[NI_MAXHOST];
116:    char service[NI_MAXSERV];

118:    memset(host, 0, NI_MAXHOST);
119:    memset(service, 0, NI_MAXSERV);
```

Kodeliste 3.39: Tjenerkode - Oppsett deklarasjoner e

Del e, kodeliste 3.39:
Linje 110-111 oppretter en ny socket struct som arver verdiene ifra lyttesock i **linje 3**

Linje 113-116 lager to character arrays med lengden til NI_MAXHOST NI_MAXSERV [68].

Linje 118-119 Setter alle characters i host og service til 0 [69].

```
122: if (getnameinfo((sockaddr )&klient, sizeof(klient), host, NI_MAXHOST,
    service, NI_MAXSERV, 0) == 0)
    {
124:  cout << host << "␣oppkobblet␣på␣port␣" << service << endl;
    }
    else
    {
128:  inet_ntop(AF_INET, &klient.sin_addr, host, NI_MAXHOST);
119:  cout << host << "␣oppkobblet␣på␣port␣" << ntohs(klient.sin_port) << endl;
    }
```

Kodeliste 3.40: Tjenerkode - Oppsett deklarasjoner f

Del f, kodeliste 3.40:
Forsøker å bruke 'getnameinfo' for å konvertere fra nettverkets binære data til streng/char array for at det skal være enkelt å lese for brukeren [70].
Om dette ikke fungerer brukes inet_ntop som fungerer motsatt en inet_pton forklart tidligere [62].
Begge disse metodene vil føre til konvertering fra binær data til lesbar streng for leseren for klientens host-navn (IPv4 adresse) og port.

```
132:  if (setsockopt (klientSocket, SOL_SOCKET, SO_RCVTIMEO, (char )&timeout,
    sizeof(timeout)) < 0)
135:  close(listening);
```

Kodeliste 3.41: Tjenerkode - Oppsett deklarasjoner g

Del g, kodeliste 3.41:
Her settes socketopt og gir den timeout structuren fra **del a**
Dette gjør slik at socketen timer"ut etter gitt tid. [71]

```
46: volatile sig_atomic_t stop;

144: signal(SIGINT, interruptHandler);



48: void interruptHandler(int signum)
   {
50:   stop = 1;
   }

146: while (!stop)
147: {
       //motta data
       //skrive til arduino
177: }

180:  close(klientSock);
181:  cout<< "Programmet avsluttet vellykket socket er stengt! "<<endl;
182:  return 0;
```

Kodeliste 3.42: Tjenerkode - Oppsett deklarasjoner h

Del h, kodeliste 3.42:
Ren avslutning av programmet. **Linje 46:** er stop variabelen gitt i det globale skopet. Datatypen sig_atomic_t brukes ved interrupts, og disse kan endres når programmet mottar ett signal som foreksempel ctrl+c for å avslutte [72].

Linje 144: setter opp interrupt rutine i programmet. Når programmet får inn SIGINT, det vil si signalinterupt som kommer ifra ctrl+c, vil interruptHandler funksjonen beskrevet i linje 3-4 kjøres [72].

Linje 48-50: viser en enkel funksjon som setter stop til verdien 1. Dette fører til at hovedløkken brytes som vist i linje 5. Dette fører igjen til at løkken avsluttes og socketen kan lukkes som beskrevet i linje 180-182. Programmet vil da avslutte med return 0;

Lytting og databehandling

```
151: int recvData = recv(klientSock, &data, 2 sizeof(int), 0);
152:  if (recvData == -1)
      {
154:      cout << "Timeout. Continue.." << endl;
155:      continue;
      }

158:  if (recvData == 0)
      {
160:      cout << "Klienten avsluttet! " << endl;
161:      break;
      }
```

Kodeliste 3.43: Tjenerkode - Lytting og databehandling

kodeliste 3.43:
Linje 151: viser hvordan tjeneren mottar data med funksjonen recv(). Den mottatte dataen vil bli skrevet til data-variabelen [73].

linje 152-155 håndterer timeouts. Om tjeneren ikke mottar data innenfor tiden satt i timeout vist i **del a** vil socketen timeout og forsette løkken.

linje 158-161 håndterer termineringen fra klientsiden. Om klienten avslutter programmet på sin tide vil programmet bryte ut av løkken og avslutte.

SPI fra Raspberry Pi:
Kodeliste 3.44
For å sende/motta data over SPI fra Raspberry Pi brukes et virtuelt dokument kalt "/dev/spidev0.0". Denne åpnes i starten av int main() og overføringshastigheten settes til 1 MHz. Ved hjelp av ioctl(dokumentnavn, oppgave som skal utføres, verdi) settes overføringshastigheten til 1 MHz på dokumentet. Samme syntax brukes for å skrive til MOSI og å lese fra MISO.

```
57:     fd = open("/dev/spidev0.0", O_RDWR);
58:     unsigned int speed = 1000000; //1Mhz clock på overforinga.
59:     ioctl (fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
```

Kodeliste 3.44: Oppsett av SPI på Raspberry Pi

sendCommand:
Kodeliste 3.45
Funksjonen sendCommand"CODE og VALUE fra ratt/pedaler som input og sender verdiene videre til spiTxRx for videre sending over SPI. Først sendes en byte som indikerer starten på en ny melding('c'). Deretter sendes CODE og til slutt VALUE. Samtidig som VALUE sendes sjekker den om slaven har sendt bekreftelse på at de 2 første bytene er mottatt. Er de det avslutter Raspberry pi sendingen etter siste

byte er sendt. Delay i sendCommander for å gi slaven tid til å lese inn verdiene før en ny verdi sendes.

```
217:  void sendCommand(int j, int k)
    {
228:                    spiTxRx('c');
229:                    usleep(10);

231:                    spiTxRx(k);
232:                    usleep (10);

234:                    spiTxRx(j);
235:                    usleep (10);

    }
```

Kodeliste 3.45: Funksjon for forsinkelse mellom hver SPI data

spiTxRx:
Kodeliste 3.46
Funksjonen 'spiTxRx' tar byte som skal sendes til argument og returnerer mottatt byte. Byte som skal sendes legge i spi.tx_buf og byte som mottas legges i spi.rx_buf. Deretter brukes ioctl med meldingslengde på 1 byte og verdi lik spi for å skrive til /dev/spidev0.0 og dermed sende/motta over SPI. Når den har gjort dette returener den eventuelle mottatte verdier til sendCommand".

```
197:    int spiTxRx(unsigned char txDat) //Definerer funksjonen.
    {
200:      unsigned char rxDat;

202:      struct spi_ioc_transfer spi;

204:      memset (&spi, 0, sizeof (spi));

206:      spi.tx_buf        = (unsigned long)&txDat;
207:      spi.rx_buf        = (unsigned long)&rxDat;
208:      spi.len           = 1;

210:      ioctl (fd, SPI_IOC_MESSAGE(1), &spi);

212:      return rxDat;
    }
```

Kodeliste 3.46: Skriver/leser til spidev fil for SPI

3.3.3 Kode på motorkontroller

Hovedoppgaven til arduinokoden er å motta ratt/pedalverdier, endre det til PWM signaler, og deretter sende det ut på ESC/styringsservo. Koden er hovedsaklig

skrevet i AVR-C og nesten ingen bibliotek er brukt. Den ferdige koden er skrevet for ATmega4809 på en Arduino Nano Every og alt av nødvendige register og pinouts er funnet i datablad til mikrokontrolleren [7][8] eller i biblioteksfiler lastet ned gjennom Arduino IDE. Arduino IDE er brukt til kompilering og kjøring, fordi det er en av få IDE som støtter ATmega4809 ettersom det er en relativt ny chip. Fullstendig kode finnes i vedlegg D.1.

PWM

For å aktivere PWM på ATmega4809 må CTRL register for TCA0 settes som vist i kodeliste 3.47. De settes med 'SINGLE' ettersom PWM skal kjøres med en 16-bit teller og ikke to 8-bit tellere. I CTRLB må CMP0EN og CMP1EN settes for å skru på PWM på hhv. pinne 9 og 10. WGMODE_SINGLESLOPE settes også for å sette PWM i singleslope modus hvor den teller til en gitt verdi og deretter nuller ut telleren(Linje 1). PERBUF som angir toppverdien til telleren settes til 10 000 fordi dette gir en periodetid på 10ms med en 16MHz klokke(Linje 2).

$$(\text{Klokkefrekvens i MHz} * \frac{periodetid\,i\,ns}{prescaler}) = \text{Toppverdi på teller}$$

I CTRLA registeret settes CLKSEL_DIV16 for å velge en prescaler på 16, som er det beste valget i dette tilfellet ettersom det er lett å oppnå 100Hz med det. ENABLE settes deretter høy få å begynne å telle. Pinnene som skal brukes til PWM må også settes som output ettersom de er satt som input ved oppstart av Arduino(Linje 5).

For å endre PWM signalet settes 16-bit registrene CMP0BUF og CMP1BUF til ønsket verdi. Dette er sammenligningsregister som sammenlignes med telleren og dermed bestemmer hvor mye av PWM signalet som er 'på'. For alle tellerverdier under CMPnBUF registerets verdi er pinnen høy, og for alle over er den lav. Både ESC og styringsservo ønsker i teorien PWM signaler på mellom 10 og 20% av periodetiden , men etter litt testing ser man at styringsservoen vil ha verdier mellom 11 og 19% av periodetiden. Dette er fordi svingradiusen på hjulene begrenser maksimalutslag på servoen.

På er den satt til verdier mellom 10 og 20% av periodetiden fordi dette er verdiene en servo krever for 180% utslag. Verdiene på ESC har egentlig ikke veldig mye å si ettersom ESC kan kalibreres mellom maksimal PWM og minimal PWM utifra hva som er ønskelig. Under void setup() settes PWM til verdier som ikke gir utslag på ESC/styringsservo for å forhindre uventet oppførsel under start(Linje 3 og 4).

```
    //Setter pinne 9 og 10 som output for TCA0 PWM, PWM i singleslope mode
1:  TCA0.SINGLE.CTRLB |= TCA_SINGLE_CMP0EN_bm | TCA_SINGLE_CMP1EN_bm |
    TCA_SINGLE_WGMODE_SINGLESLOPE_gc;
```

```
2:  TCA0.SINGLE.PERBUF = PWM_PERIODE; //Setter periodetiden på TCA0 PWM, skal v
    ære 100Hz
3:  TCA0.SINGLE.CMP0BUF = SERVO_NULL;//Setter startverdi på pinne 9
4:  TCA0.SINGLE.CMP1BUF = ESC_NULL;//Setter startverdi på pinne 10

  //Div klokke med 16 for riktig periodetid på PWM, skrur på telleren
5:  TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV16_gc | TCA_SINGLE_ENABLE_bm;
```

Kodeliste 3.47: Oppsett av PWM på mikrokontroller

Utregning av PWM signaler

PWM verdier kan regnes ut på to måter. Første måte er å bruke formelen gitt i kapittelet PWM, men for 'periodetid i ns' brukes en prosent av 'periodetid i ns'. Den andre måten er å bare ta en prosent av den utregnede verdien for toppverdi på teller som vist i kodeliste 3.48. Deretter kan verdiene for maks- og minimumsutslag brukes for å regne ut verdier for nullutslag.

```
6:    #define PWM_PRESCALE 16  // Timer 1 clock prescaler

    // Periodetiden på PWM
7:    #define PWM_PERIODE  F_CPU/100000010000/PWM_PRESCALE

    // Maksimalverdi på servo
8:    //#define SERVO_MAX   F_CPU/10000001900/PWM_PRESCALE //1900
9:    #define SERVO_MAX PWM_PERIODE 0.19
    // Minsteverdi på servo
10:   //#define SERVO_MIN   F_CPU/10000001100/PWM_PRESCALE //1100
11:   #define SERVO_MIN PWM_PERIODE 0.11
    // Nullverdi på servo
12:   #define SERVO_NULL SERVO_MIN+((SERVO_MAX-SERVO_MIN)/2) //1500

    // Maksimalverdi på ESC
13:   //#define ESC_MAX   F_CPU/10000002000/PWM_PRESCALE //2000
14:   #define ESC_MAX   PWM_PERIODE 0.1
    // Minsteverdi på ESC
15:   //#define ESC_MIN   F_CPU/10000001000/PWM_PRESCALE //1000
16:   #define ESC_MIN   PWM_PERIODE 0.20
    // Nullverdi på ESC
17:   #define ESC_NULL ESC_MAX+((ESC_MIN-ESC_MAX)/2) //1500
```

Kodeliste 3.48: Utregning av PWM signaler til ESC og styringsservo

SPI

For SPI må MISO pinnen settes som output, og MOSI, SCK og SS skal være input. SS pinnen brukes ikke her fordi Arduinoen er eneste slave. Det er også nødvendig med en PORTMUX, selv om standard SPI pinner skal brukes. PORTMUXEN setter pinne 8, 11-13 i SPI modus(Linje 18) som vist i kodeliste 3.49. For å aktivere SPI settes SPI_ENABLE høy og SPI_MASTER lav i CTRLA registeret(Linje 19). SPI_MASTER lav holder Arduinoen i slave modus. SPI_IE i INTCTRL skrur

på interrupts for SPI(Linje 20). Deretter bruke sei() for å aktivere globale variabler(Linje 21).

```
18:  PORTMUX.TWISPIROUTEA |= PORTMUX_SPI0_ALT2_gc;

    //SPI aktivert, SPI slave-mode.
19:  SPI0.CTRLA =  SPI_ENABLE_bm & (~SPI_MASTER_bm);

  //Aktiverer IE flag. interrupt.
20:  SPI0.INTCTRL = SPI_IE_bm;

21:  sei(); //Skrur på globale interrupts
```

Kodeliste 3.49: Oppsett av SPI på mikrokontroller

For å lese inn/skrive ut verdier over SPI brukes en switch case med en tellevariabel kalt 'marker' som vist i kodeliste 3.50. SPI består av tre forskjellige deler og når en del er ferdig inkrementeres marker med 1 og neste del kan starte. Alt dette skjer innenfor interrupt på SPI, altså hver gang data er tilgjengelig på MOSI pinnen. I del 'A' ser Arduinoen etter en kontrollbyte med verdien 'c'. Master vil alltid starte en dataoverføring med denne kontrollbyten, og om sending/mottak går ut av synkronisering vil denne byten holde igjen koden så alt blir synkronisert igjen(Linje 24-28).

Del 'B' leser inn påfølgende byte, som er CODE. Denne legges i en arrray for å kunne brukes videre.(Linje 29-32).

Del 'C' tar i mot neste verdi, som er VALUE, og lagrer den i samme array(Linje 34). Deretter kommer en test for å sjekke om noen av kontrollbytene har blitt lest inn for CODE eller VALUE(Linje 35). Har de ikke det sendes verdiene videre til funksjonen handleData som prosesserer og skriver ut verdiene på PWM(Linje 37). Er det en feil hopper koden over dette steget og starter ventingen på en ny kontrollbyte. Dette er på ingen måte en perfekt måte å løse det på, men den vil sjeldent føre til feil, eventuelle feil vil ha få konsekvenser og det er en veldig lett løsning å implementere/forstå.

```
22:    void spiHandler()
      {
23:      switch (marker)
        {
24:        case 0:
25:          dat = SPI0.DATA; //leser inn data fra SPI-dataregister.
26:          if (dat == SPI_START_BYTE) //Starten av all overføring av CODE og
    VALUE med START_BIT.
            {
27:            marker++;// ker  marker slik at neste case utløses.
            }
28:          break;
```

```
29:         case 1:
30:           reccived_array[1] = SPI0.DATA; //CODE innlesing fra SPI-
    dataregister.
31:           marker++; // ker  marker slik at neste case utløses.
32:           break;

33:         case 2:
34:           reccived_array[0] = SPI0.DATA; //VALUE innlesning fra SPI-
    dataregister.
35:           if (dat != reccived_array[0] && dat != reccived_array[1]) //
    Sjekker om feil verdi er lest inn i CODE eller VALUE
            {

36:                counter = 0; //Resetter teller for failsafe

               /Håndterer mottatte verdier med handleData(CODE,VALUE) /
37:                handleData(reccived_array[1], reccived_array[0]);
            }

38:            else //Funnet feil på CODE
            {
39:                Serial.println('Feil på mottatt verdi');
            }

40:            marker = 0; //Setter marker tilbake til null, slik at avlestning
    av SPI-data begynner på nytt igjen.
41:           break;
      }
    }
```

Kodeliste 3.50: Funksjon for innlesning av SPI verdier på mikrokontroller

Denne løsningen er hentet fra kode skrevet av Ralph Heymsfeld, men modifisert til formålet[74].

handleData

handleData tar inn verdier mottatt over SPI, endrer det til PWM signaler, og sender det ut til ESC og styringsservo som vist i kodeliste 3.51. Også her er det brukt en switch case, men argumentet er CODE. CODE forteller Arduinoen hva som skal bevege seg på bilen f.eks. om den er 0 er det rattet som er vridd på og signalet må gå ut på styringsservoen.

Ved CODE = 0 settes CMP0BUF til minsteverdi for servo + VALUE * wheelMultiplier. wheelMultiplier er en verdi som gjør om de 180 verdiene som kan mottas for rattet til de 1640 verdiene som kan skrives ut på PWM signalet(Linje 3-5).

CODE = 1 er revers når clutchpedalen trykkes. Her er det verdier mellom 920

og 2560 som skal skrives ut på PWM og 255 verdier som kommer inn fra SPI. 2560 er nullverdien til ESC og lavere verdier enn dette gjør at bilen kjører bakover. Her deaktiveres også bremsepedalen(Linje 6-9).

CODE = 2 er gasspedalen og denne fungerer på akkurat samme måte som revers, men for verdier mellom 2560 og 4200 ut på PWM(Linje 11-14).

CODE = 3 er bremsepedalen og her settes bile i maks revers uansett hvilken VALUE som kommer inn. Dette er for å gi maksimalt utslag på bremsen ettersom de ikke er veldig kraftige. Det er viktig at bremsen kun kan trykkes på en gang, fordi når den er trykket på en gang settes ESC i reversmodus og ett nytt rykk vil sende bilen i revers med maks hastighet(Linje 15-19).
CODE = 4 og CODE = 5 er hhv. + og - knapper på rattet. Disse brukes for å bestemme maksfarten på bilen(Linje 20-25).

```
42:    void handleData(long int code, long int value) {

43:      switch (code)
     {
       //ratt
44:        case WHEEL_CODE:
45:          TCA0.SINGLE.CMP0BUF = SERVO_MIN + value   wheelMultiplier; //
    Skriver PWM signal ut på pinne 9 (Styringsservo)
46:          break;

       //Clutch
47:        case CLUTCH_CODE:
48:          TCA0.SINGLE.CMP1BUF = ESC_NULL + value   throttleMultiplier
    gearing; //Skriver PWM signal ut på pinne 10 (ESC)
49:          brakeCheck = false; //Deaktiverer bremsepedalen
50:          break;

       //gass
51:        case ACCELERATION_CODE:
52:          TCA0.SINGLE.CMP1BUF = ESC_NULL - value   throttleMultiplier
    gearing; //Skriver PWM signal ut på pinne 10 (ESC)
53:          brakeCheck = true; //Aktiverer bremsepedalen
54:          break;

       //brems
55:        case BRAKE_CODE:
56:          if (brakeCheck){
57:            TCA0.SINGLE.CMP1BUF = ESC_MIN; //Skriver PWM signal ut på pinne
    10 (ESC)
58:            brakeCheck = false; //Deaktiverer bremsepedalen
         }
59:          break;

60:        case PLUSS_BTN_CODE:
61:          gearing = 1;
62:          break;
```

```
63:         case MINUS_BTN_CODE:
64:           gearing = 0.50;
65:           break;
      }
    }
```

Kodeliste 3.51: Funksjon som sender ut riktig PWM signal

Failsafe

En failsafe må implementeres for situasjoner hvor det oppstår brudd i kommunikasjonen mellom input fra pilot og output på bil. Et eventuelt tap av kommunikasjon i høye hastigheter kan være katastrofalt for bilen. I void loop() er det en teller som inkrementeres med 1 hvert 10 millisekund som vist i kodeliste 3.52. Når denne telleren når 20 settes motorverdien til ESC_NULL og slår av motoren. Dette gjør at om pilot mister kontakt med fører, uansett hvor i kommunikasjonen, vil bilen stoppe etter rundt 200 millisekunder. På grunn av interrupts er ikke dette alltid nøyaktig. Piloten sender verdier hvert 30ms, uavhengig av input på ratt/pedaler. Dermed vil failsafe kun aktiveres ved brudd i kommunikasjonen. Telleren til failsafe resettes hver gang en ny verdi mottas over SPI(Linje 18 under SPI).

```
66: maxCount = 20; //Teller hver itterasjon av hovedloop...

67:    delay(10);
68:    counter++; //For hver itterasjon av koden.

69:    if(counter>maxCount)
    {
70:      TCA0.SINGLE.CMP1BUF = ESC_NULL; //Stopper motoren

71:      counter = 0; //Restarter counter. Denne restartes også ved mottak av
    valid SPI-data.

    }
```

Kodeliste 3.52: Failsafe som stopper motoren ved brudd i kommunikasjon

Oppstart

Ved oppstart vil en led på mikrokontrolleren blinke for å vise at setup er ferdig. Deretter blir det satt maksutslag i begge retninger på styringsservo for å teste at alt virker. Til slutt settes to små pådrag på motoren for å teste at den også virker. Dette vises i kodeliste 3.53.

```
72:    void setup_done_LED(int led_pin)
    {
73:      for (int timer = 0;timer<500;timer+=50)
     {
```

```
74:         digitalWrite(led_pin,HIGH);
75:         delay(timer);
76:         digitalWrite(led_pin,LOW);
77:         delay(timer);
      }
    }

78:    void testServo_ESC()
    {

79:      TCA0.SINGLE.CMP0BUF = SERVO_MIN;
80:      delay(500);
81:      TCA0.SINGLE.CMP0BUF = SERVO_MAX;
82:      delay(500);
83:      TCA0.SINGLE.CMP0BUF = SERVO_NULL;
84:      delay(1000);
85:      TCA0.SINGLE.CMP1BUF = ESC_NULL - 44   throttleMultiplier   gearing;
86:      delay(500);
87:      TCA0.SINGLE.CMP1BUF = ESC_NULL - 0   throttleMultiplier   gearing;
88:      delay(500);
89:      TCA0.SINGLE.CMP1BUF = ESC_NULL - 44   throttleMultiplier   gearing;
90:      delay(500);

91:      TCA0.SINGLE.CMP1BUF = ESC_NULL;
    }
```

Kodeliste 3.53: Funksjoner for test av bil

Funksjonen setup_done_LED kjører etter at setup av mikrokontrolleren er ferdig. Den blinker med lavere og lavere frekvens helt til den når 500ms og deretter er den ferdig. testServo_ESC kjøres etter at blinkingen er ferdig og interrupts er aktivert. Først endrer den verdiene på styringsservoen med 500ms delay, deretter setter den to små pådrag på motoren som resulterer i to pip. Etter dette er bilen klar til å motta verdier.

3.3.4 VPN / Brannmur

Når Raspberry Pi kobler seg til Telenors mobilnett blir de tildelt fast, offentlig IPv4 adresse, uten at trafikken filtreres av noen form for brannmur hos Telenor. Det er satt opp en VPN forbindelse mellom de to Raspberry Pi-ene. VPN trafikken mellom enhetene går raskeste vei gjennom Telenors infrastruktur, uten noen ekstra mellomledd. Det å sende trafikken gjennom VPN medfører svært liten ekstra forsinkelse (mindre enn 1ms) [75].

Det er mulig å velge om man skal sende all kommunikasjon mellom Raspberry Pi-ene gjennom VPN tunnelen, eller å sende det i klartekst utenom VPN. Det avgjøres av hvilke IP-adresser man skriver i kommandoene for å starte video og styring. All trafikk som går gjennom VPN tunnelen vil være kryptert ende-til-ende. Videre er det satt opp brannmur på begge Raspberry Pi som blokkerer all innkommende trafikk fra det åpne internett, bortsett fra trafikk fra motsående Raspberry Pi og SSH tilkobling.

```
To                         Action      From
--                         ------      ----
22/tcp on eth0             ALLOW       Anywhere
22/tcp on wlan0            ALLOW       Anywhere
22/tcp on wwan0            ALLOW       Anywhere
51820/udp                  ALLOW       Anywhere
1:65535/tcp on wg0         ALLOW       Anywhere
1:65535/udp on wg0         ALLOW       Anywhere
Anywhere                   ALLOW       (IP på motstående Raspberry Pi)
```

Kodeliste 3.54: Brannmur konfigurasjon.

Kodeliste 3.54 viser brannmur konfigurasjonen på hver Raspberry Pi.
Pilot har offentlig IPv4 adresse: 78.158.240.251
Bil har offentlig IPv4 adresse: 78.158.241.23

C++ koden på pilot-pi sender styresignal til bilens IP, og GStreamer på bil-pi sender video til pilotens IP.

WireGuard benyttes som VPN løsning, all data WireGuard sender over mobilnettet sendes som UDP pakker. I WireGuard er det ingen innebygd server/klient arkitektur, alle noder i nettverket er i utgangspunktet likeverdige [76]. Begge Raspberry Pi vil kontinuerlig prøve å kontakte den andre, for å sette opp VPN tunnel. Dette gjør at VPN forbindelsen fungerer selv om bare én Raspberry Pi er tilkoblet mobilnettet. Dette kan være nyttig i situasjoner hvor bilen har 5G dekning, men utstyret for fjernstyring står inne i en bygning hvor det ikke er dekning. Da kan pilot-pi tilkobles kablet internett-forbindelse, og man vil kunne oppnå lavere forsinkelse enn hvis pilot-pi hadde vært tilkoblet 4G. Dette vil kun fungere om man sender data gjennom VPN. I tilfeller hvor begge Raspberry Pi er tilkoblet mobilnettet kan man sende data utenom VPN.

```
[Interface]
PrivateKey = (Fjernet fra dette dokumentet)
Address = 10.0.0.1/32
ListenPort = 51820

[Peer] #Bil
PublicKey = ixkeKcpszo4fGD6iNBTxNsPXA8YoC1wUHtL33R/sTwk=
AllowedIPs = 10.0.0.2/32
Endpoint = 78.158.241.23:51820 #IP tilhørende SIM-kort i bil
```

Kodeliste 3.55: WireGuard konfigurasjon - pilot-pi

```
[Interface]
PrivateKey = (Fjernet fra dette dokumentet)
Address = 10.0.0.2/32
ListenPort = 51820

[Peer] #Pilot
PublicKey = Drkv8BMAgOe6kbhEIjFe7yas3zkDbC5uo28/++kBTQA=
AllowedIPs = 10.0.0.1/32
Endpoint = 78.158.240.251:51820 #IP tilhørende SIM-kort i pilot
```

Kodeliste 3.56: WireGuard konfigurasjon - bil-pi

Kodeliste 3.55 og 3.56 viser WireGuard konfigurasjonsfiler. Dette er et svært enkelt VPN-nettverk, med kun 2 enheter som kommuniserer direkte med hverandre.

3.3.5 Video-overføring

GStreamer

For å overføre video benyttes GStreamer. GStreamer beskrives som et 'pipeline-based multimedia framework'. GStreamer inneholder en rekke moduler som kan settes sammen til å utføre oppgaver tilknyttet video/lyd prosessering [77]. I dette prosjektet benyttes GStreamer i alle ledd fra kamera, enkoding, sending over nettverk, dekoding og avspilling på skjerm.

Kodeliste 3.57 og 3.58 viser kommandoene som benyttes for å starte GStreamer for sending/mottak av video. Hver boks er en enkelt kommando, hver GStreamer modul er satt på egen linje for å gjøre det lettere å lese. Utropstegnet (!) benyttes for å indikere at data som kommer ut fra forrige modul skal mates inn i neste modul.

```
gst-launch-1.0

v4l2src do-timestamp=true extra-controls="a,h264_profile=0,video_bitrate_mode
    =1,h264_i_frame_period=1" !

video/x-h264,framerate=60/1,width=720,height=480 !

h264parse !

rtph264pay !

udpsink host=78.158.240.251 sync=false
```

Kodeliste 3.57: GStreamer kommando for å sende video

Forklaring av modulene i kommandoen over:

1. v4l2src: Hele prosessen starter med at modulen 'v4l2src' (VideoForLinux2 source) henter video fra kameraet, som enkodes med H.264 kodeken. 'extra-controls' gir noen ekstra parameter til H.264 dekoderen, disse er:
    - h264-profile=0: Velger baseline enkoding profil. Det er den enkleste formen for H.264 enkoding. Det krever lite prosessering og bevarer mye redundans i videoen som sendes
    - video-bitrate-mode=1: Konstant bitrate (10 Mb/s om ikke annet er spesifisert), gir lavere forsinkelse enn flytende bitrate, som er standard
    - h264-i-frame-period=1: Send alle bilder som I-frame. Det krever mindre prosessering og er i teorien mer robust enn om man skulle sendt mange P-frames
2. video/x-h264: Dette er et såkalt caps filter som setter begrensninger på dataen som går videre til neste modul [78]. Parameterene her påvirker forrige blokk, altså v4l2src. Her settes oppløsning og bildefrekvens

3. rtph264pay: Pakker H.264 enkodet video inn i RTP pakker som 'payload'.

4. udpsink: pakker RTP inn i UDP og sender de til pilot-pi på oppgitt IP adresse. Ved å skrive inn enten 78.158.240.251 eller 10.0.0.1 velger man om video skal gå åpent eller i VPN tunnel. sync=false gjør at pakker sendes av gårde så fort som mulig.

```
gst-launch-1.0

udpsrc port=5004 retrieve-sender-address=false !

application/x-rtp, encoding-name=H264, payload=96, a-framerate=60 !

rtph264depay !

h264parse !

omxh264dec !

videoconvert !

fbdevsink
```

Kodeliste 3.58: GStreamer kommando for å motta video

Forklaring av modulene i kommandoen over:

1. udpsrc: Mottar UDP pakkene sendt fra bil-pi

2. application/x-rtp: Caps filter, inneholder info om hva UDP pakkene inneholder.

3. rtph264depay: Pakker H.264 enkodet video ut av RTP pakkene som mottas.

4. h264parse: Leser H.264 bitstrøm, henter ut informasjon som er nødvendig for dekoding.

5. omxh264dec: Dekoding av H.264 video, til 'råvideo' som kan vises på skjerm. Dekoding gjøres av GPU på Raspberry Pi (maskinvareakselerert dekoding).

6. videoconvert: Konverterer mellom forskjellige 'color space' og 'YUV' formater

7. fbdevsink: Sender video ut på HDMI port til skjermen.

H.264 optimalisering

Basert på testing er disse innstillingene på H.264 enkoderen endret for å oppnå lavest mulig forsinkelse:

- Profile: Videoen som sendes fra bilen er enkodet med baseline profil, som er den enkleste formen for H.264 enkoding [21]. B-frames benyttes ikke. Dermed slipper man unna den ekstra forsinkelsen knyttet til B-frames.
- Fast bitrate: Ett annet triks som benyttes i H.264 video er varierende bitrate. Video deles opp i korte sekvenser. Om ønsket bitrate er satt til 10 Mb/s betyr det at i løpet av en kort tidsperiode vil bitraten i gjennomsnitt være 10 Mb/s, men den kan variere i løpet av den korte tidsperioden.
  Dette gjør at data må mellomlagres før den enkodes, slik at gjennomsnittlig bitrate i perioden blir korrekt. Dette skaper forsinkelse, i koden i kodeliste 3.57 benyttes fast bitrate for å unngå denne forsinkelsen.
- I-frame period: Det sendes bare I-frames, ikke P-frames.

Kapittel 4

Prosess

Prosessen startet med lesing av relevant materiale for å jobbe mot en løsning på oppgaven. Her ble det brukt kilder som datablad, lignende prosjekter gjort av andre. Etter mye lesing kunne gruppen designe en teoretisk løsning på oppgaven og begynne arbeidet med å realisere det teoretiske.

Etter at alle delene var mottatt begynte byggingen av selve løsningen. Under byggingen ble det tydelig at ting måtte tenkes gjennom på nytt og designet måtte endres. Dermed ble ting hele tiden endret etterhvert som oppgaven utviklet seg og mer kunnskap ga et bedre bilde på hva som måtte gjøres.

Når bilen var ferdig bygget begynte testingen. Det var også gjort tester på enkeltdeler underveis, men testene på bilen som en helhet viste de mest riktige resultatene. Disse testene viste at ting måtte endres så igjen ble ting redesignet og modifisert for et bedre sluttresultat. Når bilen var ferdig bygget og resultatene var så optimaliserte som tidsperspektivet for oppgaven tillot ble bilen ferdigstilt og måling av forsinkelse og lignende ble utført, dokumentert og brukt i rapporten.

4.1 Programvare og verktøy brukt under prosessen

Gantt

Gantt diagram var viktig for å strukturere oppgaven. Alle delene av prosjektet kunne samles på en plass, og fikk tildelt en egen tidsperiode i tillegg til en tidsfrist. Dermed kunne alle på gruppen vite hva som måtte jobbes på, hvilke deler som var ferdige og hva som manglet. For gantt ble nettisden clickup.com brukt.

Microsoft Teams

Microsoft Teams ble brukt for filoppbevaring og møter med oppdragsgiver. Det er også mulig å lage Wiki og tasks, noe som gjør at alt er samlet på en plass. Det

er lett å sende møteinnkallinger gjennom teams så dette ble brukt for møter med oppdragsgiver.

Miro

Miro ble brukt til å lage flowcharts i starten av prosjektet. Dette var for å lage den overordnete strukturen valgte løsning på en god måte. Deretter kunne mer detaljerte planer for hver del av strukturen lages utifra det.

Latex/Overleaf

Den ferdige rapporten ble skrevet i LaTeX, hovedsaklig på grunn av god støtte for kodeskriving. Det gir også bedre muligheter for design av oppgaven og er god på automatisering av kilder, innholdsfortegnelse, figurer, kildelister og ordlister. Overleaf ble brukt til dette

Timeliste

Timelister ble ført i et felles Excel dokument. Der kan timer for hver dag legges inn, hva personen har jobbet med den dagen og timene summeres for uke, toukersperiode og totalt. Timelistene gjør det enkelt for alle å se hva andre gruppemedlemmer jobber med.

Personlige handlingsreferat

Alle på gruppa skrev daglige handlingsreferat over arbeidet utført den dagen og eventuelle resultater. Dette gjorde det lettere å skrive sluttrapporten. De fungerer også som en mer utfyllende versjon av timelisten så arbeid utført er tydelig dokumentert.

Forsøksrapporter

For å dokumentere forsøk relatert til oppgaven ble det laget forsøksrapporter. Der ble målet med forsøket og resultatene dokumentert for å kunne brukes ved skriving av sluttrapport.

Facebook Messenger/Discord

Det meste av kommunikasjon internt internt i gruppa ble gjort skriftlig over Messenger eller muntlig over Discord, på grunn av den pågående korona-situasjonen.

Fysiske møter begynte ikke før langt ut i prosjektet når det ble nødvendig i test-sammenheng.

WireGuard nettverk for testing

Gjennom store deler av prosjektet jobbet alle hjemmefra, med egne Raspberry Pi. Det var behov for å kunne kommunisere mellom disse Raspberry Pi-ene. Det ble derfor opprettet et eget VPN nettverk som alle Raspberry Pi ble koblet til. Gjennom dette kunne det utføres testing som innvolverte flere Raspberry Pi, uavhengig av hvor disse fysisk befant seg.

Kapittel 5

Implementering

5.1 Programmeringsspråk

5.1.1 C++

C++ er et objektorientert programmeringsspråk som brukes til programmer med krav om høy ytelse [79].

5.1.2 Python 3

Python 3 er ett fortolket og objektorientert programmeringspråk [80]. Python 3 har mange innebygde kodebiblioteker samt ett stort åpentkildekode miljø som gir ut biblioteker som man enkelt kan installere ved bruk av PIP. PIP er ett verktøy for installasjon av pakker utviklett av andre. Python 3 har blitt brukt i prototypefasen for å teste idéer, samt behandle data for å lage grafer og lignende. Python 3 sin svakhet er at det bruker mye minne og er ofte tregere en kompilerte språk. Men det er også ofte mye raskere å utvikle en løsning ved bruk av Python 3. Ettersom oppgaven krever minst mulig treghet i systemet ble C++ valgt til oppgaven [81].

5.2 IDE Programmerings-software

5.2.1 GitHub

GitHub er en tjeneste der man kan laste opp kode eller annen teknisk dokumentasjon til prosjekter. GitHub har innebygd versjonkontroll og gode verktøy for oversikt over prosjekter. I denne oppgaven har det blitt brukt for å laste opp ny kode samt endringer på allerede utviklet kode. Vedhjelp av versjonkontrollen kan man enkelt se hvilke endringer de ulike deltakerene har gjort samt skrive kommentarer for å dokumentere prosjektet underveis [82].

5.2.2 Visual Studio

I oppgaven har det blitt brukt Visual Studio for utvikling av programmet i C++ [83]. Visual Studio Community er ett gratis integrert utviklingsmiljø. GitHub har mange funksjoner, men i oppgaven har det blitt benyttet deres innebygde git grensesnitt. det har også blitt anvendt 'remote compile' slik at man enkelt kan skrive program koden på Windows for å så sende koden til Linux maskinen for å kompileres/debugges [84].

5.2.3 Arduino IDE

Arduino IDE er et åpentkildekode integrert utviklingsmiljø for programmering av Arduino maskinvare. Her kan man også skrive i 'Arduino C'. Som er en rekke biblioteker samt noe endring på syntax for å enklere programmere mikrokontrollere [85].

5.2.4 Atmel Studio 7

Atmel Studio er ett integrert utviklingsmiljø for programmering av Atmel sine mikrokontrollere. Programmet er basert på Microsoft Visual Studio 'Shell' slik at grensesnittet er temmelig likt [86]. Atmel Studio har innebygde verktøy spesieltfor mikrokontrollere slik at man kan visuelt se simulering av for eksempel mikrokontrollerens registere.

5.3 evdev - event device

Evdev er ett generisk inngangs grensesnitt for å lese av data fra tilkoblete enheter via driverene deres. Driverene skriver data til spesialfiler i /dev/input/. Disse datane er standardisert for alle hardware arkitekturer, slik at det ikke er avhenge av at man bruker for eksempel samme leverandør for at det skal fungere. I oppgaven bruker vi joydev som bruker grensesnittene js0, js1 og oppover [87][47].

5.4 Verktøy

5.4.1 Tegneprogram

FreeCad

FreeCad er et 3D-modelleringsprogram som kan eksportere filer i .stl format. Dette er formatet som brukes for 3D-printing.

Sketchup 2017

Sketchup er ett 3D-modelleringsprogram som brukes innen arkitektur, mekanisk-design og videospill-design [88]. Sketchup har også ett miljø for utviklerpakker, blant disse pakkene finnes pakke for STL-eksport. Dette åpner opp for å eksportere filer til 3D-printing [89].

5.4.2 SSH-kommunikasjon

PuTTY
PuTTY er en open source SSH klient for Windows, det blir brukt for å kobble seg til Raspberry Pi [90].

WinSCP
WinSCP er et program som benyttes til filoverføring til/fra Raspberry Pi. WinSCP benytter SFTP (SSH File Transfer Protocol) [91].

5.4.3 Produksjon

For produksjon av 3D-design er det både benyttet 3D-printerne ultimaker 2+ extended og Makerbot Z18. Tilgang til printerne er både igjennom eksterne kontakter, samt elektronikk og prototypelaboratoriet ved institutt for elektroniske systemer. I begge tilfellene har kravene til produksjon vært .STL-filer.

5.4.4 notepad++

Er ett åpent kildekode tekstbehandlingsverktøy. Programmet har støtte for tekst utheving for en rekke programmering og script språk. Programmet er veldig lettvektig å er veldig egnet til både utvikling der man bruker en andreparts kompilator/tolk manuelt vedsiden av utviklingen. det er også svært egnet til å lese eller redigere konfigurasjonsfiler [92].

5.4.5 CircuitMaker PCB-designprogramvare

Circuitmaker er ett gratis åpenkilde PCB-designverktøy for å tegning av skjematikk og PCB-utlegg. Programvaren er utviklet av Altium og har derfor sterke paralleller knyttet til det profesjonelle designverktøyet; Altium Designer. I programvaren kan en hente ned komponenter fra online bibliotek ferdig for skjematikk og utlegg, som videre kan implementeres i eget design. Videre har programmet funksjoner for eksport av gerber-filer og drill-filer, samt kretssjekk og 3D-illustrasjon av ferdig design [93].

5.5 Biblioteker

5.5.1 spidev.h

spidev.h brukes for å sende og motta data over SPI på Raspberry Pi. Den setter opp en virtuell fil som kan skrives til og leses fra. All data som skal over SPI går gjennom denne filen.

5.5.2 sys/ioctl.h

sys/ioctl.h brukes for å endre underliggende parametre for spesielle filer. I dette tilfelle endres filen for sending/mottak over SPI /dev/spidev.0.0 fra biblioteket spidev.h. Mer spesifikt brukes den for å endre overføringshastigheten til fila og for å skrive/lese data fra den.

5.5.3 stdio / iostream

stdio blir brukt i denne oppgaven for std::cout funksjonen som viser tekst i kommandovinduet.

5.5.4 sys/socket.h

sys/socket.h brukes for nettverkskommunikasjon i programmet. Den inneholder det meste av nettverks funksjonene som, socket() setsockopt() listen() accept() bind() connect() recv() og send().

5.5.5 arpa/inet.h

inet.h inneholder htons() funksjonen som brukes for å konvertere data til nettverksbinær form og ntohns for motsatt funksjon.

5.5.6 signal.h

signal bliblioteket brukes for å ta inn ett signal fra brukeren under runtime. I denne oppgaven brukes det for å avslutte tjeneren på riktig måte mens tjeneren går.

5.5.7 chrono

chrono brukes i denne oppgaven for å ta tiden i mellom operasjoner.

5.5.8 string.h

string.h brukes for strcpy() funksjonen, for å kopiere en streng fra en variabel til en annen.

5.5.9 unistd.h

brukes for getopt() funksjonen for å kunne ta inn argumenter fra kommandolinjen. /

Kapittel 6

Hvordan bruke løsningen

I dette kapittelet beskrives hvordan man starter systemet, for å kunne kjøre bilen.

6.1 Starte systemet

6.1.1 Fysisk tilkobling av utstyr

Begge Raspberry Pi vil forsøke å koble seg opp på mobilnett når de skrus på. Dette krever at 5G-HAT allerede er slått på, derfor bør 5G-HAT tilkobles strøm før Raspberry Pi.

Pilot

USB-kabelen fra Logictech G29 ratt/pedaler kobles til på Raspberry Pi. På pilotsiden vil både 5G-HAT og Raspberry Pi 3B+ alltid være tilkoblet 230V nettspenning. De to tilhørende strømforsyningene med micro-USB-plugg kobles til Raspberry Pi og 5G-HAT.

Bil

Utstyret som er montert på bilen må kobles til tilhørende batteripakker. Dette gjelder både for Raspberry Pi 4B, 5G-HAT og ESC. Raspberry Pi 4B kobles til med USB-C, 5G-HAT med micro-USB. Disse kobles til hver sin powerbank. Til sist kobles ESC til den proprietære batteritilkoblingen som går ut fra bilens batteripakke.

6.1.2 Tilkobling, SSH

For å koble til Raspberry Pi over SSH brukes PuTTY [90].

For å koble til bil-pi:
IP 78.158.241.23
Port 22

For å koble til pilot-pi:
IP 78.158.240.251
Port 22

Brukernavn: pi
Passord: (sendes på mail)

6.1.3 Starte signal- /video overføring

Både video- og signal overføring må startes med en kommando på hver Raspberry Pi, fire kommandoer totalt. Signal overføring må startes på bil før pilot, video overføring må startes på pilot før bil.

Anbefalt rekkefølge av kommandoer er:

1. På bil:
   /home/pi/server/main

2. På pilot:
   /home/pi/client/main

3. På Pilot:
   gst-launch-1.0 udpsrc port=5004 retrieve-sender-address=false ! application/x-rtp, encoding-name=H264, payload=96, a-framerate=60 ! rtph264depay ! h264parse ! omxh264dec ! videoconvert ! fbdevsink

4. På bil:
   gst-launch-1.0 v4l2src do-timestamp=true extra-controls="a,h264_profile=0, video_bitrate_mode=1,h264_i_frame_period=1"! video/x-h264,framerate=60/1, width=720,height=480 ! h264parse ! rtph264pay ! udpsink host=78.158.240.251 sync=false

For å kjøre flere kommandoer på samme Raspberry Pi kan man enten åpne flere tilkoblinger i PuTTY, eller bruke kommandoen 'tmux' (terminal multiplexer) for å kunne kjøre flere kommandoer samtidig over en enkelt SSH tilkobling (ett vindu i PuTTY). Ved å bruke tmux vil kommandoer fortsette å kjøre selv om PuTTY lukkes.

tmux kommandoer:

- Start tmux: 'tmux'
- Splitt vinduet horisontalt: Ctrl+B, %
- Splitt vinduet vertikalt: Ctrl+B, "
- Velg vindu (markert i grønt): Ctrl+B, piltaster
- Koble til tmux session: 'tmux a'
- Lukk tmux delvindu: exit

[94][95]

6.1.4 Koble til 5G

Begge Raspberry Pi-ene vil prøve å koble seg opp på mobilnettet når de starter opp. Så lenge dette fungerer som det skal er det ikke nødvendig å gjøre noe av det som står beskrevet i dette delkapittelet.
Når Raspberry Pi slås på, starter en egen service som heter 'mobilnett'. Denne vil starte en tmux session (som også heter 'mobilnett'). I 'mobilnett' tmux session kjøres skriptet som kobler til 5G/4G. For at dette skal fungere må HAT være påskrudd og klar når Raspberry Pi starter.

For å koble til denne tmux session (må gjøres som bruker 'pi'):

```
tmux a -t mobilnett
```

Om automatisk tilkobling ikke fungerer er det også mulig å gjøre det manuelt, utenom 'mobilnett' service. Da må Raspberry Pi kobles til LAN, og man må finne lokal IP for å kunne logge inn over SSH.

```
/home/pi/SIM8200-M2_5G_HAT_code/Goonline/simcom-cm -p PIN -s mdatelenor.no
(eller 'systemctl start mobilnett')
```

Kodeliste 6.1: Kommando for å koble til mobilnett

Kodeliste 6.1 viser kommando for å koble til mobilnett ('PIN' byttes ut med SIM-kortets PIN-kode).

Når Raspberry Pi er tilkoblet mobilnettet benyttes nettverksgrensesnittet 'wwan0' for kommunikasjon over 5G/4G. 'wwan0' vil automatisk få prioritet over andre nettverksgrensesnitt ved tilkobling.
For å spesifisere om man skal koble seg til 5G eller 4G benyttes 'AT commands'. AT commands er en form for tekst kommandoer for å kommunisere med modem.

```
minicom -D /dev/ttyUSB2
```

Kodeliste 6.2: Kommando for å bruke AT commands.

For å benytte AT commands brukes programmet 'minicom', kodeliste 6.2 viser kommando for å opprette kommunikasjon med modemet.

```
AT+CPIN=PIN #Lås opp SIM-kort

AT+CNMP=38 #Bruk kun 4G RAN
AT+CNMP=71 #Bruk kun 5G RAN (fungerer ikke etter hensikten)
AT+CNMP=109 #Bruk både 4G/5G RAN

AT+CSQ #Sjekk signalstyrke

AT+CPSI? #Viser en linje for hver radio teknologi som benyttes (4G/5G)
```

Kodeliste 6.3: Eksempler på AT commands.

Kodeliste 6.3 viser eksempler på AT commands. For komplett liste over kommandoer, se referanse [96].

Kommandoer knyttet til 'mobilnett' service:

- systemctl status mobilnett (viser status)
- systemctl start mobilnett (starter service)
- systemctl stop mobilnett (stopper service)
- systemctl restart mobilnett (restarter service)
- systemctl enable mobilnett (aktiver automatisk oppstart)
- systemctl disable mobilnett (deaktiver automatisk oppstart)

Kapittel 7

Testing

Det ble gjennomført tester både på enkeltdeler og det ferdige systemet. Testene i rapporten er hovedsaklig gjort på det ferdige systemet.

7.1 Forsinkelse i nettverk

Det er to parameter som har stor påvirkning på hvor responsivt det oppleves å kjøre bilen. Den ene er forsinkelsen fra man gjør en bevegelse med rattet eller pedalene, til bilen reagerer. Den andre er forsinkelsen på video, fra kamera til skjerm.

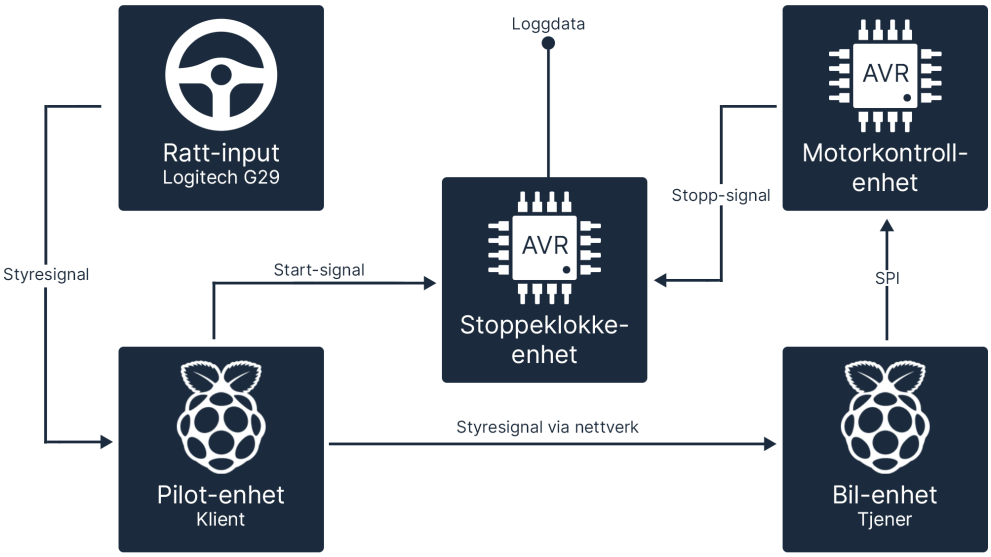
Begge disse forsinkelsene er avhengig av forsinkelsen i nettverket. For å teste dette kan man 'pinge' den ene Raspberry Pi fra den andre. Ping testen viser tiden det tar å sende data fra den ene Raspberry Pi til den andre, og tilbake igjen. Ved å dele denne tiden på to får man forsinkelsen én vei.

Videre kan det være interessant å se på hvor mye av forsinkelsen som stammer fra trådløs kommunikasjon mellom Raspberry Pi og 4G/5G basestasjon, og hvor mye som stammer fra at pakkene må rutes via kjernenett. Med utstyret som brukes i prosjektet er det ikke mulig å måle forsinkelse mellom Raspberry Pi og basestasjonen.

Det er derimot gjort forsøk på å anslå denne forsinkelsen. Ved å pinge IP adresser som er plassert i Oslo-området, fra en fiber forbindelse i Trondheim ser man forsinkelsen som stammer fra å sende data denne distansen over fiber. Om man pinger samme adresse fra Raspberry Pi tilkoblet 4G/5G ser man den totale forsinkelsen. Ved å trekke fra forsinkelsen som ble funnet i test over fiber, skal man i teorien sitte igjen med forsinkelsen som stammer fra å sende data trådløst over 4G/5G. Dette er langt fra noen perfekt test, men det gir grunnlag for å si noe om forsinkelsen som oppstår når data sendes trådløst over 4G/5G.

7.2 Styringssignal-forsinkelse

Ett viktig aspekt ved kjøretøyet er lav forsinkelse fra piloten roterer på rattet eller tråkker på pedalen, til det skjer en reaksjon på kjøretøyet. For å måle denne forsinkelsen ble det laget en test, veldig lik 'glass to glass' testen som er nevnt under video-testing. Arkitekturen på testen vises i figur 7.1.



Figur 7.1: Test av forsinkelse med brukergenerert signal.

Denne testen går ut på at når brukerinput fra ratt mottas over USB sendes det ut et digitalt signal fra Raspberry Pi enhetens GPIO-pinne.

Dette signalet sendes til en Arduino Uno, som fungerer som en stoppeklokkeenhet. Når signalet blir mottatt settes t0,og timing begynner. Deretter går brukerinputen fra Raspberry Pi (pilotenheten), over nettverket og inn på Raspberry Pi som sitter på kjøretøyet. Raspberry Pi-en som sitter på kjøretøyet sender så denne brukerinputen via SPI til motorkontrolleren, deretter hånterer motorkontrolleren dataen og genererer ett PWM-signal som endrer posisjon på hjulene.

Til slutt sendes ett logisk-høy signal ut fra motorkontrolleren og inn på stoppeklokkeenheten, som igjen setter slutt-tidspunktet, t1 for testen. Deretter beregnes delta_t, før resultatet skrives til seriell-monitoren.

For å kunne loggføre mange punkter, leses serielloverføringen av ved hjelp av en Raspberry Pi, loggeenhet. Denne loggenheten leser inn resultatene over USB-seriell fra hver enkelt punkt inn i en fil og lagres for videre analyse og plotting. For

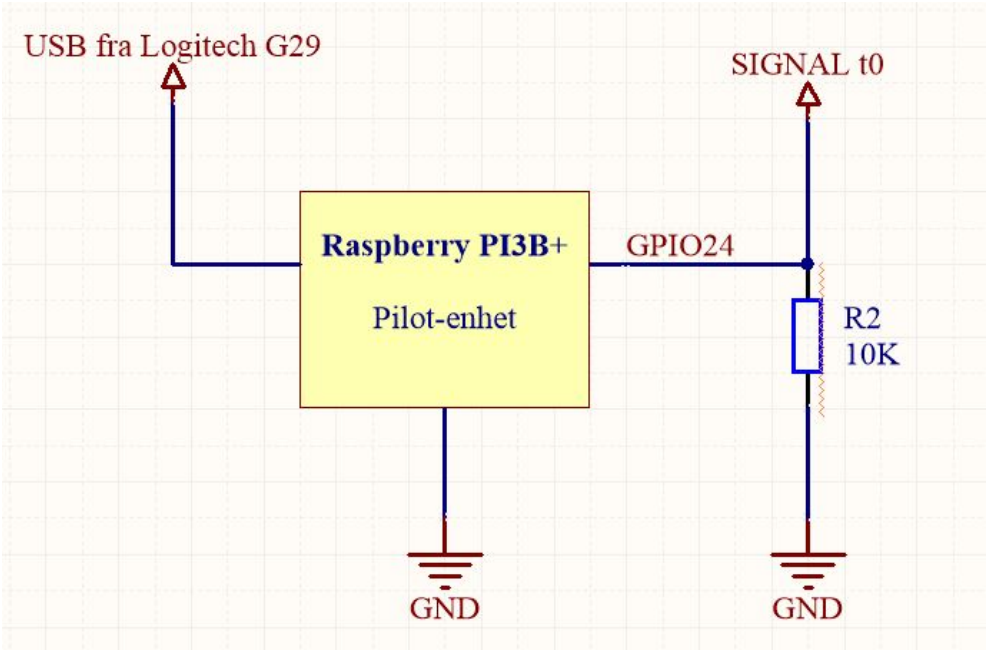
å fremstille dataene brukes numpy og matplotlib i Python til å finne medianverdi, minimums, maksimums og gjennomsnittsverdi. Deretter skrives disse resultatene ut, både som klartekst og graf. Grafen viser alle disse verdiene, samt ett samt løpende gjennomsnitt av forsinkelsen.

7.2.1 Teknisk beskrivelse av testen

I dette delkapittelet forklares teknisk design og kode som brukes i signaltesten.

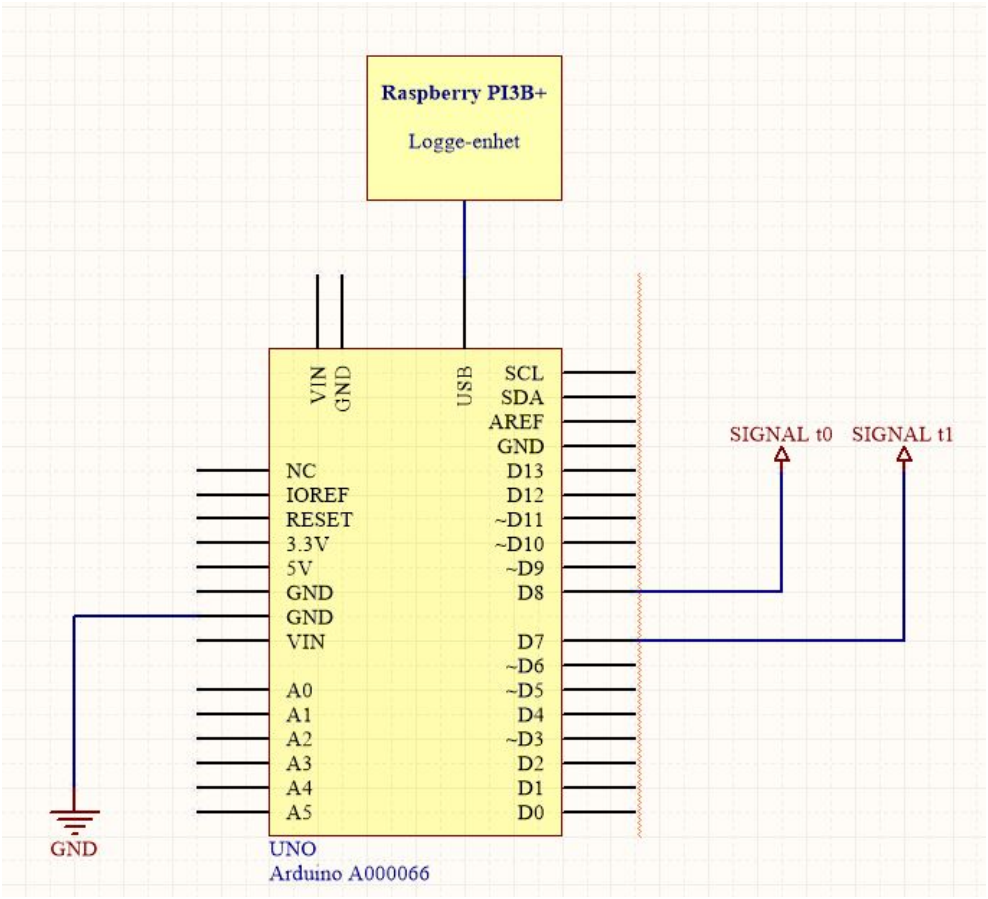
Krets

For testing ble en enkel krets benyttet realisert på breadboard ved bruk av THT-komponenter og dupontkabler.



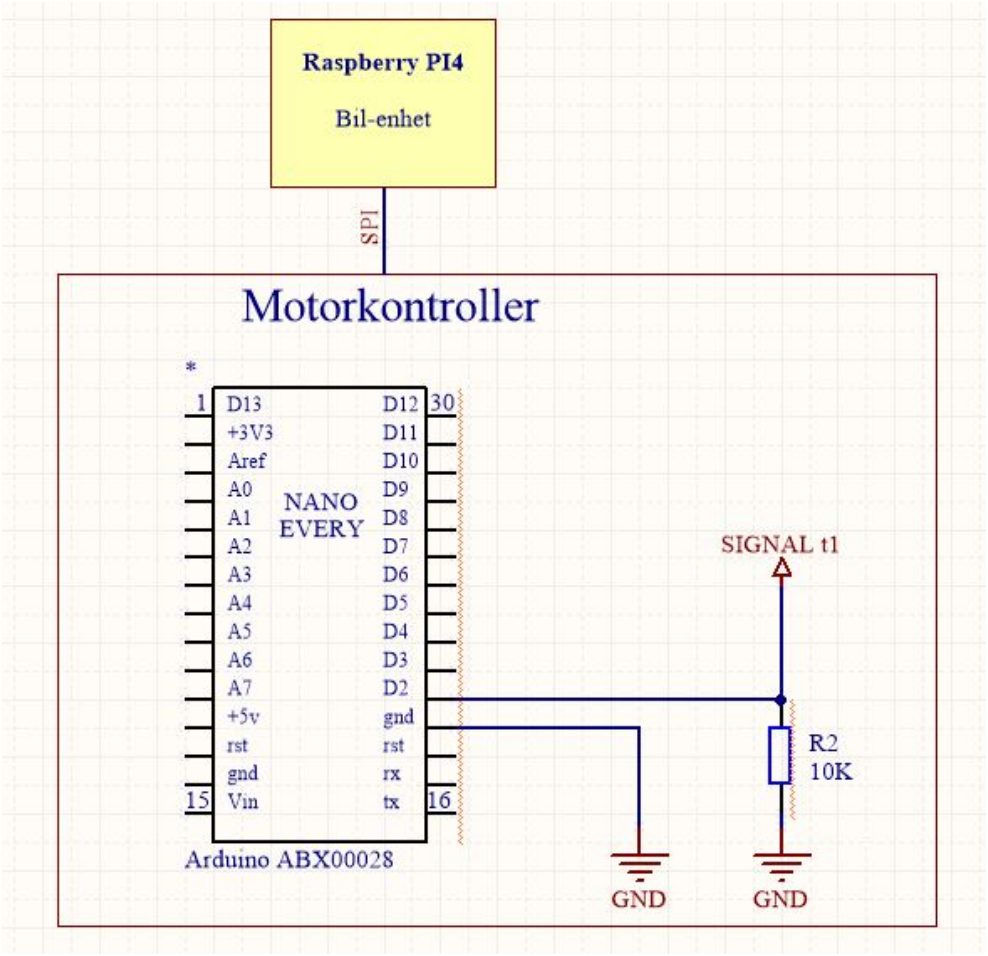
Figur 7.2: Skjematisk tegning av Raspberry Pi-signal.

Figur 7.2 viser hvordan startsignalet t0 sendes fra pilot-enheten. Det benyttes også en motstand parallell-koblet ned til jord for å unngå at signalet 'flyter'.



Figur 7.3: Skjematisk tegning av stoppeklokke-enheten.

Stoppeklokke-enheten mottar signalet fra pilot-enheten. Dette vises i figur 7.3.



Figur 7.4: Skjematisk tegning av motorkontroller-signal.

Motorkontrollenheten sender ut signalet via SIGNAL t1 for å sette t1 når styresignalet er mottatt over nettverk. Også her er det benyttet en 10k ohms motstand parallellkoblet ned til jord, for å dra SIGNAL t1 ned til 0V og dermed unngå uønskede flytverdier. Dette vises i figur 7.4.

Dette oppsummerer hvordan kretsen er realisert. I neste delkapittel vil oppbygging av kode bli forklart.

7.2.2 Arduino-kode

I dette delkapittelet forklares det hvordan forsinkelsesmålingen blir realisert i kode. Dette med uttdrag fra koden.

Deklarerte og definerte variabler

```
    / Latency testing /
//Definerer pinneutganger
#define signalPin 8   //digital input kobles til PILOT SIGNAL t0
#define sensPin 7     //digital input Kobles til BIL SIGNAL t1
//Deklarerer variabler
int sigRead;          //for avlesning av signalPin SIGNAL t0
int sensRead;         //for avlesning av sensPin SIGNAL t1
bool risingEdge = 0;  //Start tilstand 0
int lastSignal = 0;   //Start tilstand 0
int sensReady = 0;    //Start tilstand 0
//Variabler for måling av tid.
unsigned long startTime; //start tid. t0
unsigned long endTime;   //slutt tid. t1
unsigned long delta_t;   //delta_t, for tidsdifferanse.
```

Kodeliste 7.1: Variabler brukt i koden

I kodeliste 7.1 defineres variabler i koden for avlesning av pinner og timing. Avlest SIGNAL t0 er i koden sigRead, og sensRead svarer til avlest verdi fra SIGNAL t1 som beskrevet i kretsfigurene.

Videre deklareres stoppeklokkevariablene, startTime, endTime og delta_t. Disse benyttes til avlesning av millis()-funksjonen.

Arduino Setup() Arduino setup kjøres engang. I kodeliste 7.2 initialiseres innganger og seriell-kommunikasjonen.

```
void setup() {
  // put your setup code here, to run once:
  pinMode(signalPin,INPUT); //Setter pinne som inngang.
  pinMode(sensPin,INPUT);   //Setter pinne som inngang.
  Serial.begin(115200);     //Setter baud-rate til 115200. Høy baud for å unngå
      forsinkelser.
}
```

Kodeliste 7.2: Variabler brukt i koden.

Pinnene brukt i testen settes som innganger av den innebygde arduino-funksjonen pinMode. Pinmode setter de nødvendige registerene på atmega328p for avlesningsmodus. Deretter settes baudraten til seriell-kommunikasjonen til 115200 symboler pr sekund. Når dette er gjort, er programmet klart til å gå inn i den uendelige løkka, void loop().

void loop() I void-loop, vist i kodeliste 7.3, plasseres all kode som skal kjøres så lenge Arduinoen er tilkoblet strøm. I loopen avleses digitalpinnene og timing blir satt og til slutt kalkulert og skrevet til seriellmonitoren.

```
void loop() {
  sensRead = digitalRead(sensPin); //Leser av sensor-pinne 2. Sluttsignal for
      timer
  sigRead = digitalRead(signalPin); //Leser av sensor-pinne 1. Startsignal for
      timer

  //Håndtering av SIGNAL t0.
  if(sigRead !=lastSignal) //ved startsignal for timer
  {
    if((sigRead==1)&&(lastSignal==0)) //Sjekker om RISING EDGE.
    {
        //Serial.println("RISING EDGE"); utkommentert for å unngå og bremse
            programmet.
        risingEdge = 1; //Setter rising edge til 1. Boolsk True.
        startTime = millis();//micros();//millis();
        sensReady = 1; //Setter sensReady til 1. Boolsk True.

    }
    else
    {
      risingEdge = 0; //setter rising edge til 0. Boolsk false.
    }
    lastSignal = sigRead; //Mellomlagrer siste verdi for å sammenligne og
        sjekke RISING EDGE.
  }

  if((sensRead==1)&&(sensReady)) //ved sluttsignal for timer. SIGNAL t1.
  {
    endTime=millis();//micros();//Setter sluttid for måling. t1.
    sensReady=0; //tilbakestiller slik at sensor-avlesning kun skjer hvis
        rising-edge er satt.
    //Serial.println("FIRST SENSOR MESSURE"); utkommentert for å unngå og
        bremse programmet.

    delta_t = endTime - startTime; //beregner tidsdifferansen og dermed
        forsinkelsen i ms.
    Serial.print("latency:");Serial.println(delta_t); //skriver ut resultat av
        måling.
  }
}
```

Kodeliste 7.3: Variabler brukt i koden.

Loopen begynner ved å oppdatere verdiene i sensRead og sigRead med hjelp av den innebygde Arduino-funksjonen digitalRead. digitalRead leser av registrene knyttet til sensPin og SignalPin og returnerer den aktuelle verdien. Avlesningshastigheten til Arduino Uno er testet flere steder, deriblant av Arashi projects ved hjelp av micros()-funksjonen på Arduino, der resultatene viser 4-8 mikrosekunder for avlesning av pinner og [97].

En kan derfor vise at avlesningstiden er neglisjerbar , da testen måler i millisekunder.

Videre sjekkes det om sigRead er på stigende flanke 'rising edge', altså om signalet går fra logisk lav til logisk høy. Dette gjøres for å kun sette startsignalet engang så lenge sigRead er høy. Deretter settes startTimer-variabelen ved å lese av returnert verdi fra den innebygde arduino-funksjonen millis(). millis()-funksjonenen bruker en egen feilrettings-algoritme for å holde på nøyaktighet. 'From the analysis below the millis() timer will be continuously corrected and is not in error by more than 2ms'[98]. Algoritmen har ifølge beskrivelsen altså ett største avvik på ca. 2ms. Nøyaktigheten til millis() blir også da testens største nøyaktighetsbegensning.
Til sist i denne delen av koden settes sensReady til 1.

Videre sjekkes det om sensPinnen er høy, og videre om sensReady er satt til 1. SensReady brukes for å kun sette sluttiden endTime hvis startTime er satt. Videre beregnes tidsdifferansen delta_t, som er den totale forsinkelsen og resultatet. Resultatet skirves tilslutt til seriell-monitoren, hvor resultatet leses av logge-enheten, Raspberry Pi 3B+.

Kode på pilot-enheten Fra pilot-enheten sendes det ett signal ved pluss-knappetrykk på Logitech G29-rattet. Måling av tiden det tar fra G29-rattet trykkes på til det registreres på Raspberry Pi 3B+, pilot-enheten er under ett millisekund. Dette basert på Pollingraten på USB-HID enheter er oppgitt til 1000Hz (kilde til pollingrate HID), og videre bekreftet i målinger gjort i . Avlesning av ratt er satt til 2000Hz, altså det dobbelte av USB-hastigheten.
Samplingen beskrevet i den matematiske formelen:

$$f_{read} = Avlesningshastighet$$

$$f_{USB} = Pollinghastighet$$

$$f_{read} = 2 \times f_{USB}$$

Dermed skal all data i teorien bli samplet ofte nok til at ingen ny data ikke blir avlest.

Signalsendingen blir realisert ved bruk av wiringPi som er ett bibliotek for setting av GPIO-pinner på raspberry-Pi. Hastigheten til GPIO-pinnesetting ved bruk av wiringPi er godt dokumentert med bruk av oscilloskopet PicoScope 544B, med avlest maksimalhastighet på 22MHz ved generering av firkantpulser[99]. Med denne hastigheten er setting av høy/lav på GPIO-pinner nede på nano-sekunder, og derfor også neglesjerbart i testen.

```
/ WIRING PI: t0-signal -> latency-test. /
        wiringPiSetup () ; //Setter opp wiringPi.
        pinMode(5, OUTPUT); //GPIO24
```

Kodeliste 7.4: Uttdrag fra client-kode: Oppsett wiringPi.

Kodeutdraget i kodeliste 7.4 blir wiringPi satt opp og pinMode satt til utgang i henhold til wiringPi-dokumentasjonen [100].

Videre må signalet genereres. Dette signalet skal genereres når plussknappen blir trykt og satt til høy. Kodeliste 7.5 viser dette.

```
if (data[1] == MINUS_BTN_CODE || data[1] == PLUSS_BTN_CODE)
    {
        / Filterer slik at boolsk knappetrykk kun sendes engang ved endring. /
        if (preButtonValue!=data[0])
        {

                send(sock,&data, 2 sizeof(int), 0); //sender arrayet data.
                if((data[1]==PLUSS_BTN_CODE)&&data[0]==1)
                {
                digitalWrite(5,HIGH);//setter GPIO24 høy.
                usleep(0.01 main_microsecond); //holder GPIO24 høy i 10ms.
                digitalWrite(5,LOW); //Setter GPIO24 lavt igjen.
                cout<<"\r["<<delta_t<<"]  \t["<<"[SIGNAL GENERERT] TERMINAL
                    OUTPUT KLIKKER, MEN DEN KJORER..."<<"]\t[Value]:"<<data
                    [0]<<"  \t[Code]:\t"<< data[1]<<"        "<<std::flush;
                                    }
```

Kodeliste 7.5: Utdrag fra klient-kode: generering av signal

Dette realiseres i klient-koden når ett knappetrykk blir registert.Først sjekkes det om dataen ikke er lik forrige verdi, for å unngå å sende hele tiden. Deretter sjekkes det om knappetrykket var fra plussknappen, og at signalet er høyt og ikke lavt. Når disse kriteriene blir møtt, settes GPIO-pinne 24 høy ved hjelp av digitalWrite-funksjonen fra wiringPi-biblioteket. WiringPi bruker for ordens skyld andre pinne-navn, slik at wiringPi-pinne 5 svarer til GPIO-pinne 24. Pinnen holdes så høy i 10ms, før den igjen settes lav.

Kode på motorkontroller Signal t1 generert fra motorkontrolleren blir realisert ved å sette utgang D2 høy ved når knappetrykk på plussknappen er blitt registert mottatt av motorkontrolleren. Dette realiseres i kodeliste 7.6:

```
void handleData(long int code, long int value) {

  switch (code)
  {
      case PLUSS_BTN_CODE:
      gearing = 1;
      //Latency testing. sette pinne hoy. Setter lav igjen i hovedloop.
      digitalWrite(signal_pin,HIGH);
      Serial.println("Latency signal sendt.");
      break;
```

Kodeliste 7.6: Variabler brukt i koden

Hvis plussknappen er blitt trykket og dataen blir overført suksessfult vil case PLUSS_BTN_CODE bli utløst. Deretter settes signal_pin høy og signal t1 blir generert. Denne pinnen blir så satt lav etter en liten forsinkelse i hovedløkka i koden. For full forklaring av motorkoden kan dette leses i kapittelet teknisk design.

Nøyaktighet

Testens nøyaktighet blir noe nevnt i de foregående delkapittelene, og i dette delkapittelet vil alle unøyaktighetsleddene bli lagt frem og summert.

USB-forsinkelse Avlesningsforsinkelse knyttet til data-overføring fra Logitech G29 over USB er begrenset til USB-HID pollingrate. Denne er opplyst til 1000Hz[101]. og videre også testet til 1000Hz ved hjelp av avlesning i Python. Headerfilen som tar seg av avlesningen sampler med en hastighet på 2000hz. Forsinkelse knyttet til USB-avlesningen settes derfor til 1ms.

GPIO-forsinkelse Setting av GPIO-pinner er oppgitt til en maksimal-rate på 22Mhz. Dette er så raskt at denne forsinkelsen neglisjeres[99].

Arduino-forsinkelse Det er knyttet tre forskjellige typer forsinkelser til Arduino. Den første er digitalWrite-funksjonen. Flere nettsider har testet hastigheten på digitalWrite og resultater fra for eksempel Robotics Backend viser at forsinkelsen er 3.4us [102].
Den andre forsinkelsen er knyttet til digitalRead-funksjonen. Også denne er testet flere steder, deriblant Arashi Projects oppgir forsinkelsen til å ligge på mellom 4 og 8us [97].

Den siste forsinkelsen knyttet til Arduinoen er nøyaktigheten til den innebygde funksjonen millis(). Ifølge forklaring av algoritmen på best-microcontroller-projects som brukes for å lage millis()-returndataen og samtidig rette feilverdier, skal ikke feilmarginen kunne bli større enn maksimalt 2ms. Videre opplyses det at micros-funksjonen har enda bedre nøyaktighet[98].

Total feilmargin Basert på de forskjellige test-leddenes feilmarginer og teoretiske forsinkelser kan en finne en total feilmargin for testen.

$$t_{USB} = 1\ ms$$

$$t_{millis} = 2\ ms$$

$$t_{total=t_{USB}+t_{millis}}$$

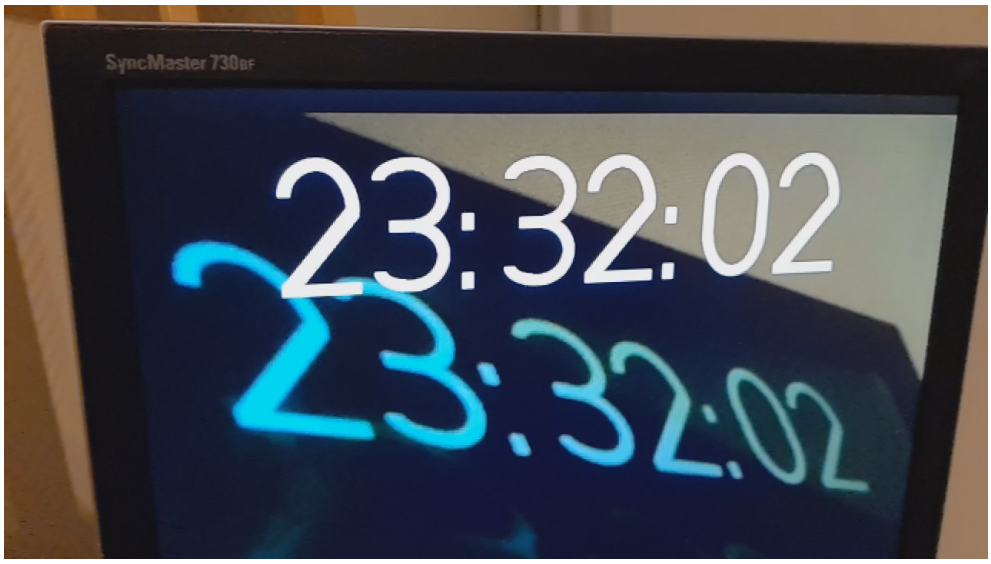
Ved å summere de største avvikene som kan oppstå med størst forsinkelse vil summen være størst mulig avvik. Basert på dette har vi ett totalt avvik på +-3ms i testen. Dette stemmer også godt med testing av uttstyret, hvor resultatene ligger mellom 0-3ms ved forsinkelse tilnærmet null. Testen som ble utført bruker signalgeneratoren til å generere t0 og t1 signalene og se på differansen.

7.3 Videoforsinkelse

Tidlige tester

Under det tidlige arbeidet med video ble det aller meste av testing gjennomført mellom to Raspberry Pi 3B+, over kablet LAN. Dette var for å utelukke forsinkelse over 4G/5G, og legge fokus på så rask video enkoding/dekoding som mulig.

Det første fungerende videosystemet bestod av 'raspivid' for å enkode og sende video, og VLC for å spille av. Forsinkelse ble anslått på øyemål ved å bevege en hånd forran kameraet, og se hvor lang tid det tok før det vistes på skjermen. Både lavere oppløsning og høyere bildefrekvens så ut til å gi lavere forsinkelse.



Figur 7.5: Test av forsinkelse, raspivid/VLC.

Når video fungerte begynte målingen av mer nøyaktig forsinkelse. Dette ble i begynnelsen gjort ved å skrive tidspunkt på videoen, for så å peke kameraet på skjermen. Så ble skjermen filmet med mobiltelefonen, i 120 eller 240 fps, som vist i figur 7.5. Ved å gå gjennom videoen bilde for bilde kunne fosinkelsen fra kamera til skjerm regnes ut. Om telefonen filmer i 240 fps, vil den ta et nytt bilde hvert 4,17 ms.

$\frac{1000}{240} = 4,17ms$
Hvis det ene klokkeslettet endres 24 bilder etter det andre, kan man enkelt regne ut at forsinkelsen er ca. 100 ms

$24 \times 4,17 = 100ms$

Dette gir noen feilmarginer, men det ga en god pekepin på forsinkelsen i systemet.

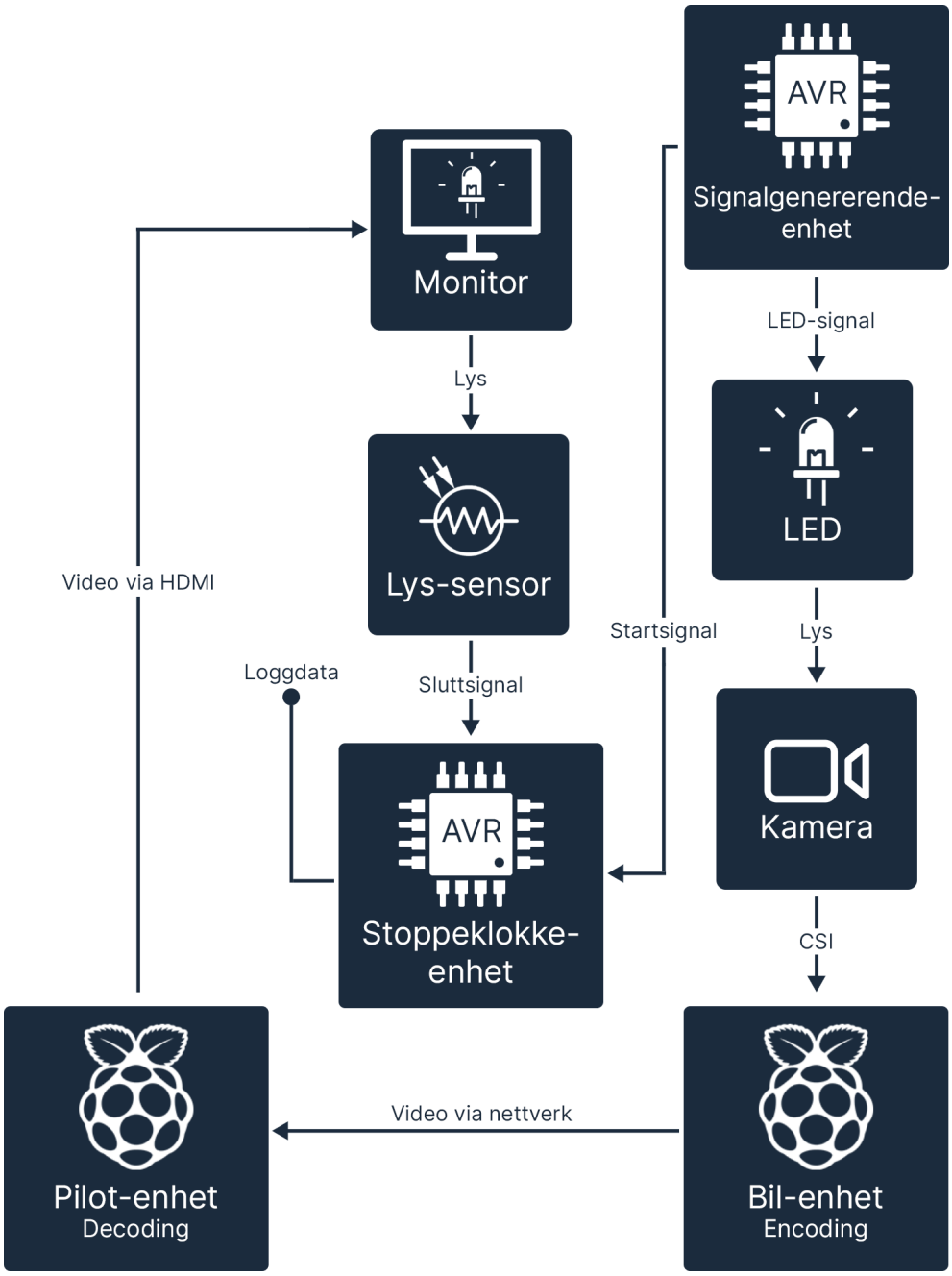
Det ble også testet ved å filme blinkende lys istedenfor klokke på skjermen.

Figur 7.6: Test av forsinkelse, GStreamer.

Underveis i prosessen ble raspivid/VLC byttet ut med GStreamer. For å teste forsinkelsen i GStreamer ble det brukt en lignende testmetodikk som tidligere. En tidskode ble skrevet på videoen, men i motsetning til raspivid kan man i GStreamer også skrive på millisekund.
Denne testen så ut til å gi ganske nøyaktig 100 ms i forsinkelse, som vist i figur 7.6. Testen er ikke helt nøyaktig, blant annet inkluderer den ikke forsinkelsen i skjermen (fra signal inn over HDMI til bilde ut). Vi la merke til at ikke alle 3 siffrene i telleren for millisekunder oppdaterte seg like ofte, så det er tydelig at verdiene som vises på skjermen ikke er 100% korrekt. Det er også usikkert hvordan det å skrive på denne teksten påvirker forsinkelsen.

7.3.1 Nøyaktig testing: 'glass-to-glass'

For nøyaktig testing av total forsinkelse i video-overføringen lagde vi en 'glass-to-glass' test. Strukturen på testen vises i figur 7.7.



Figur 7.7: Prinsipp for nøyaktig test av forsinkelse.

Denne testmetoden går ut på å utsette kamerasensoren på sendersiden for ett sterkt lys, for så å måle hvor lang tid det tar før dette lyset registreres på skjermen på mottakersiden. Dette times ved bruk av tidsdifferanse,

$$delta\_t = t1 - t0$$

teller t0 blir satt når lyset blir tent, teller t1 teller helt frem til skjermen lyser opp. delta_t gir oss da tidsdifferansen og dermed også forsinkelsen. Teknisk sett ble dette utført ved hjelp av to Arduinoer.

Beskrivelse av testmetoden

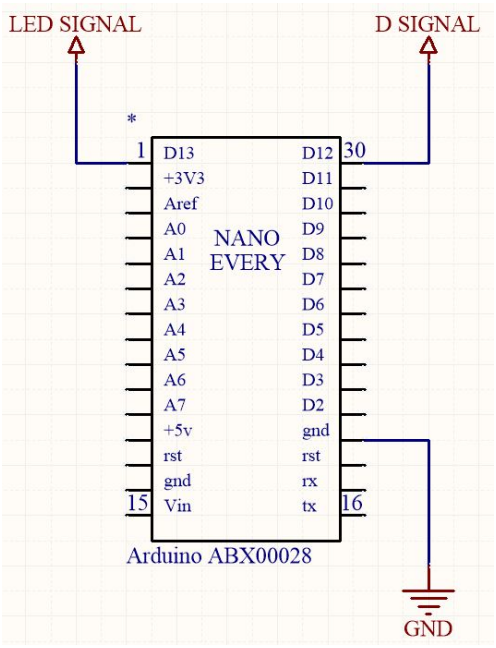
Arduino A, som er en Arduino Uno med en atmega328p mikrokontroller fungerte som stoppeklokkeenhet. Arduino B, som er en Arduino Nano Every med en ATmega4809 mikrokontroller fungerte som signalgenerator.
Arduino B sendte ut to elektrisk 5 volts signal samtidig, det ene signalet tenner LED fremfor kameralinsen og det andre signalet sendes til Arduino A. Dette setter starttidspunkt på timingen, t0.

Videre står det en sensor festet på skjermen bestående av en photoresistor-spenningsdeler. Når skjermen lyser opp går spenningen fra sensoren opp, dette avleses analogt av Arduino A ved hjelp av den innebygde ADC-en, som igjen setter slutt-tidspunktet, t1 for testen. Deretter beregnes delta_t, før resultatet skrives til seriell-monitoren.

For å kunne loggføre mange punkter, leses serielloverføringen av ved hjelp av en Raspberry Pi, loggeenhet. Denne loggenheten leser inn resultatene over USB-seriell fra hver enkelt punkt inn i en fil og lagrer dette for videre analyse og plotting. For å fremstille dataene brukesbibliotekene og glsmatplotlib i Python til å finne medianverdi, minimums, maksimums og gjennomsnittsverdi. Deretter skrives disse resultatene ut, både som klartekst og graf. Grafen viser alle disse verdiene, samt ett samt løpende gjennomsnitt av forsinkelsen.

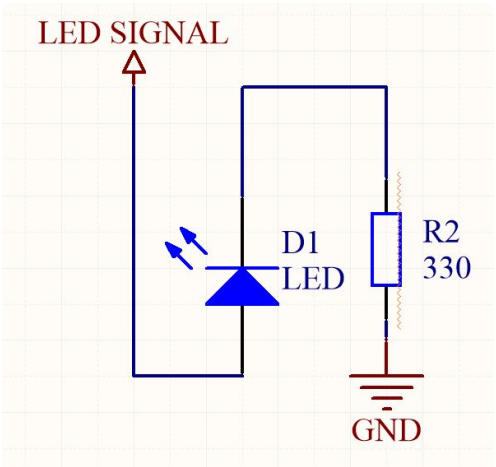
Krets og kode

Kretsen er realisert igjennom en relativt enkel krets. Første del av kretsen er den signalgenererende Arduino B som vist i figur 7.8.



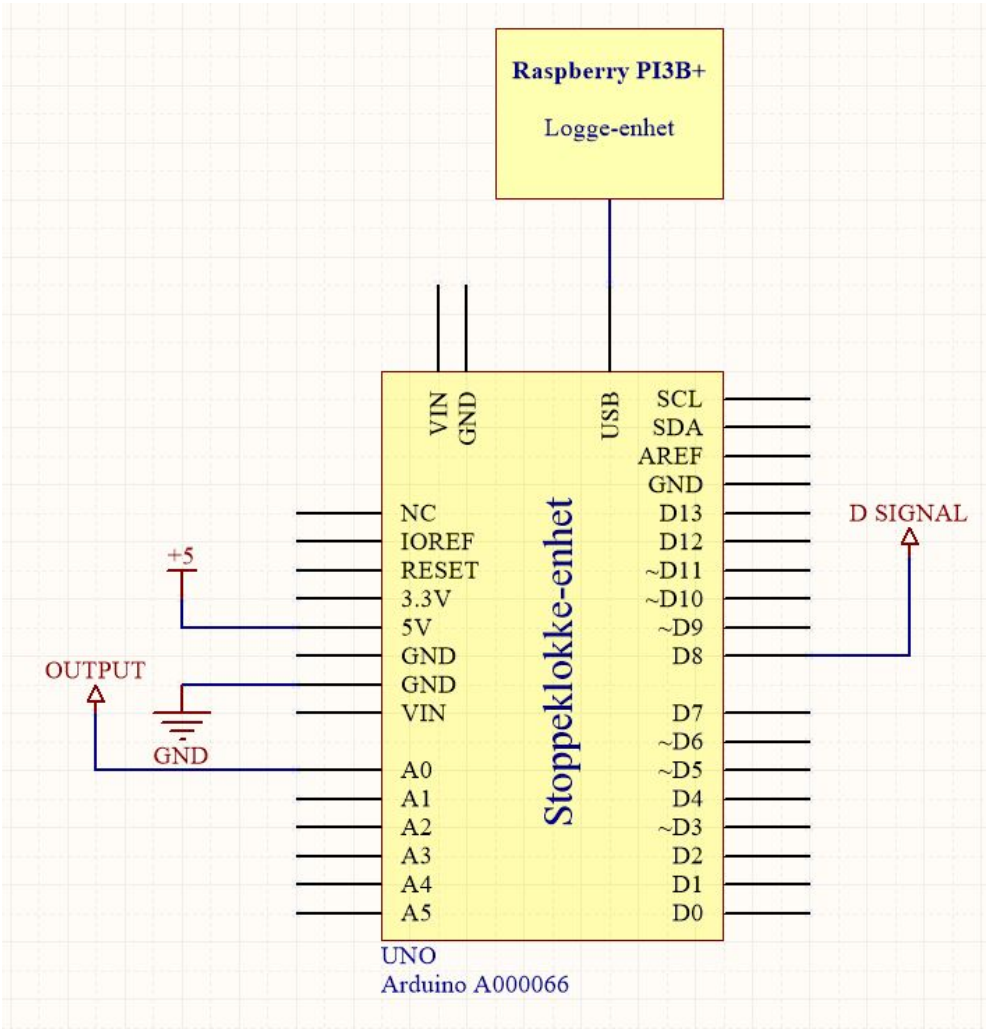
Figur 7.8: Arduino B, signalgenerator-mikrokontroller.

Denne genrerer to utgangssignaler på 5V, LED SIGNAL og D SIGNAL.
Neste del av kretsen er lysdioden som sitter inne i en lystett hette. Dette vises i figur 7.8.



Figur 7.9: LED-krets for genererings av lyssignal.

Som illustrert i kretstegningen er lysdioden koblet i serie med en strøm-begrensende motstand. Lysdioden blir styrt av LED SIGNAL, fra den signalgenererende enheten, Arduino B.
Videre i kretsen er stoppeklokkeenehten, Arduino A, som vist i figur 7.9. Dette er den lyttende enheten som setter start og sluttidspunkt, samt kalkulerer forsinkelsen og skriver denne ut via seriell-porten over til loggenheten, Pi.



Figur 7.10: Arduino-A, stoppeklokke-enhet.

D SIGNAL tas inn på inngang D8, som igjen setter starttidspunktet for timingen. Deretter står den analoge inngangen å måler spenning inn fra OUTPUT ut fra lyssensorkretsen.
Siste ledd i kretsen er lyssensorkretsen.
Kalibrerings-kode, se vedlegg F.

Nøyaktighet

Det er flere ledd i testen som teoretisk sett kan påvirke nøyaktigheten av testen.

Treghet i ADC

Tiden ADC bruker på konvertering er bestemt av klokkefrekvensen og prescaleren. En Atmega328P har en klokkefrekvens på 16MHz og standard prescaler for ADC på chippen er 128[7]. Dette gir en frekvens på 125kHz. Som vist i databladet for Atmega328P (s. 249) tar en standard ADC konversjon 13 klokkesykluser, altså $13 \times \frac{1}{125kHz} = 0.000104s = 0.104ms = 104us$. Avlesningshastigheten til glsadc-en kan derfor ses bort fra, da testen oppgir resultater i millisekunder, med nøyaktighet ned til nærmeste hele millisekund.

Videre har vi også treghet i digitalwrite og digitalread-funksjonene, men disse kan trygt neglisjeres. Nøyaktigheten til millis-funksjonen er gitt til å være maksimalt 2 ms. Alle disse funksjonenes nøyaktighet er beskrevet i kapittelet Test av signalforsinkelse under delkapittel nøyaktighet.
Utover disse nøyaktighetene er det forsinkelse knyttet til tenning av LED. Etter gjennomgang av flere datablader, ser det ut til at forsinkelsen ikke blir oppgitt. Måling gjort viser at tiden det tar å tenne en hvit LED er nede i nano-sekundersområdet [103]. En kan dermed neglisjere også denne tiden i testen.

Total unøyaktighet Basert på de forskjellige test-leddenes feilmarginer og teoretiske forsinkelser kan en finne en total feilmargin for testen.

$$t_{adc} = 0.104\ ms$$

$$t_{millis}k = 2\ ms$$

$$t_{total} = t_{adc} + t_{millis}$$

$$t_{total} = 2.104\ ms$$

Ved å summere de største avvikene som kan oppstå med størst forsinkelse vil summen være størst mulig avvik. Basert på dette har en ett totalt avvik på +-2 ms i testen.

Test av video med pakketap

Det ble også gjort en test av hvordan video vil oppføre seg med pakketap. Første forsøk ble gjort med 5% pakketap, man kunne da ikke merke noen forskjell med det blotte øyet. Deretter ble pakketap gradvis justert helt opp til 90%. Selv med 90% pakketap fungerer videoen overraskende bra, det er ikke noen merkbar ekstra forsinkelse med det blotte øyet. I løpet av minuttene videoen ble testet hendte det et par ganger at videoen hakket litt, videoen frøs også en gang i noen sekunder, før den fortsatte.

```
tc qdisc add dev eth0 root netem loss 5%
tc qdisc change dev eth0 root netem loss 90%
tc qdisc change dev eth0 root netem loss 0%
```

Kodeliste 7.7: Sette pakketap på nettverksgrensesnitt

Kodeliste 7.7 viser kommandoer for å sette pakketap på eth0 nettverksgrensesnitt.

Figur 7.11: Lilla striper i video.

Figur 7.12: Lilla felt i video.

Figur 7.11 og 7.12 viser feil som kan oppstå i video bildet. Det virker som det har sammenheng med 4G/5G dekning, men det er usikkert nøyaktig hva det skyldes. I dette tilfellet varte problemet i 2 minutter og 20 sekunder, over en distanse på mange hundre meter. Både i starten og slutten av denne perioden frøs bildet i over ett sekund, dette trolig på grunn av at bilen byttet basestasjon. Dette var på en kjøretur på 13 minutter, over en distanse på 2 km. Under denne kjøreturen ble ikke dette problemet observert forutenom akkurat på denne strekningen.

7.4 Subjektiv langkjøringstest

Det har har blitt gjennomført en langkjøringstest fra Tyholttårnet til NTNU Gløshaugen. Distansen på ruten som ble kjørt var på 2,0 km.

Kapittel 8

Resultater

8.1 Forsinkelse i nettverk

Forsinkelse	4G	5G
Totalt	52 ms	27 ms
RAN	2 $\times 22ms$	2 $\times 9.5ms$
Fiber	2 $\times 4ms$	2 $\times 4ms$

Tabell 8.1: Nettverksforsinkelse fra én Raspberry Pi til den andre.

Tabell 8.1 viser gjennomsnittlig forsinkelsen én vei, fra en Raspberry Pi til en annen. 'RAN' er estimert forsinkelse som stammer fra trådløst hopp mellom basestasjon/Raspberry Pi. 'Fiber' er estimert forsinkelse som oppstår fordi pakkene må gå innom kjernenett, tur/retur Oslo-Trondheim på fiber.
Dette er basert på et lite antall tester, tallene er ikke nødvendigvis representative for kvaliteten i nettverket over tid.

Figur 8.1: Signalvei fra én Raspberry Pi til den andre.

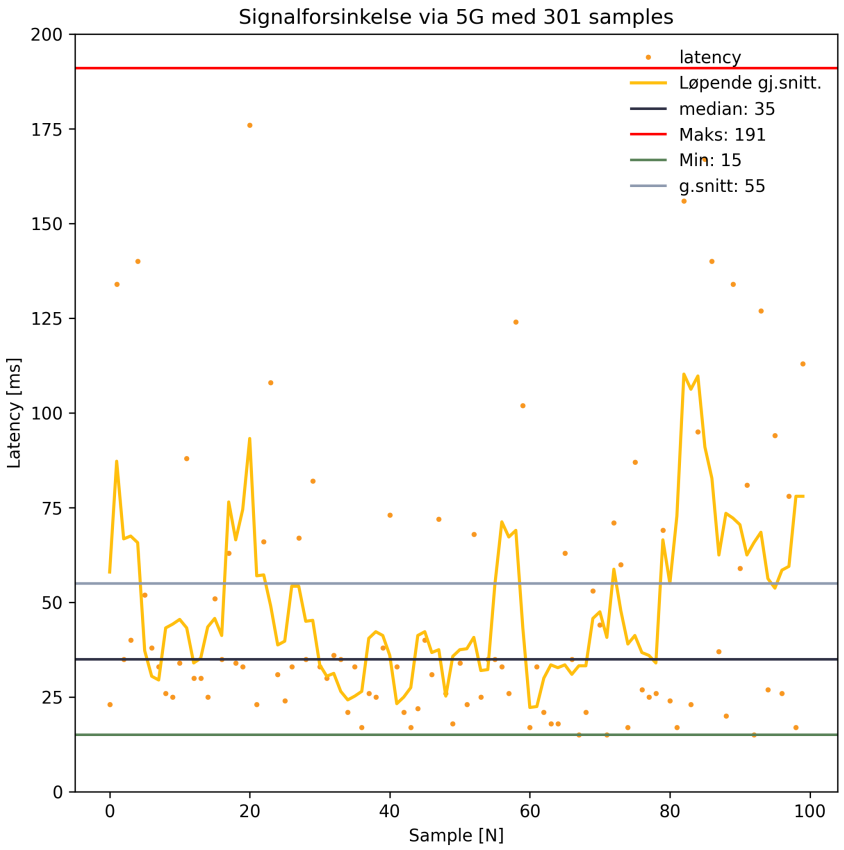
Figur 8.1 illustrerer signalveien. Pilen markert med '5G' er trådløs kommunikasjon mellom Raspberry Pi og basestasjon. Pilen markert med 'Fiber' er kommunikasjon mellom basestasjon og mobilnettets kjernenett.

8.2 Styringssignal-forsinkelse

Forsinkelse	4G	5G	LAN
Gjennomsnitt	66 ms	55 ms	0-1 ms
Median	39 ms	35 ms	0-1 ms
Min	20 ms	15 ms	0-1 ms
Maks	361 ms	191 ms	0-1 ms

Tabell 8.2: Signalforsinkelse fra ratt tilkoblet pilot-pi, til bil-pi.

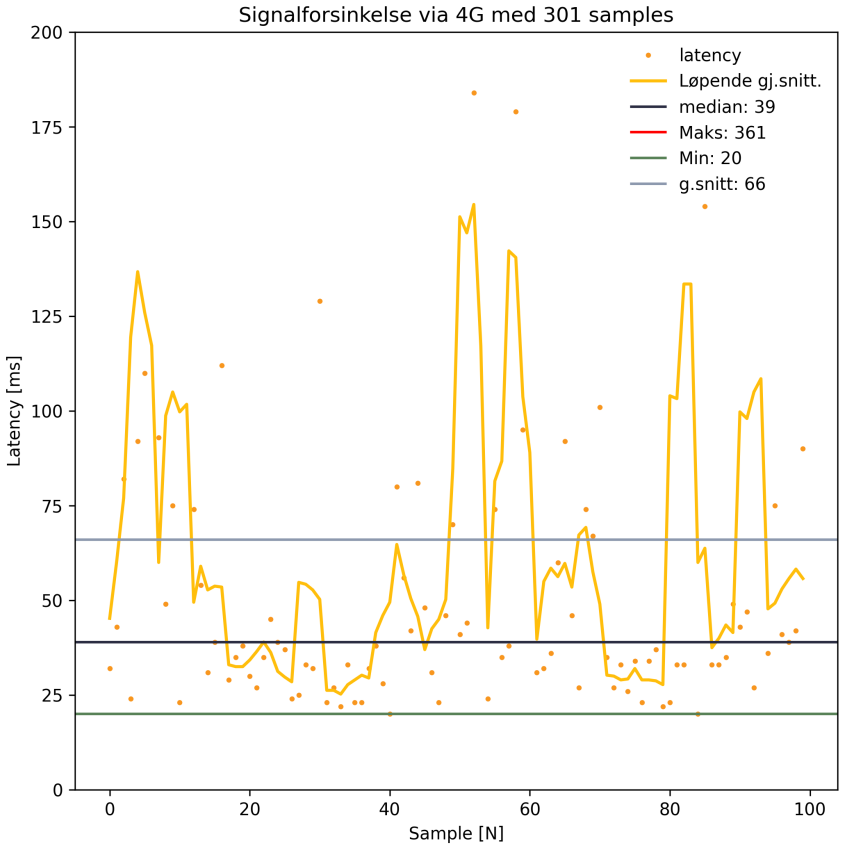
Tabell 8.2 viser signalforsinkelse som ble målt over forskjellige nettverk.



Figur 8.2: Grafisk fremstilling av signaltest over 5G.

Figur 8.2 viser resultatet av test av styresignal-forsinkelse over 5G. De gule prikkene som er spredt på diagrammet er ett enkelt testresultat. Gul linje er løpende

gjennomsnitt av hvert enkelt testresultat.



Figur 8.3: Grafisk fremstilling av signaltest over 4G.

Figur 8.3 viser resultatet av test av styresignal-forsinkelse over 4G. De gule prikkene som er spredt på diagrammet er ett enkelt testresultat. Gul linje er løpende gjennomsnitt av hvert enkelt testresultat.

8.3 Videoforsinkelse

Tabell 8.3 viser videoforsinkelse som ble målt over forskjellige nettverk (under uformell testing har 4G ofte hatt høyere forsinkelse enn dette). Testene er utført med Raspberry Pi 4B på bil, Raspberry Pi 3B+ på pilot-side. Testing med 3B+ i begge ender ga samme resultat.

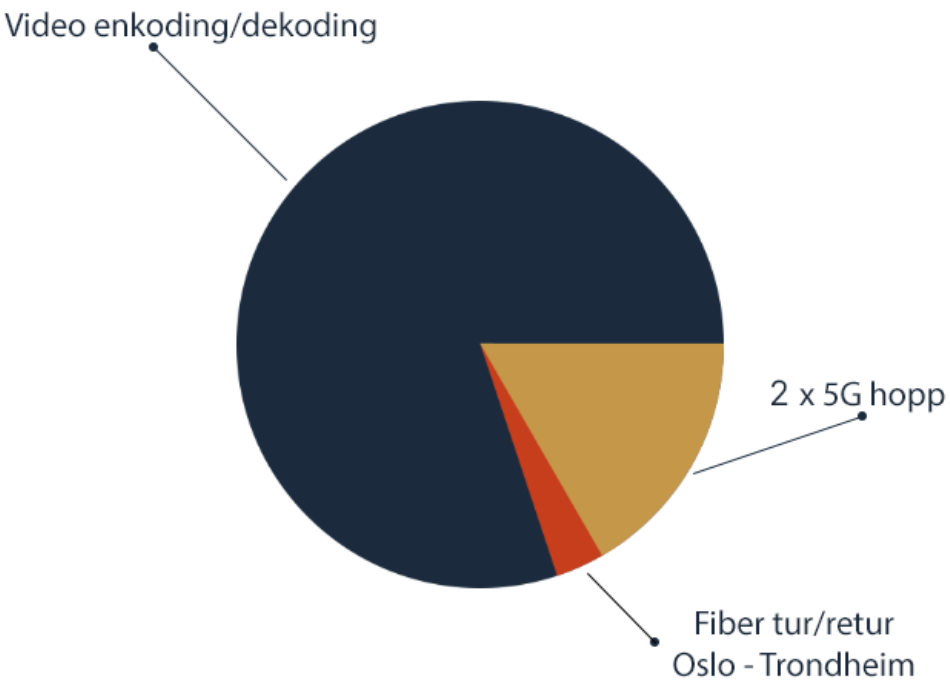
Forsinkelse	4G	5G	LAN
Gjennomsnitt	133 ms	125 ms	93 ms
Median	133 ms	123 ms	92 ms
Min	84 ms	78 ms	52 ms
Maks	195 ms	208 ms	148 ms

Tabell 8.3: Videoforsinkelse, forskjellige nettverk.

```
Oppløsning: 720x480
Bildefrekvens: 60
Bitrate: fast (10 Mb/s)
Enkoder: v4l2src
Dekoder: omx_h264_dec
```

Kodeliste 8.1: Testparameter (tabell 8.3)

Kodeliste 8.1 viser testparameter i GStreamer som ble brukt under testingen som ga resultatene i tabell 8.3.



Figur 8.4: Kilder til videoforsinkelse.

Figur 8.4 viser hva som bidrar til videoforsinkelsen, med utgangspunkt i 5G, tabell 8.3. 'Video enkoding/dekoding' inkluderer også forsinkelse i skjermen.

Forsinkelse	Pi 3B+	Pi 4B
Gjennomsnitt	145 ms	254 ms
Median	142 ms	252 ms
Min	100 ms	217 ms
Maks	211 ms	299 ms

Tabell 8.4: Videoforsinkelse, forskjellig Raspberry Pi (30 fps, LAN)

Tabell 8.4 viser videoforsinkelse (over LAN) ved forskjellige Raspberry Pi modeller på pilot-side. Grunnen til at forsinkelsen er høyere på den kraftigere Raspberry Pi 4 er at det benyttes et annet API for dekoding av video på pilot-siden. På Raspberry Pi 3 benyttes GStreamer modulen 'omxh264dec', som er bygget på OpenMAX API [104]. Dette API støttes ikke på Raspberry Pi 4. Dermed må modulen 'avdec_h264_mmal' (eller 'v4l2h264dec') brukes, disse har dårligere ytelse.

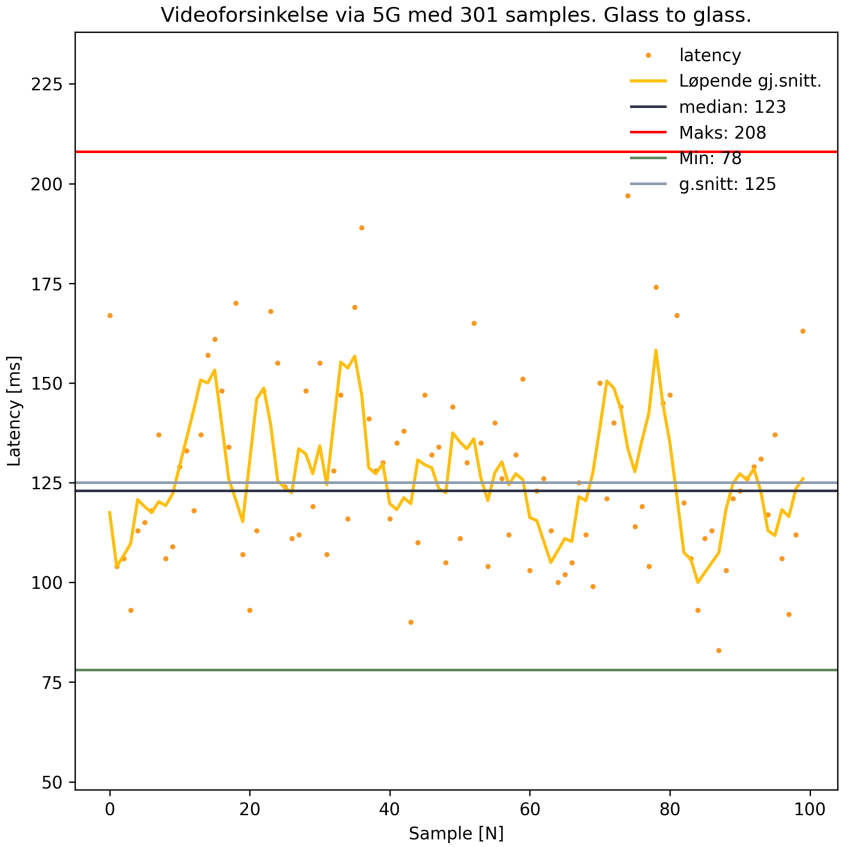
```
Oppløsning: 720x480
Bildefrekvens: 30 (60 fungerer ikke med avdec_h264_mmal)
Bitrate: fast (10 Mb/s)
Enkoder: v4l2src
Dekoder: omx_h264_dec (Pi 3) / avdec_h264_mmal (Pi 4)
```

Kodeliste 8.2: Testparameter (tabell 8.4)

Kodeliste 8.2 viser testparameter i GStreamer som ble brukt under testingen som ga resultatene i tabell 8.4.
I tillegg til de nevnte testene ble det også utført testing hvor én og én variabel ble endret, for å finne hvilken påvirkning hver enkelt variabel har på videoforsinkelsen.

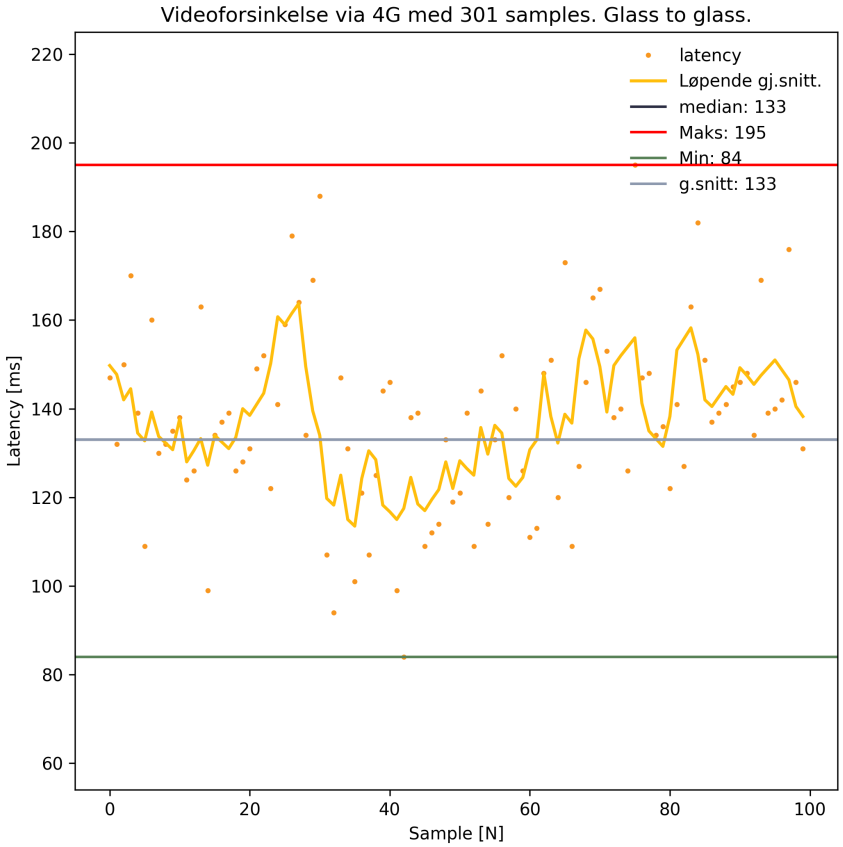
Listen under viser de mest sentrale variablene som ble testet, med resultatet som ga lavest forsinkelse.

- Bildefrekvens: 60 FPS
- Bitrate: Har ingen påvirkning (5/10/20 Mbps ble testet)
- Flytende/fast bitrate: Fast bitrate
- Dekoder: omxh264dec (ikke tilgjengelig på Pi 4)



Figur 8.5: Grafisk fremstilling av videotest over 5G.

Figur 8.5 viser resultatet av test av video-forsinkelse over 5G. De gule prikkene som er spredt på diagrammet er ett enkelt testresultat. Gul linje er løpende gjennomsnitt av hvert enkelt testresultat.



Figur 8.6: Grafisk fremstilling av videotest over 4G.

Figur 8.6 viser resultatet av test av video-forsinkelse over 4G. De gule prikkene som er spredt på diagrammet er ett enkelt testresultat. Gul linje er løpende gjennomsnitt av hvert enkelt testresultat.

8.4 Subjektiv langkjøringstest

Langkjøringstesten ble utført ved å kjøre på trafikkert fortau og vei, fra Tyholttårnet til NTNU - Gløshaugen i Trondheim. En strekning på totalt 2 km. Det gir en gjennomsnittshastighet på $\frac{2km}{\frac{13}{60}min} = 9.23\frac{km}{t}$. Piloten hadde følelse av god kontroll på kjøretøyet under hele testen, med få video- eller signal-forstyrrelser.

8.5 Maskinvare

Figur 8.7: Maskinvare - Endelig løsning på bil

Figur 8.7 viser den ferdigmonterte bilen.

Bil-siden	Pilot-siden
SIM8200EA-M2 5G HAT	SIM8200EA-M2 5G HAT
Raspberry Pi 4B	Raspberry Pi 3B+
Arduino Nano Every	Logitech G29 Ratt og pedaler
Egenutviklet kretskort	Playseat Challenge Sammenleggbar stol
Powerbank, Li-Po, 10Ah, Ansmann	BenQ 27” skjerm GL2780
Traxxas Slash 1/10 2WD radiostyrt bil	

Figur 8.8: Maskinvare - Pilot.

Figur 8.8 viser oppsett av ratt pedaler, racingstol og skjerm.

Kapittel 9

Diskusjon

9.1 Tanker rundt resultatene

Forsinkelse i nettverk

Testing av nettverksforsinkelse viser at hver gang data sendes trådløst mellom brukerterminal/basestasjon over 4G er det en gjennomsnittlig forsinkelse på i overkant av 20 ms. I dagens 5G-nett er denne forsinkelsen gjennomsnittlig rett under 10 ms. Det er også verdt å nevne at forsinkelsen over 5G er mer stabil enn over 4G. Det vil si at enkelte pakker sendt over 4G kan ha forsinkelse på flere hundre millisekunder, mens så høy forsinkelse sjeldnere blir observert over 5G.

Styresignal-forsinkelse

Testing av styresignal-forsinkelse viser at over lokalt nettverk er forsinkelsen tilnærmet 0 ms. Om man sammenligner median på 4G/5G ser det ikke ut til at det er veldig store forskjeller. Om man ser på gjennomsnittlig forsinkelse er forskjellen større. Dette er på grunn av at forsinkelsen over 5G er mer stabil enn forsinkelsen over 4G.

Videoforsinkelse

Testing av videoforsinkelse viser at forskjellen i forsinkelse på video sendt over 4G/5G ikke er stor. Subjektivt merker man større forskjell mellom 4G/5G enn det man kanskje skulle tro på bakgrunn av tallene i tabell 8.3. Video over 5G oppleves som mer stabil enn video over 4G. Dette er trolig et resultat av mer stabil forsinkelse over 5G.

Internt i gruppen var det diskusjoner om hvorvidt tallene for 4G, i tabell 8.3 kunne være riktige, eller om det var feil i loggdata. Det ble vurdert om man skulle forkaste disse resultatene og gjøre nye målinger. Det ble funnet at tallene er korrekte, dermed ble det ikke gjort nye målinger. Tallene i tabell 8.3 er basert på ca.

300 målinger, over en periode på 10-20 minutter. Det er mulig at flere tester, på andre tidspunkt hadde gitt større forskjell mellom 4G og 5G.

9.2 Teknisk design

I dette delkapittelet reflekteres det rundt de tekniske løsningene som ble valgt og utviklingsprosessen bak disse.

9.2.1 Valg av ettkortsdatamaskin

Raspberry Pi ble brukt som selve ryggraden i prosjektet. Ettersom Raspberry Pi er en liten datamaskin med forholdsvis mye kraft for sin størrelse og strømtrekk fungerte den godt i prosjektet. Det er også tilgang på GPIO-pinner.

På Raspberry Pi kjører Linux som er et operativsystem med åpen kildekode som egner seg godt til utvikling av lignende systemer. Dette har ført til at det har vært enkelt å finne ressurser som kan bidra til en raskere og bedre utvikling av prosjektet.

Linux bruker evdev til kommunikasjon med eksternt utstyr, noe som har gjør det enkelt å lese av data fra ratt og pedal. Linux har også mange innebygde biblioteker for nettverkskommunikasjon, samt GPIO på Raspberry Pi som gir mulighet for SPI-kommunikasjon med ekstern maskinvare. Svakheten med GPIO pinnene til Raspberry Pi er at de har dårligere støtte for hardware-PWM enn Arduino. Spenningsnivået på PWM skal også være 5V for å styre ESC og styringsservo. Raspberry Pi har 3,3V PWM signaler så en ekstra logikk-konverteringskrets måtte vært brukt. Raspberry Pi er også i motsetning til Arduino mer følsom for overspenning som kan forekomme fra tilkoblet utstyr. Videre har gruppa god erfaring med bruk av pwm-signaler generert av Arduino.

Den største fordelen med Raspberry Pi i dette prosjektet er at 5G-HAT som er brukt er laget for Raspberry Pi. Den kan brukes med andre enheter også, men er hovedsaklig designet for Raspberry Pi. Dette, i kombinasjon med tidligere erfaring, gjorde at Raspberry Pi ble sett på som beste løsning i starten av prosjektet.

Raspberry Pi krever 3A som vist i vedlegg B.2 for å fungere optimalt. Det er ikke alle powerbanker som kan levere så mye strøm, og i mange nettbutikker er det heller ikke opplyst hvor mye strøm hver port på powerbank kan levere. Dette har ført til mye prøving og feiling i prosjektet før vi fant frem til en powerbank som fungerte optimalt for løsningen.

Alternativer til Raspberry Pi ble også vurdert, derunder Nvidia Jetson Nano. Også Jetson Nano har mye kraft for sin størrelse og strømtrekk, med muligheten for oppkobling av GPIO-pinner.

Jetson Nano kjører kjører likt som Raspberry Pi også Linux, med egen distro. Videre er det også mulighet for kommunikasjon med ekstern maskinvare.

Videre er det også egen maskinvareløsning for video- enkoding og dekoding.

Utfordringene ved å bruke Jetson Nano er tilgjengeligheten av informasjon og tidligere prosjekter. Sammenlignet med Raspberry Pi som har ett stort og åpent utviklermiljø med nettforum og guider. Videre har Raspberry Pi dominert markedet for ettkortsdatamaskiner de siste tiårene, som gjør at det også finnes mange utvidelseskort spesifikt laget for Raspberry Pi.

9.2.2 5G-HAT

For 5G-kommunikasjon var det behov for utvidelseskort montert på Raspberry Pi. Dette fordi Raspberry Pi i utgangspunktet kun har integrert løsning for trådløskommunikasjon via WiFi og bluetooth. Da 5G-teknologien er særdeles ny, finnes det få kommersielt tilgjengelige enheter som kommunisere over 5G. Grunnet begrensninger i det kommersielle markedet, ble to alternativer vurdert.

Det første alternativet var å bruke ett vanlig USB-modem av typen 'Huawei 5G Mobilt WiFi E6878-870'. Dette modemet har innebygget batteri og WiFi5-aksesspunkt, men ingen kablet tilkobling for kommunikasjon som gjør at en innfører enda ett usikkerhetsmoment i kommunikasjonen.

Det andre alternativet var en såkalt HAT av typen Waveshare 'SIM8200EA-M2 5G'. Denne kommer med mekanisk innfestning koblet til GPIO-pinnene på Raspberry Pi og kommuniserer med Raspberry Pi via USB. Utfordringer knyttet til HAT sammenlignet med USB-modemet er at HAT ikke kommer med egen batteripakke. Dette gjør at en må ha en ekstern batteripakke på bilen, i tillegg til Raspberry Piens batteripakke. I motsetning til USB-modem fra Huawei har HAT eksterne antenner som kan tenkes å gi bedre rekkevidde.

Etter anbefaling fra eksternveileder ble 5G-HAT fra Waveshare valgt.
Ved oppsett av begge enhetene, ble det oppdaget feil på den ene 5G-HAT. Dette resulterte i en del feilsøking og kommunikasjon med Waveshare for å løse feilen. Det viste seg at det manglet en kondensator på det ene kortet. Etter instruks fra Waveshare ble det gjort forsøk på å kortslutte terminalene kondensatoren skulle vært loddet fast på. Da dette ikke hjalp ble det gjort forsøk på å laste opp ny firmware til kortet. Heller ikke dette løste problemet, så det måtte sendes nytt kretskort fra Waveshare i Kina, noe som gjorde at fungerende 5G-HAT til begge Raspberry Pi først var på plass 5-6 uker før oppgaven skulle leveres. Dette gjorde at mye av testingen og måling av forsinkelse først kunne gjøres nær slutten av prosjektet. Det gjorde at oppgaveskriving også ble forskjøvet i forhold til den

opprinnelige planen.

I tillegg til ett defekt kort har det også vært utfordringer med å holde 5G-HAT fast på 5G. Ifølge dokumentasjonen skal det være mulig å sette 5G-HAT i en fast 5G-modus (CNMP modus 71), men dette har kun ført til at 5G-HAT ikke ville koble seg opp i det hele tatt [105]. Ved testing av både ping, styresignal og video har 5G-HAT stått i dynamisk modus. Via AT commands har det blitt bekreftet at 5G-HAT har vært tilkoblet i riktig nettverksmodus under disse testene [106].

Gruppen har vært i kontakt med en gruppe masterstudenter ved NTNU som bruker samme 5G-HAT, og de kan bekrefte samme problematikken knyttet til nettverksmodus.

9.2.3 Video

Hvordan man skulle oppnå video med svært lav forsinkelse har vært en av de mest utfordrende problemene i prosjektet. Ingen i gruppen hadde erfaring med video enkoding på forhånd, og det stiltes strenge krav til lav forsinkelse.

I starten av prosjektet visste gruppen at Raspberry Pi hadde støtte for maskinvareakselerert video enkoding/dekoding av H.264, og at det fantes mye informasjon rundt dette på nettet. Det ble brukt mye tid på å lese om video enkoding, og deretter om hvilken programvare andre hadde brukt for å strømme video.

I starten ble 'raspivid' brukt for å sende video, og VLC for å motta. Det var enkelt å komme i gang med dette, og det ga raskt fungerende video med ca. 200 ms forsinkelse over LAN.

Det har blitt oppnådd forsinkelse på ca. 100-150 ms ved bruk av raspivid og VLC, men det var flere problemer med videoen. Blant annet oppsto det mange forstyrrelser i bildet når data ble sendt over UDP. Disse forstyrrelsene var ikke tilstede når data ble sendt over TCP, men TCP ga høyere forsinkelse over 4G enn det UDP gjorde. Det ble også tydelig at forsinkelsen økte over tid, etter bare noen minutter var det merkbart høyere forsinkelse enn når videoen ble startet.

Det ble gjort tester med raspivid/VLC hvor video ble sammenlignet over TCP/UDP. På lokalnett var det svært liten forskjell i forsinkelse, men over 4G var det merkbart høyere forsinkelse om man brukte TCP fremfor UDP. En ulempe ved å bruke UDP var at det oppsto forstyrrende 'artifacts' i bildet i form av firkanter. Det ble også oppdaget at videoen hadde lav forsinkelse i starten, men i løpet av noen minutter vokste forsinkelsen seg større og større.

Videre ble GStreamer testet og ved subjektive vurderingsmetoder så det ut til at GStreamer ga lavest forsinkelse. De nevnte problemene rundt raspivid og VLC var

ikke tilstede ved bruk av GStreamer. Utover dette gir GStreamer også mer kontroll over enkoding- og dekoding parametere.

Ved å teste forskjellige enkoding-parametere, og forskjellige API for dekoding ble det oppnådd video med ca. 100 ms forsinkelse over LAN. Dette er trolig nær grensen for hva som er mulig å oppnå med H.264 videokodeken på Raspberry Pi 3. Raspberry Pi støtter maskinvareakselerert enkoding/dekoding av H.264 video. Det viser seg at Raspberry Pi 4B ikke støtter OpenMAX (omx) API, som benyttes til dekoding av H.264 video. Andre API kan benyttes, men dette gir dårligere ytelse. Dette førte til at dekoding-hastigheten på Raspberry Pi 4B ble svært mye tregere enn på Raspberry Pi 3B+. Derfor benyttes Raspberry Pi 3B+ på pilot-side.

9.2.4 Arduino

Under testing ble det hovedsaklig brukt en Arduino Uno som mikrokontroller for motorkontroller. Arduino er et svært utbredt innenfor mikrokontrollere på hobby- og proffnivå. Alle på gruppen hadde erfaring med Arduino fra før av og alle hadde det tilgjengelig under testfasen. Problemet med Arduino Uno er at 16-bit telleren som brukes til PWM har kun to utganger, og en av de brukes til SlaveSelect for SPI. ESC og servo trengte hver sin utgang så en alternativ Arduino måtte brukes i den ferdige løsningen. Til slutt ble en Arduino Nano Every valgt fordi den har flere 16 bits tellere og teller 1 (telleren som brukes) har tre utganger hvor ingen hindres av SPI. Den er også ganske liten i forhold til mange andre mikrokontrollere noe som passer bra til den begrensede plassen på bilen. Det ble tidlig bestemt av en mikrokontroller skulle stå for sending av PWM signal til ESC og styringsservo. Dette var så Raspberry Pi kunne fokusere på sending/mottak av styringssignaler og video.

I^2C/SPI

Et av problemene med løsningen er at ratt/pedaler sender data veldig ofte. Dermed må overføringen fra Raspberry Pi til Arduino på bilen holde følge, passe på at riktig data kommer fram og beholde rekkefølgen på dataen. Først ble I^2C testet. Det ble fastslått at I^2C ikke er en rask nok løsning. Om data sendes raskere over I^2C enn den kan mottas holdes sendingen igjen og skaper store forsinkelser i systemet. Arduinoen kan også stoppe og må restartes manuelt. Om dette skjer under kjøring kan det føre til at bilen krasjer og noe blir ødelagt. Dermed måtte det byttes til SPI. Problemet med SPI er at det ikke brukes egne bibliotek for innlesning av data. Dermed må det lages en løkke som leser inn en og en byte til hele meldingen er lest inn. Dersom timingen ikke er helt riktig blir feil data lest inn, noe som kan gi uventede verdier ut på motor/servo. Heldigvis er SPI rask nok til sende data like raskt som det leses inn fra ratt og pedaler.

PWM signal

En servo opererer vanligvis på 50Hz signaler. Derfor var dette førstevalget for testing av ESC og styringsservo. Senere i prosjektet ble strøm til servo hentet fra ESC slik den originalt er. Dette var ikke gjort under tidlig testing fordi det krevde at batteriet til bilen var koblet i og ESC var skrudd på. Når 6V ble hentet fra ESC kunne styringssignal til servoen gi utslag på motoren i form av forstyrrelser over 6V koblingen. Det ble gjort en del testing på ulike løsninger og den letteste løsningen var å endre PWM signalene til 100Hz. Testing viste at den originale senderen til bilen også brukte et 100Hz signal. Dette fjernet alt av forstyrrelser, men det ble ikke brukt mer tid på å finne ut akkurat hvorfor. 100Hz fungerte like bra som 50Hz på både ESC og styringsservo. Dermed ble dette endret i den ferdige arduinokoden.

9.2.5 Menneskelig grensesnitt

Når det kom til menneskelig grensesnitt sto det mellom 3 alternativer. Ratt/pedaler, tastatur og en spillkontroller. Tastatur følte gruppen ikke ville gi ett godt inntrykk på brukeren, dette kunne også føre til at det ble vanskelig å kjøre. Dette kunne ha ført til at piloten følte at han ikke har god kontroll over kjøretøyet, som var ett krav til oppgaven. En spillkontroller kunne ha fungert men dette ble også noe gruppen anså som en ekstra barriere mellom pilot og kontroll over bilen for at piloten skulle føle på kontroll over bilen.

Da falt valget på ratt og pedaler fordi det gir den mest realistiske følelsen av å kjøre en ordentlig bil. Dette var innenfor budsjettet og følelsen av å faktisk kjøre bilen hjelper mye på den helhetlige følelsen av oppgaven. En fordel med tastatur eller spillkonsoll er at de går på batterier eller strøm fra Raspberry Pi på pilotsiden. Med ratt/pedaler må 230V strømuttak være tilgjengelig og det tar mye større plass under transport. Det var også nødvendig med en god stol å sitte i for å få den fulle effekten av rattet og pedalene. Denne tok også mye plass under transport, men var med på å forbedre kjøreopplevelsen.

9.2.6 Styresignal over nettverk

Første nettverksløsningen baserte seg på ASIO (Asynchronous input/output) som er et rammeverk for C++. Denne nettverksløsningen baserte seg på flertrådskjøring der nettverkskommunikasjon gikk i en egen tråd ved siden av 'hoved' programmet. Ved bruk av denne løsningen kunne bilen oppdatere dataene sine ved forespørsel fra pilot-pi ved behov. I denne løsningen fungerte bil-pi som klient som koblet seg opp til pilot-pi, altså motsatt fra endelig løsning.

```
void hentdata(asio::ip::tcp::sock& sock)
{
        sock.async_read_some(asio::buffer(vBuffer.data(), vBuffer.size()),
                [&](std::error_code ec, std::size_t lengde)
                {
                        if (!ec)
                        {
                                std::cout << "\n\r Avlest: " << lengde << "
                                    bytes\n\n";

                                for (int i = 0; i < lengde; i++)
                                        std::cout << vBuffer[i];

                                hentdata(sock);
                        }
                }
        );
}
```

Kodeliste 9.1: Tidligereløsning - fra C++ ASIO prototype

I koden ser man et eksempel på hvordan bil-pi ville hentet data fra pilot-pi. hentdata er en rekursiv funksjon som bruker socket objektets async_readsome funksjon for å requeste data fra piloten. De mottatte datane vil så bli fylt opp i bufferen Vbuffer.

På grunn av sin asynkrone funksjonalitet samt flere tråder hadde denne løsningen mulighet til å skrive til SPI samtidig som den mottok fersk data.

Problemer oppsto med biblioteket da løsningen ble sydd sammen med seriell kommunikasjon til og fra Arduino og input fra ratt og pedaler. Gruppen valgte da å vrake løsningen for en enklere nettverksløsning som ikke var avhengig av eksterne biblioteker.

I den endelige løsningen ble det først brukt toveis kommunikasjon mellom bilen og piloten. Der det ble anvendt en variabel kalt 'readyData'. Løsningen baserte seg på at bil-pi ville sende en melding til pilot-pi når den hadde skrevet data ferdig til mikrokontrolleren. Klienten ville så sende den ferskeste verdien avlest fra ratt og pedal til bilen.

Denne løsningen fungerte bra når det ble testet med sending av tilfeldige tall mellom klient og tjener. Det oppsto problemer når dataene ikke ville gå igjennom filteret. Samt at systemet noen ganger kunne komme ut av 'synk' slik at begge gikk i lyttemodus samtidig. På grunn av for mye tid på å finne en løsning og ingen resultat ble også denne funksjonaliteten tatt ut.

Dermed måtte en forsinkelse mellom hver sending av data innføres. Denne passer på at mikrokontrolleren har tid til å prosessere mottatt data før Raspberry Pi sender neste verdi. Dette er på ingen måter en optimal løsning, men fungerer bra nok for formålet. Med lengre tid kunne problemene med 'readyData' vært fikset, men den har ikke har stor betydning for funksjonaliteten på løsningen.

Det har ikke blitt testet eksakt hvor lang tid nettverkskommunikasjonen tar alene, men det har blitt testet sammen med SPI. Dermed målte testen fra tidspunktet før dataene ble sendt til tjeneren til etter bilens motor har fått ny verdi.

Denne testen viser til at løsningen på kablet nettverk har en forsinkelse på under 1 ms, resultatene lå omlag 650 mikrosekunder. Dette vil si at siden systemets interne forsinkelse er neglisjerbar, vil styresignalets forsinkelse i all hovedsak gå ut ifra nettverkets hastighet.

Ved tester over 5G ser man at forsinkelsen ligger på omlag 20-30 ms. Noe som tilsvarer de hastighetene som har blitt målt tidligere vedbruk av vanlig ping test.

Begrunnelse for valg

Valg av språk

I starten var det en diskusjon om hvilke programmeringsspråk løsningen skulle utvikles i, og valget falt på C++ da det var ønsket minst mulig treghet i systemet.

Valg av transportprotokoll

Når det skulle velges transportprotokoll sto valget mellom UDP og TCP. UDP er en lettvektig transportprotokoll som i motsetning til TCP ikke har noen form for feilsjekk for at dataene blir avlest i riktig rekkefølge. Det er heller ingen test for om noen av datapakkene blir tapt under overføring. UDP bruker mindre data en TCP. Der TCP har et maksimum header på 60 byte og en minimum header på 20 byte, har UDP kun en headerstørrelse på 8 byte.
Valget falt likevel på TCP da løsningen er avhengig av at dataene kommer frem og i riktig rekkefølge for å at bilen skal få riktig instruksjoner.
Med en polling rate på 1000hz fra ratt/pedal vil den maksimale bitraten på styresignalet ved UDP være:

$$8 \times (8byte + 4 \times 2byte) \times 1000hz = 128\frac{Kbit}{s}.$$

mens et normalt TCP foreløp med en header størrelse på 20 byte vil bitraten være:

$8 \times (20byte + 4 \times 2byte) \times 1000hz = 224\frac{Kbit}{s}$

med maksimal TCP header på 60 byte vil bitraten være:

$8 \times (60byte + 4 \times 2byte) \times 1000hz = 544\frac{Kbit}{s}$.

Ettersom bitraten på 4G/5G er svært stor i forhold til den teoretiske maksimalstørrelsen på styresignalet at det er mest tidseffektivt å bruke TCP. Da vil det også bli frigjort mer tid i utviklingsprosessen i stedet for å måtte utvikle en egen kontrollsjekk på dataene.

9.3 Hva kunne vært gjort annerledes

9.4 Fremtidige løsninger

Den totale videoforsinkelsen fra kamera til skjerm er i gjennomsnitt ca. 125ms. Av dette utgjør video enkoding/dekoding ca. 100ms, og nettverksforsinkelse over 5G non-standalone ca. 25ms. Video behandling utgjør altså 80% av forsinkelsen. For å få ned forsinkelsen fra kamera til skjerm vil det dermed være mest å hente på å forbedre video enkoding/dekoding. H.264 videokodeken ble standardisert i 2003, mer moderne videokodeker vil potensielt kunne gi lavere forsinkelse. Det optimale vil trolig være maskinvare som støtter maskinvareakselerert enkoding/dekoding av en nyere videokodek.

Videre vil fremtidig 5G-standalone URLLC potensielt kunne kutte nettverksforsinkelsen fra ca. 20-25ms til nær 0ms (dagens 5G-non standalone med begge enheter i Trondheim, VS. 5G-standalone URLLC, med bil og pilot på samme basestasjon) [6].

9.4.1 Bruk av dedikert maskinvare for enkoding/dekoding

Etter intern testing samt eksterne resultater på enkodingen/dekodingen med H.264 på Raspberry Pi ser det ut som det ikke er mulig å nå langt lavere forsinkelse på dette. En mulighet gruppen ser på dette er å flytte arbeidet over på mer egnet ekstern maskinvare.

For å få lavere forsinkelse på video er det mest å hente på å forbedre video enkoding/dekoding prosessen. Etter testene gjennomført i denne oppgaven, samt ekstern testing ser det ut som den nåværende løsningen for videoenkoding ikke kan bli stort raskere med maskinvaren som brukes nå.

Men det er mulig å få den lavere ved å bruke en egen ASIC eller FPGA [107].

Det er med dette mulig å kjøpe IP core med H.264 videokodek, noen av disse lover video enkoding/dekoding på under ett ms [108]. I eksemplet vist i artikkelen fra Intel [107], viser de til ett system som har capture-to-display forsinkelse på 20.54ms ved 1080p 30fps. Dette er fem ganger så raskt som det som er oppnådd med Raspberry Pi i dette prosjektet. Om man legger til resultatene fra nettverks testen ville videosignalet da ha en total forsinkelse på ca.
$25ms\ 5G + 20.54ms\ enkoding/dekoding\ =\ \underline{\underline{45.54ms}}\ totalt.$

Mens den nåværende løsningen har en forsinkelse på 125ms. Ved bruk av ASIC/FPGA kunne det ha vært mulig å senke forsinkelsen på videosignalet med:
$100 - \frac{45.54}{125} \times 100\% = \underline{\underline{63.56}}\%$
mens i det totalet systemet fra ratt til skjerm ville det vært en forskjell fra 151 totalt til 71.54 føre til en senking av forsinkelsen på:
$(100 - \frac{2\times25\ ms\ 5G\ +\ 1\ ms\ styresignal\ +20.54\ ms\ videokodek}{2\times25\ ms\ 5G\ +\ 1\ ms\ styresignal\ +100\ ms\ videokodek}) \times 100\% = (100 - \frac{71.54}{151}) \times 100\% = 52\%$ totalt i systemet.

9.5 Konsekvenser av arbeidet

Oppgaven kan gi grunnlag for videre utvikling at fjernstyrte kjøretøy over 5G. Som tidligere forklart er oppgaven et 'proof of concept' og ikke et ferdig utviklet produkt. Resultatene fra testingen gir et tydelig bilde på hva som er mulig med et begrenset budsjett og tid, men også hvilke resultatet videre optimalisering kan gi. Kjøring av bilen er med på å støtte opp resultatene og viser at dette er et levedyktig alternativ til vanlig styring av kjøretøy.

Kapittel 10

Konklusjon

Under prosjektet er det designet, bygget og testet et kjøretøy som kan styres over 5G eller 4G. For sending av styrings- og videosignal er det laget et relativt robust system, noe som gir muligheter for kjøring uten visuell kontakt mellom pilot og kjøretøyet. Prosjektet hadde et budsjett på opptil 50.000,-, totalkostnaden knyttet til utvikling var rundt 30.000,-.

Oppgaven viser at fjernstyring av kjøretøy over 5G/4G er mulig. Det er tydelig at fjernstyring over 5G gir både mer stabil og mer responsiv video enn fjernstyring over 4G.

Forsinkelse fra kamera til skjerm har blitt målt til 123ms over 5G, kravet på total forsinkelse under 100ms er dermed ikke er nådd. En mulig løsning på dette er dedikert maskinvare for enkoding og dekoding av video. Forsinkelsen på styringssignalet er målt til å være tilnærmet lik nettverksforsinkelsen, cirka 25ms over 5G, altså betydelig lavere enn videoforsinkelsen. Til tross for større forsinkelse gjør valg av ratt og pedaler at piloten, subjektivt sett, har god kontroll over kjøretøyet.

I dette prosjektet har en tidlig versjon av 5G-nettet blitt benyttet. De neste årene vil oppgradering av 5G-nettet kunne kutte kraftig ned på nettverksforsinkelsen. Videre kan maskinvare dedikert til enkoding og dekoding av video brukes for å redusere forsinkelsen i systemet ytterligere. Video-prossesering står for mesteparten av forsinkelsen i dagens løsning, som vist i figur 8.4.

Bibliografi

[1] Yara. (2018). «Yara Birkeland press kit,» adresse: https://www.yara.com/news-and-media/press-kits/yara-birkeland-press-kit/.

[2] Telenor. (2019). «Bygger verdens første selvkjørende fraktskip,» adresse: https://www.telenor.no/bedrift/aktuelt/internet-of-things/yara/.

[3] (2021). «5G Solutions - LL3: Smart Cities Ports,» adresse: https://www.5gsolutionsproject.eu/living-labs/smart-cities-and-ports/.

[4] (2021). «5G Solutions - Living Labs,» adresse: https://5gsolutionsproject.eu/living-labs/.

[5] Nasjonal kommunikasjonsmyndighet. (2020). «Om 5G,» adresse: https://www.nkom.no/frekvenser-og-elektronisk-utstyr/om-5g.

[6] International Telecommunication Union, «Minimum requirements related to technical performance for IMT-2020 radio interface(s),» 2017, s. 7–8. adresse: https://www.itu.int/dms_pub/itu-r/opb/rep/R-REP-M.2410-2017-PDF-E.pdf.

[7] Microchip Technology. (n.d.). «megaAVR® Data Sheet "ATmega48A/PA/88A/PA/168A/PA/328/P" adresse: https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061B.pdf.

[8] Microchip Technology. (n.d.). «ATmega4808/4809 Data Sheet,» adresse: https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega4808-09-DataSheet-DS40002173C.pdf.

[9] S. Campbell. (2016). «BASICS OF THE I2C COMMUNICATION PROTOCOL,» adresse: https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/.

[10] B. Paradigm. (2016). «Introduction to I²C and SPI protocols,» adresse: https://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/.

[11] S. Campbell. (2016). «BASICS OF THE SPI COMMUNICATION PROTOCOL,» adresse: https://www.circuitbasics.com/basics-of-the-spi-communication-protocol/.

[12] M. Barr. (2001). «Introduction to Pulse Width Modulation(PWM),» adresse: https://barrgroup.com/Embedded-Systems/How-To/PWM-Pulse-Width-Modulation.

[13] R. D. L.WernliSr. (2014). «The ROV Manual (Second Edition) "A User Guide for Remotely Operated Vehicles",» adresse: https://www.sciencedirect.com/science/article/pii/B9780080982885000075.

[14] (2021). «Pulse-width modulation,» adresse: https://en.wikipedia.org/wiki/Pulse-width_modulation.

[15] P. Fox. (2020). «Transmission Control Protocol (TCP),» adresse: https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:the-internet/xcae6f4a7ff015e7d:transporting-packets/a/transmission-control-protocol--tcp.

[16] Florida Tech. (2006). «TCP/IP Protocol,» adresse: https://cs.fit.edu/~mmahoney/cse4232/tcpip.html.

[17] Telenor. (2021). «Visste du dette om Telenors dekning?» Adresse: https://www.telenor.no/dekning/.

[18] O. R. Valmot. (2019). «5G, fiber eller satellitt: Måten vi transporterer digital informasjon på endres nok en gang,» adresse: https://www.tu.no/artikler/5g-fiber-eller-satellitt-maten-vi-transporterer-digital-informasjon-pa-endres-nok-en-gang-br/469150.

[19] (2021). «5G-VINNI,» adresse: https://www.5g-vinni.eu.

[20] D. Jones. (2016). «Picamera documentation - Camera Hardware,» adresse: https://picamera.readthedocs.io/en/release-1.13/fov.html.

[21] International Telecommunication Union, «ITU-T Recommendation H.264,» 2003, s. 1, 3, 5. adresse: http://handle.itu.int/11.1002/1000/6312-en?locatt=format:pdf&auth.

[22] (2020). «Video compression picture types,» adresse: https://en.wikipedia.org/wiki/Video_compression_picture_types.

[23] C. J. van den Branden Lambrecht, «Vision models and applications to image and video processing,» 2001, s. 209. adresse: https://www.springer.com/gp/book/9780792374220.

[24] (2021). «List of monochrome and RGB color formats,» adresse: https://en.wikipedia.org/wiki/List_of_monochrome_and_RGB_color_formats#3-level_RGB.

[25] (?). «YCBCR COLOR SPACES,» adresse: https://www.hisour.com/ycbcr-color-spaces-26075/.

[26] International Telecommunication Union. (2017). «Signalling, backward compatibility and display adaptation for HDR/WCG video coding,» adresse: https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-H.Sup18-201710-I!!PDF-E&type=items.

[27] (2021). «YCBCR COLOR SPACES,» adresse: https://en.wikipedia.org/wiki/YCbCr.

[28] dataVideo. (2020). «What are 8-bit, 10-bit, 12-bit, 4:4:4, 4:2:2 and 4:2:0,» adresse: https://datavideo.com/eu/article/What%20are%208-bit,%2010-bit,%2012-bit,%204:4:4,%204:2:2%20and%204:2:0.

[29] (n.d.). «Raspberry Pi 3 Model B+,» adresse: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/.

[30] Chris Fleck, Eben Upton. (2021). «40 Million Raspberry Pi sold,» adresse: https://twitter.com/chrisfleck/status/1392224692862427138.

[31] (2020?). «SIM8200EA-M2 5G HAT for Raspberry Pi,» adresse: https://www.waveshare.com/sim8200ea-m2-5g-hat.htm.

[32] (n.d.). «Arduino Every Overview,» adresse: https://store.arduino.cc/arduino-nano-every.

[33] (n.d.). «Traxxas 3018R Sealed XL-5 Forward/Reverse/Brake ESC,» adresse: https://www.samirc.no/products/traxxas-sealed-xl-5-forwardreversebrake-esc-.

[34] (n.d.). «Traxxas 2075 Digital High Torque Servo,» adresse: https://www.samirc.no/products/traxxas-2075-digital-high-torque-servo-.

[35] (n.d.). «GPIO Raspberry Pi Documentation,» adresse: https://www.raspberrypi.org/documentation/hardware/raspberrypi/gpio/README.md.

[36] (n.d.). «List of things to consider when ordering PCBs from Elektronikk- og Prototypelaben,» adresse: https://innsida.ntnu.no/documents/portlet_file_entry/10157/List+of+things+to+consider+when+ordering+PCBs+from+Elektronikk.pdf/91858740-a5f1-4399-b1b4-a4369e28ce88?status=0.

[37] (n.d.). «Bi-Directional Logic Level Converter Hookup Guide,» adresse: https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide.

[38] (n.d.). «The importance of pcb trace widths in pcb design,» adresse: https://www.sfcircuits.com/pcb-school/pcb-trace-widths.

[39] (n.d.). «PCB Trace Width Calculator,» adresse: https://www.7pcb.com/trace-width-calculator.php.

[40] (n.d.). «Design rule checker,» adresse: https://documentation.circuitmaker.com/display/CMAK/PCB_Dlg-DesignRuleChecker((Design+Rule+Checker))_CM.

[41] (n.d.). «Logitech G29 Driving Force Racing,» adresse: https://www.logitechg.com/en-us/products/driving/driving-force-racing-wheel.html.

[42] (n.d.). «Playseat Challenge,» adresse: https://www.komplett.no/product/771684/gaming/gaming-utstyr/spillkontrollere/simulator/playseat-challenge?feature=freightwidget.

[43] Raspberry Pi Foundation. (2016). «Camera Module V2,» adresse: https://www.raspberrypi.org/products/camera-module-v2/.

[44] Raspberry Pi Foundation. (2020). «Raspberry Pi High Quality Camera,» adresse: https://www.raspberrypi.org/products/raspberry-pi-high-quality-camera/.

[45] (n.d.). «Traxxas Slash 1/10 2WD Black RTR :: Komplett,» adresse: https://www.elefun.no/p/prod.aspx?v=45769.

[46] (n.d.). «Joystick API Documentation,» adresse: https://www.kernel.org/doc/Documentation/input/joystick-api.txt.

[47] (n.d.). «Evdev input,» adresse: https://www.kernel.org/doc/html/latest/input/input.html.

[48] (n.d.). «Linux Core API Documentation,» adresse: https://www.kernel.org/doc/html/v4.12/core-api/index.html.

[49] (2021). «calloc,» adresse: https://www.cplusplus.com/reference/cstdlib/calloc/.

[50] (2021). «Basic Input/Output,» adresse: https://www.cplusplus.com/doc/tutorial/basic_io/.

[51] (2021). «fcntl(2) — Linux manual page,» adresse: https://man7.org/linux/man-pages/man2/fcntl.2.html.

[52] S. Kelly. (2018). «Big Endian and Little Endian,» adresse: https://www.youtube.com/watch?v=cNdlrbZSkyQ.

[53] codehoose. (2018). «bbclient.cpp,» adresse: https://gist.github.com/codehoose/d7dea7010d041d52fb0f59cbe3826036.

[54] codehoose. (2018). «bbtcpserver.cpp,» adresse: https://gist.github.com/codehoose/020c6213f481aee76ea9b096acaddfaf.

[55] (n.d.). «Registered Port,» adresse: https://www.sciencedirect.com/topics/computer-science/registered-port.

[56] (n.d.). «getopt(3) — Linux manual page,» adresse: https://man7.org/linux/man-pages/man3/getopt.3.html.

[57] (n.d.). «strcpy,» adresse: https://www.cplusplus.com/reference/cstring/strcpy/.

[58] (2021). «struct sockaddr_in, struct in_addr,» adresse: https://www.gta.ufrj.br/ensino/eel878/sockets/sockaddr_inman.html.

[59] (2001). «htons(3) - Linux man page,» adresse: https://linux.die.net/man/3/htons.

[60] (n.d.). «Big Endian and Little Endian,» adresse: https://chortle.ccsu.edu/AssemblyTutorial/Chapter-15/ass15_3.html.

[61] (2021). «socket(2) — Linux manual page,» adresse: https://man7.org/linux/man-pages/man2/socket.2.html.

[62] (2021). «inet_pton(3) — Linux manual page,» adresse: https://man7.org/linux/man-pages/man3/inet_pton.3.html.

[63] (2021). «connect(2) — Linux manual page,» adresse: https://man7.org/linux/man-pages/man2/connect.2.html.

[64] (2021). «send(2) — Linux manual page,» adresse: https://man7.org/linux/man-pages/man2/send.2.html.

[65] (n.d.). «chrono C++ Reference,» adresse: https://www.cplusplus.com/reference/chrono/.

[66] (n.d.). «sys/time.h - time types,» adresse: https://pubs.opengroup.org/onlinepubs/7908799/xsh/systime.h.html.

[67] (2021). «bind(2) — Linux manual page,» adresse: https://man7.org/linux/man-pages/man2/bind.2.html.

[68] (2021). «accept(2) — Linux manual page,» adresse: https://man7.org/linux/man-pages/man2/accept.2.html.

[69] (n.d.). «memset(3) — Linux manual page,» adresse: https://man7.org/linux/man-pages/man3/memset.3.html.

[70] (n.d.). «getnameinfo(3) — Linux manual page,» adresse: https://man7.org/linux/man-pages/man3/getnameinfo.3.html.

[71] (n.d.). «setsockopt(3p) — Linux manual page,» adresse: https://man7.org/linux/man-pages/man3/setsockopt.3p.html.

[72] (n.d.). «signal(7) — Linux manual page,» adresse: https://man7.org/linux/man-pages/man7/signal.7.html.

[73] (n.d.). «recv(2) — Linux manual page,» adresse: https://man7.org/linux/man-pages/man2/recv.2.html.

[74] R. Heymsfeld. (n.d.). «raspberry pi to arduino spi communication,» adresse: http://robotics.hobbizine.com/raspiduino.html.

[75] J. A. Donenfeld, «WireGuard: Next Generation Kernel Network Tunnel,» 2020, s. 18. adresse: https://www.wireguard.com/papers/wireguard.pdf.

[76] J. A. Donenfeld. (2020). «WireGuard: Next Generation Kernel Network Tunnel,» adresse: https://www.wireguard.com/papers/wireguard.pdf.

[77] (n.d.). «About GStreamer: Foundations,» adresse: https://gstreamer.freedesktop.org/documentation/application-development/introduction/basics.html?gi-language=c.

[78] (n.d.). «GStreamer: capsfilter,» adresse: https://gstreamer.freedesktop.org/documentation/coreelements/capsfilter.html?gi-language=c.

[79] (n.d.). «C++ Introduction,» adresse: https://www.w3schools.com/cpp/cpp_intro.asp.

[80] (n.d.). «What is Python? Executive Summary,» adresse: https://www.python.org/doc/essays/blurb/.

[81] J. Anderson. (n.d.). «Python vs C++: Selecting the Right Tool for the Job,» adresse: https://realpython.com/python-vs-cpp/#summary-python-vs-c.

[82] (n.d.). «Hello World - What is GitHub?» Adresse: https://guides.github.com/activities/hello-world.

[83] (n.d.). «Develop C and C++ applications,» adresse: https://visualstudio.microsoft.com/vs/features/cplusplus/.

[84] (n.d.). «Connect to your target Linux system in Visual Studio,» adresse: https://docs.microsoft.com/en-us/cpp/linux/connect-to-your-remote-linux-computer?view=msvc-160.

[85] (n.d.). «Arduino IDE,» adresse: https://www.arduino.cc/en/guide/environment.

[86] (n.d.). «Visual Studio or Atmel Studio?» Adresse: https://www.visualmicro.com/page/User-Guide.aspx?doc=Getting-started-which-IDE.html.

[87] (n.d.). «Evdev,» adresse: https://en.wikipedia.org/wiki/Evdev.

[88] (n.d.). «SketchUp,» adresse: https://en.wikipedia.org/wiki/SketchUp.

[89] (n.d.). «SketchUp STL,» adresse: https://extensions.sketchup.com/extension/412723d4-1f7a-4a5f-b866-281a3e223337/sketch-up-stl.

[90] PuTTY. (n.d.). «PuTTY,» adresse: https://www.putty.org/.

[91] winscp. (n.d.). «Winscp,» adresse: https://winscp.net/eng/index.php.

[92] (n.d.). «What is Notepad++,» adresse: https://notepad-plus-plus.org/.

[93] (n.d.). «circuitmaker,» adresse: https://circuitmaker.com/About.

[94] Linuxize.com. (2019). «Getting started with Tmux,» adresse: https://linuxize.com/post/getting-started-with-tmux/.

[95] (n.d.). «Tmux Cheat Sheet Quick Reference,» adresse: https://tmuxcheatsheet.com.

[96] SIMCom. (n.d.). «SIM8200 Series AT Command Manual,» adresse: https://www.waveshare.com/w/upload/1/17/SIM8200_Series_AT_Command_Manual_V1.00.01_0515.pdf.

[97] (2019). «Speed read/write Arduino I/O,» adresse: http://kunoichi.be/projects/speed-read-arduino-i-o/.

[98] (2019). «Speed read/write Arduino I/O,» adresse: https://www.best-microcontroller-projects.com/arduino-millis.html.

[99] (2015). «Benchmarking Raspberry Pi GPIO Speed,» adresse: https://codeandlife.com/2012/07/03/benchmarking-raspberry-pi-gpio-speed/.

[100] (n.d.). «WiringPi blink example,» adresse: http://wiringpi.com/examples/blink/.

[101] (n.d.). «Mouse polling rate,» adresse: https://wiki.archlinux.org/title/Mouse_polling_rate.

[102] (n.d.). «Speed read/write Arduino I/O,» adresse: https://roboticsbackend.com/arduino-fast-digitalwrite/#Why_Arduino_digitalWrite_is_not_fast.

[103] R. J. Binns. (2016). «Edge Transient Effects on Power LED Switching,» adresse: https://www.researchgate.net/publication/320021349_Edge_Transient_Effects_on_Power_LED_Switching.

[104] Khronos Group. (n.d.). «OpenMAX Overview,» adresse: https://www.khronos.org/openmax/.

[105] SIMCom, «SIM8200 Series AT Command Manual,» n.d. S. 18. adresse: https://www.waveshare.com/w/upload/1/17/SIM8200_Series_AT_Command_Manual_V1.00.01_0515.pdf.

[106] SIMCom, «SIM8200 Series AT Command Manual,» n.d. S. 88–92. adresse: https://www.waveshare.com/w/upload/1/17/SIM8200_Series_AT_Command_Manual_V1.00.01_0515.pdf.

[107] N. Zervas. (2016). «Video Streaming with Near-Zero Latency Using Altera Arria V FPGAs and Video and Image Processing Suite Plus the Right Encoder,» adresse: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-cast-low-latency.pdf.

[108] (n.d.). «H264-D-BP Low-Latency AVC/H.264 Baseline Profile Decoder,» adresse: https://www.cast-inc.com/compression/avc-hevc-video-compression/h264-d-bp/.

Vedlegg A

Prosjektavtale

NTNU
FAKULTET FOR INFORMASJONTEKNOLOGI OG ELEKTROTEKNIKK

AVTALE

Avtale for gjennomføring av bacheloroppgaven mellom NTNU, oppdragsgiver (firma, etat) og student(er).

Avtalepartnere

NTNU Institutt for elektroniske systemer / elkraft / teknisk kybernetikk	Veileders navn/telefon/e-postadresse.: Cuong Phu Le +47 942 57 411, cuong.le@ntnu.no
Oppdragsgiver (Firma/etat): Yara Norge AS	Kontaktperson/navn: Stig Myrland
	Telefon/e-postadresse/adresse: +47 907 91 444, stig.myrland@yara.com
Student: Karsten Sedal Slagstad	
Student: Aksel Digre Søbstad	
Student: Jon Ola Landgraff	
Student: Lars Markus Lerdahl	
Prosjekt-tittel/arbeidstittel	Fjernstyring av kjøretøy over 4G/5G
Prosjektnummer	E2118

Andre relevante dokumenter: Prosjektmanual Bacheloroppgaven.

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer.

1.
Studenten(e)/prosjektgruppen skal gjennomføre prosjektet i perioden fra januar 20xx til yy. mai 20xx.

Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU og oppdragsgiver yter veiledning. Oppdragsgiver stiller til rådighet kunnskap og materiale som vil kunne bidra til gjennomføringen av prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. NTNU skal stille til rådighet egen veileder. Oppdragsgiver plikter å gi en evaluering/sensur av prosjektet vederlagsfritt.

2.
Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
Oppdragsgiver og NTNU dekker hver sin del av den veiledningstid som gis. Dekning av reiser og opphold langt fra studiested dekkes enten av studentene eller av oppdragsgiver ut fra den part som er aktiv for at reise og opphold er nødvendig. Studentene dekker evt. utgifter for trykking og ferdigstillelse av den skriftlige besvarelsen vedrørende prosjektet med mindre oppdragsgiver yter slik bistand.

3.
Eiendomsrett
Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet inklusiv publisering. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og studentene bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. NTNU skal ha rett til vederlagsfri utnyttelse av besvarelsen og resultatene fra bachelorarbeidet til undervisnings- og forskningsvirksomhet inklusive publisering. Dette gjelder også data som underbygger resultatet i besvarelsen med mindre det vil være i strid med lov/forskrift eller godkjennelser som er gitt av Regional komité for medisinsk og helsefaglig forskningsetikk (REK), Norsk samfunnsvitenskapelig datatjeneste (NSD) eller andre institusjoner.

Hvis kandidaten skal utføre forskningsprosjektet som del av et større prosjekt, gjelder det som er avtalt om IP-rettigheter i dette prosjektet. Dette beskrives her:

NTNU

FAKULTET FOR INFORMASJONTEKNOLOGI OG ELEKTROTEKNIKK

AVTALE

4.
Hvis arbeidet medfører publisering og studentenes bidrag tilfredsstiller Vancouver-konvensjonens krav til medforfatterskap, skal studentene oppføres som medforfattere. Dersom bidraget deres ikke tilstrekkelig for medforfatterskap, skal de anerkjennes for bidraget.

5.
NTNU står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av faglærer/veileder og sensor.

6.
Offentliggjøring.
Papirkopi av besvarelsen registreres og plasseres i et åpent arkiv ved instituttet. Oppdragsgiver kan ved prosjektstart kreve at prosjektet skal behandles som *lukket prosjekt* det vil si ikke publiseres eller plasseres i det åpne arkivet dersom dette kan begrunnes i lov eller forskrift eller ut fra kommersielle hensyn. I tilfelle av lukket prosjekt, skal allikevel besvarelsen normalt kunne publiseres og plasseres i åpent arkiv etter en på forhånd avtalt periode, som normalt ikke skal overskride 3 år.

7.
Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

8.
Taushetserklæring
Ved underskrivelse av denne avtalen erklærer studentene ved sin underskrift alminnelig taushetsplikt vedrørende tekniske innretninger, fremgangsmåter, drifts eller forretningsforhold hos oppdragsgiver som det er av betydning å behandle konfidensielt.

9.
Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift etter LOV 2004-05-14 nr. 25: Lov om voldgift.

10.
Denne avtalen utferdiges med et eksemplar til hver av partene. Signert dokument godtas på pdf-fil. På vegne av NTNU er det intern veileder som godkjenner avtalen.

11. Annet

12.
Signaturer

Dato/ Veileder NTNU Institutt for elektroniske systemer/elkraft/teknisk kybernetikk 22/01-21 Cuong Phu Le
Dato/Oppdragsgiver/kontaktperson 25/1-21 Stig Myrland.
Dato/Student 20.01.21 Karsten Sedal Slagstad Karsten S. Slagstad
Dato/Student 20.01.21 Aksel Digre Søbstad
Dato/Student 20.01.21 Jon Ola Landgraff
Dato/Student 20.01.21 Lars Markus Lerdahl Lars M Lerdahl

Vedlegg B

Raspberry Pi spesifikasjoner

B.1 Raspberry Pi 3B+

Spesifikasjoner for Raspberry Pi 3B+:

- Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- 1GB LPDDR2 SDRAM
- 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
- Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)
- Extended 40-pin GPIO header
- Full-size HDMI
- 4 USB 2.0 ports
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- 4-pole stereo output and composite video port
- Micro SD port for loading your operating system and storing data
- 5V/2.5A DC power input
- Power-over-Ethernet (PoE) support (requires separate PoE HAT)

B.2 Raspberry Pi 4B

Spesifikasjoner for Raspberry Pi 4B (8GB versjonen):

- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- 8GB LPDDR4-3200 SDRAM
- 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
- Gigabit Ethernet
- 2 USB 3.0 ports; 2 USB 2.0 ports.
- Raspberry Pi standard 40 pin GPIO header (fully backwards compatible with previous boards)
- 2 × micro-HDMI ports (up to 4kp60 supported)
- 2-lane MIPI DSI display port
- 2-lane MIPI CSI camera port
- 4-pole stereo audio and composite video port
- H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode)
- OpenGL ES 3.0 graphics
- Micro-SD card slot for loading operating system and data storage
- 5V DC via USB-C connector (minimum 3A*)
- 5V DC via GPIO header (minimum 3A*)
- Power over Ethernet (PoE) enabled (requires separate PoE HAT)
- Operating temperature: 0 – 50 degrees C ambient

B.3 Installerte pakker, Raspberry Pi

- gstreamer1.0-plugins-base
- gstreamer1.0-plugins-good
- gstreamer1.0-plugins-bad
- gstreamer1.0-plugins-ugly
- gstreamer1.0-libav
- gstreamer1.0-doc
- gstreamer1.0-tools
- gstreamer1.0-omx-rpi (kun Raspberry Pi 3)
- libgstreamer1.0-0
- ufw
- wireguard

Vedlegg C

Nettverkskode

C.1 Klient - Pilot

```
//lars
#include <G29_wheel_input.h> //importerer class
unsigned int main_microsecond = 1000000; //brukes ikke

#define BRAKE_CODE 3
#define ACCELERATION_CODE 2
#define WHEEL_CODE 0
#define CLUTCH_CODE 1
#define MINUS_BTN_CODE 20
#define PLUSS_BTN_CODE 19
#define accInk 10
#define wheelInk  5

int preData;
int preButtonValue;
unsigned long preMillis;//deklarerer pre_millis
unsigned long delta_t;
const char  codeNames[4] = {"WHEEL       ","CLUTCH       ","ACCELERATION","BRAKE       "};


/// lars
// denne versjonen er bygd på linux sine innebygde socket funksjon for c/c++
// læreматriell er https://www.youtube.com/watch?v=cNdlrbZSkyQ og koden er i hovedsak basert på denne men har
    blitt modifisert mye i løpet av prosjekt perioden
// lenke til orgial kode: https://gist.github.com/codehoose/d7dea7010d041d52fb0f59cbe3826036

#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <string.h>
#include <iostream>
#include <chrono>
#include <unistd.h> // getopt

///
using namespace std;
//#define JOY_DEV "/dev/input/js0" //Skrevet om bane for å matche den oppdaterte filbanen.

/


int main(int argc, char  argv[])
{
 char addr[20] = "78.158.241.23"; //standard adressen
 int kom; // for oppstarts kommandoer
 int PORT = 54000;  // setter standard port
```

```
   std::cout << "[Klient] ingen argumenter port: " << PORT << "ip: " << addr << endl;
  }

}

/ Wheel setup /
JoystickInput G29;
std::cout<<"                                                    "<<std::endl;
std::cout<<"                                                    "<<std::endl;
std::cout<<"[Wheel setup]\t Oppsett for ratt.\n[Ratt oppsett]\t Venligst vendt..."<<std::endl;
G29.setup();
G29.setWheelRange(180);
G29.readEvent(); //Oppdaterer rattverdier.

usleep(1    main_microsecond);//sleeps for 3 second//Delay her//Delay  sekund
std::cout<<"[Ratt oppsett]\t Ferdig!"<<std::endl;
/ Wheel setup END /

/ Client setup /
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) // oppretter socketen
{
 std::cout << ("\n [Klient]\t Kunne ikke opprette socket ERROR! \n") << ::endl;
 return -1;
}

serv_addr.sin_family = AF_INET; // setter sin_family verdien til AF_INET for IPv4
serv_addr.sin_port = htons(PORT); // gjør om port til nettverksbyte

//inet_pton - convert IPv4 and IPv6 addresses from text to binary
if (inet_pton(AF_INET, addr, &serv_addr.sin_addr) <= 0) //inet_pton for å gjøre om ipadresse strengen til binær
    verdi
{
 std::cout << ("\n[Klient]\t inet_pton feilet sjekk om IP adressen er korrekt!\n") << ::endl;
 return -1;
}

if (connect(sock, (struct sockaddr )&serv_addr, sizeof(serv_addr)) < 0) // kobler opp til tjeneren
{
 std::cout << ("\n[Klient]\t Oppkobling feilet! Avsluetter... \n") << ::endl;
 return -1;
}
std::cout<<"[Client]\t Koblet til: "<<addr<<std::endl;
std::cout<<"                                                    "<<std::endl;
std::cout<<"                                                    "<<std::endl;
std::cout<<"_____________________________________________________________________________________________
    _______"<<std::endl;
std::cout<<"Ettersending av data hvert: 30ms"<<std::endl;
std::cout<<"\t|Tid siden siste sendte verdi [ms]|\t|sendt verdi|\t|sendt kode|"<<std::endl;
data[0] = 90;
data[1] = 0;
for (;;)
{
 //std::cout<<"Waiting on server ready...                                  ."<<std::endl;
 //read(sock, &dataRecived, sizeof(int)); //Mottatt data fra server: >NOE RART MED DENNE NÅ. rein klient-serverl
     øsning funker denne, men ikke nå.
 //std::cout << dataRecived << std::endl;
 //if (1 == 1) { //her skal egentlig dataRecived variabelen stå, men det funker ikke atm. tror problemet ligger
     på serversiden.
 //std::cout << dataRecived << std::endl;

  if (1==1)
  {

   G29.readEvent(); //Oppdaterer rattverdier.
   data[0] = G29.eventValue(); //leser inn fra ratt
```

```
data[1] = G29.eventCode(); //leser inn fra ratt

delta_t =(std::chrono::system_clock::now().time_since_epoch() /
std::chrono::milliseconds(1))-preMillis;

if (data[0]!=preData || ( (data[1] == 20) || (data[1] == 19) ))//For sending av
 {


  / PEDAL-FILTER /
  if (data[1] == BRAKE_CODE ||
   data[1] == ACCELERATION_CODE ||
   data[1] == CLUTCH_CODE)
   {
    if  ((data[0]>preData+accInk)||(data[0]<preData-accInk)
     ||( (data[0]==0)||(data[0]>252) ) )  //Send alltid 0-verdi minst 1 gang. PEDAL
       //send toppverdi minst engang. PEDAL

     {
      send(sock,&data, 2 sizeof(int), 0); //sender arrayet data.
      preData = data[0]; //sammenligningsvariabel. Brukes til å sammenligne ny og gammel ratt_value.
      preMillis= (std::chrono::system_clock::now().time_since_epoch() /
      std::chrono::milliseconds(1));

      cout<<"\r["<<delta_t<<"]  \t["<<codeNames[ data[1] ]<<"]\t[Verdi]:"<<data[0]<<"  \t[Kode]:\t"<< data
          [1]<<"        "<<std::flush;


     }
   }
   / RATT-FILTER /
   / Filtrerer bort noen verdier slik at
    hver
   /
  if (data[1] == WHEEL_CODE)
   {
    if((data[1]==0) ||  //Send bunnverdi ratt
    (data[0]==180)  ||  //send toppverdi ratt
    (data[0]==90) || //send midtstilt ratt.
    ((data[0]>80 && data[0] < 100 )) //send alle verdier 10 over eller under midtstilt
    )
     {
      send(sock,&data, 2 sizeof(int), 0); //sender arrayet data.
      preData = data[0]; //sammenligningsvariabel. Brukes til å sammenligne ny og gammel ratt_value.
      preMillis= (std::chrono::system_clock::now().time_since_epoch() /
      std::chrono::milliseconds(1));
      cout<<"\r["<<delta_t<<"]  \t["<<codeNames[ data[1] ]<<"]\t[Verdi]:"<<data[0]<<"  \t[Kode]:\t"<< data
          [1]<<"        "<<std::flush;

     }
   }




 }//if !preData
 / Sender data hvert 100ms for å kunne legge inn failsafe på bilen /
 delta_t = ( (std::chrono::system_clock::now().time_since_epoch() /
    std::chrono::milliseconds(1))-preMillis);
 if(delta_t >30)
 {
  send(sock,&data, 2 sizeof(int), 0); //sender arrayet data.
  preMillis = (std::chrono::system_clock::now().time_since_epoch() /
    std::chrono::milliseconds(1));
```

C.2 Tjener - Bil

```
/
Server med SPI kommunikasjon.
 Server sender en ready to recv til klienten, som igjen svarer med å sende data fra ratt/pedal.
 dataen skrives så ut på SPI.
Repeat.
 /

// denne versjonen er bygd på linux sine innebygde socket funksjon for c/c++
// læreматriell er https://www.youtube.com/watch?v=cNdlrbZSkyQ og koden er i hovedsak basert på denne men har
    blitt modifisert mye i løpet av prosjekt perioden
// lenke til orgial kode: https://gist.github.com/codehoose/020c6213f481aee76ea9b096acaddfaf

#include <stdio.h>
#include <signal.h>
#include <sys/ioctl.h>
#include <iostream>
#include <sys/types.h>
#include <sys/unistd.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <string.h>
#include <string>
#include <unistd.h> // getopt


/ SPI includes /
#include <linux/spi/spidev.h>
#include <fcntl.h>
#include <cstring>
#include <unistd.h> //sleeep
#include <chrono>// Millis timer
/ SPI includes END /
using namespace std;


/ Timer variabler. /
int milliWait = 10; //defineres en gang.
unsigned long preMillis;//deklarerer pre_millis
unsigned long delta_t;//deklarerer pre_millis
unsigned long preMicro;//deklarerer pre_millis
const char  codeNames[4] = {"WHEEL       ","CLUTCH       ","ACCELERATION","BRAKE       "};
//Variabel SPI
int fd;

//SPI funksjoner
int spiTxRx(unsigned char txDat); //Deklarerer funksjonen.
void sendCommand(int j, int k);

/  FUNKSJON FOR CLEAN EXIT NÅR PROGRAMMET LUKKES VIA CTRL+C /
volatile sig_atomic_t stop;

void interruptHandler(int signum)
{
    stop = 1;
}
```

```
int main(int argc, char  argv[])
{
 / SPI Initialisering /
 fd = open("/dev/spidev0.0", O_RDWR);
 unsigned int speed = 1000000; //1Mhz clock på overforinga.
 ioctl (fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed); //Setter hastighet, IOCTL. usikker. må sjekkes ut nærmere. bruke
     sogså i spiTxRx funksjonen.
 / SPI Initialisering END /

    int PORT = 54000;
    char addr[20] = "0.0.0.0";
    int kom;

    while ((kom = getopt(argc, argv, "i:p:")) != -1)
    {
        switch (kom)
        {
        case 'i':
            strcpy(addr, optarg);
            std::cout << "Ip adresse gitt: " << optarg << endl;
            break;
        case 'p':
            PORT = atoi(optarg);
            std::cout << "port gitt: " << optarg << endl;
            break;
        default:
            std::cout << "ingen argumenter port: " << PORT << " ip: " << addr << endl;
        }

    }

 struct timeval timeout; //Timeout for ctrl+c
    timeout.tv_sec = 10;
    timeout.tv_usec = 0;

    // oppretter lytte socketen IPv4/TCP
    int lytteSock;
    if ((lytteSock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        cout << ("\n [Klient]\t Kunne ikke opprette socket ERROR!\n") << endl; //????????????
        return -1;
    }

    // beskriver hrovrdan serveren skal fungere, lytte adresse, IPv4/TCP, PORT
    sockaddr_in serv_addr;
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);
    inet_pton(AF_INET, addr, &serv_addr.sin_addr);

    //binder lytesocketen
    bind(lytteSock, (sockaddr )&serv_addr, sizeof(serv_addr));
    cout << "[SERVER]Tjener lytter..." << endl;

  //begynner å lytte
    listen(lytteSock, SOMAXCONN);


    sockaddr_in klient;
    socklen_t klientSize = sizeof(klient);
    // aksepterer klienten og binder informasjonen til klientSock
    int klientSock = accept(lytteSock, (sockaddr )&klient, &klientSize);

    char host[NI_MAXHOST];
```

```
    char service[NI_MAXSERV];

    memset(host, 0, NI_MAXHOST);
    memset(service, 0, NI_MAXSERV);

    //Printe info fra den oppkoblede klienten, port og ip
    if (getnameinfo((sockaddr )&klient, sizeof(klient), host, NI_MAXHOST, service, NI_MAXSERV, 0) == 0)
    {
        cout << host << "[SERVER]oppkobblet␣på␣port␣" << service << endl;
    }
    else
    {
        inet_ntop(AF_INET, &klient.sin_addr, host, NI_MAXHOST);
        cout << host << "[SERVER]oppkobblet␣på␣port␣" << ntohs(klient.sin_port) << endl;
    }

if (setsockopt (klientSock, SOL_SOCKET, SO_RCVTIMEO, (char  )&timeout, sizeof(timeout)) < 0)

    // Close listening socket
    close(lytteSock);

    // While loop: accept and echo message back to client
    int data[2];
int serv_i=0;
int SPI_milliWait = 1; //defineres en gang.
unsigned long SPI_preMillis;//deklarerer pre_millis
unsigned long SPI_delta_t;

    signal(SIGINT, interruptHandler); // FOR CLEAN EXIT CTRL+C

    while (!stop) //Går ut av løkka om den får signalet CTRL+C; programmet går til clean exit
    {

        //send(clientSocket, &readyData,sizeof(int), 0); //klar for å mota
        // Wait for client to send data
        int recvData = recv(klientSock, &data, 2 sizeof(int), 0); //Denne blokkerer videre kjøring frem til
            mottatt data.
        if (recvData == -1)
        {
            cout << "Timeout.␣kontinue.." << endl;
            continue;
        }

        if (recvData == 0)
        {
            cout << "[SERVER]Klienten␣avsluttet␣tilkoblingen!␣" << endl;
            break;
        }
            sendCommand(data[0], data[1]);

        //cout<<serv_i<<"\t[SERVER]\t[data[0]]Value/SPI:"<<data[0] << "\t [data[1]]Code: "<< data[1]  << '\t' <<
            '\r' << std::flush; //

 //cout << serv_i << "\t" << data[1] << "\t" << data[0] << endl;
 cout<<"\r[SERVER]["<<serv_i<<"]""\t[Value]:"<<data[0]<<"␣␣\t[Code]:\t"<< data[1]<<"␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
     ␣␣␣␣␣␣␣␣␣"<<std::flush;

 //if(1==1){
 //}

 //SIMULERER I2C med sleep.
 //usleep(0.01   microsecond);//sleeps for #  microsecond
  //I2C goes here?
 serv_i++;
   }
```

```
    // Close the socket
    close(klientSock);
    cout<< "[SERVER]Programmet avsluttet vellykket. Socket ble stengt! "<<endl;
    return 0;
}

/ SPI definerer /
/
spiTxRx
 Transmits one byte via the SPI device, and returns one byte
 as the result.
 Establishes a data structure, spi_ioc_transfer as defined
 by spidev.h and loads the various members to pass the data
 and configuration parameters to the SPI device via IOCTL
 Local variables txDat and rxDat are defined and passed by
 reference.
                                                        /

int spiTxRx(unsigned char txDat) //Definerer funksjonen.
{

  unsigned char rxDat;

  struct spi_ioc_transfer spi;

  memset (&spi, 0, sizeof (spi));

  spi.tx_buf        = (unsigned long)&txDat;
  spi.rx_buf        = (unsigned long)&rxDat;
  spi.len           = 1;

  ioctl (fd, SPI_IOC_MESSAGE(1), &spi); //WHATS GOING ON HERE?

  return rxDat;
}
/ SPI definerer END /


void sendCommand(int j, int k)
{
 unsigned char resultByte;

/
Send the parameters one byte at a time.
                                                        /

 resultByte = 'd';
 while(resultByte != 'a')
 {
  spiTxRx('c');
  usleep(10);

  spiTxRx(k);
```

Vedlegg D

Arduinokode

D.1 Kode på mikrokontroller

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <pins_arduino.h>
#include <stdio.h>
#include <SPI.h>
 / INCLUDES END /


//#define F_CPU 16000000UL
#define PWM_PRESCALE 16  // Timer 1 clock prescaler

// Periodetiden på PWM
#define PWM_PERIODE  F_CPU/1000000 10000/PWM_PRESCALE

// Maksimalverdi på servo
                                //#define SERVO_MAX   F_CPU/1000000 1900/PWM_PRESCALE //1900
#define SERVO_MAX PWM_PERIODE 0.19
// Minsteverdi på servo
                                //#define SERVO_MIN   F_CPU/1000000 1100/PWM_PRESCALE //1100
#define SERVO_MIN PWM_PERIODE 0.11
// Nullverdi på servo
#define SERVO_NULL SERVO_MIN+((SERVO_MAX-SERVO_MIN)/2) //1500

// Maksimalverdi på ESC
                                //#define ESC_MAX   F_CPU/1000000 2000/PWM_PRESCALE //2000
#define ESC_MAX   PWM_PERIODE 0.1
// Minsteverdi på ESC
                                //#define ESC_MIN   F_CPU/1000000 1000/PWM_PRESCALE //1000
#define ESC_MIN   PWM_PERIODE 0.20
// Nullverdi på ESC
#define ESC_NULL ESC_MAX+((ESC_MIN-ESC_MAX)/2) //1500


 / Definisjoner code /

//Beskrivende variabler for de forskjellige mottatte codes.
 / Definisjoner code /
#define BRAKE_CODE 3
#define ACCELERATION_CODE 2
#define WHEEL_CODE 0
#define CLUTCH_CODE 1
#define MINUS_BTN_CODE 20
#define PLUSS_BTN_CODE 19
#define KNAPP_3
#define KNAPP_4
 / Definisjoner code END /
```

```
/
  OUTPUT PINS
             /
/ PWM pins /
#define PWM_SERVO 9
#define PWM_ESC 10
/ SPI pins /
#define MOSI 11
#define MISO 12
#define SCK 13
/ LED pins /
#define LED2 7 //Indikatorlys.
/ SPEAKER PIN /
#define MUSICPIN 3 // PWM-pinne
/ OUTPUT PINS END /



//Gjør om input verdier fra ratt/pedaler til verdier som passer til PWM signalet.
float wheelMultiplier = 4.55556; //Differansen mellom SERVO_MAX og SERVO_MIN delt på 180 inputverdier
float throttleMultiplier = 1.9608; //Differansen mellom ESC_MAX og ESC_NULL delt på 255 inputverdier
float gearing = 1;
bool brakeCheck = true;
bool reverseCheck = true;

volatile char dat; //Variabel for innlesning av SPI data
byte marker = 0; //Teller for å plassere riktig inputverdi i riktig variabel
char SPI_START_BYTE = 'c'; //Første byte i SPI-overføringen. Kontroll-variabel for å unngå feil-innlesning.

volatile int reccived_array[2]; //Mottaksarray
volatile int last_reccived_value; //lagrer siste mottatte verdi, for sammenligning.
unsigned long i = 0; //Teller brukt til testing
unsigned long j = 0; //Teller til failsafe

/ FAILSAFE MED COUNTER /
unsigned long int counter;
unsigned long int maxCount= 100000;
/ FAILSAFE END /

/ For å holde LED som varsler om frisk data høy lenge nok til å vises /
/
unsigned long int t0_led = millis();
unsigned long int t1_led = 0;
unsigned long int delta_led=0;
 /

/ Deklarerer funksjon / //Burde gjøres med alle funkjosnene som er definert.
void handleData(long int code, long int value);
void spiHandler();
void setup_done_LED(int led_pin);
void melodyPlayer(int speakerPin);
void testServo_ESC();
```

```
/ SETUP registersetting og start av seriell kommunikasjon /
void setup()
{

  pinMode(MUSICPIN,OUTPUT);
  //melodyPlayer(MUSICPIN); //MELODY JUST FOR FUN

  //Setter PE0, 1 og 2 som SPI pins
  PORTMUX.TWISPIROUTEA |= PORTMUX_SPI0_ALT2_gc;

  pinMode(LED2, OUTPUT); //LED-pinne output.
  pinMode(PWM_SERVO, OUTPUT); //PWM på pinne 9
  pinMode(PWM_ESC, OUTPUT); //PWM på pinne 10
  pinMode(MOSI, INPUT); //MOSI som input på pinne 11
  pinMode(MISO, OUTPUT); //MISO som output på pinne 12
  pinMode(SCK, INPUT); //SCK som input på pinne 13

  //Setter pinne 9 og 10 som output for TCA0 PWM, PWM i singleslope mode
  TCA0.SINGLE.CTRLB |= TCA_SINGLE_CMP0EN_bm | TCA_SINGLE_CMP1EN_bm | TCA_SINGLE_WGMODE_SINGLESLOPE_gc;

  TCA0.SINGLE.PERBUF = PWM_PERIODE; //Setter periodetiden på TCA0 PWM, skal være 100Hz
  //Div klokke med 8 for rikitg periodetid på PWM, Skrur på Telleren
   TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV16_gc | TCA_SINGLE_ENABLE_bm;

  / ESC INTITIALVERDI /
  TCA0.SINGLE.CMP1BUF = ESC_NULL;//Setter startverdi på pinne 10

  / SERVO INITIALVERDI /
  TCA0.SINGLE.CMP0BUF = SERVO_NULL;//Setter startverdi på pinne 9
  delay(250);

  //SPI aktivert, SPI slave-mode.
  SPI0.CTRLA =  SPI_ENABLE_bm & (~SPI_MASTER_bm);


  Serial.begin(115200); //Starter serial port
  setup_done_LED(LED2); //OBS. Denne delayer koden litt før start, men greit å ha en indikator på oppstart.

  //Aktiverer IE flag. interrupt. Gjøres etter at alt er initialisert og startet.
  SPI0.INTCTRL = SPI_IE_bm;

  sei(); //Skrur på globale interrupts

  / Tester servo side til side og motor BIP /
  / Testen ligger bak interrupt, og kan derfor avbrytes. /
  testServo_ESC(); //TAR 3 SEKUNDER
  / Test ferdig... /
}


void loop()
{

    digitalWrite(LED2,LOW);//Setter nydata-LED lav.

    maxCount = 20; //Teller hver itterasjon av hovedloop...

    delay(10);
    counter++; //For hver itterasjon av koden.
    //Serial.println(counter);
    if(counter>maxCount)
    {
      //Kunne lagt til bremsing, men er risikabelt for bakoverkjøring ved avbrudd i kodekjøring..
      TCA0.SINGLE.CMP1BUF = ESC_NULL; //Stopper motoren
       Serial.println("FAILSAFE_UTLOST                                                    ");
```

```
      counter = 0; //Restarter counter. denne restartes også ved mottak av valid SPI-data.
    }
}

/ INTERRUPT RUTINE
 Definerer hva som skal skje ved interrupt.
 SPI-kommunikasjonen er interruptbasert, og sending starter alltid med interrupt
 /
ISR (SPI0_INT_vect)
{

  spiHandler(); //Kjører funksjon som leser in data for hver byte mottatt
  SPI0.INTFLAGS = SPI_IF_bm; //Resetter interruptflag
  if (reccived_array[0]!=last_reccived_value)
  {
  digitalWrite(LED2,HIGH);//Høy på utgangen ved interrupt.
  last_reccived_value = reccived_array[0];
  //t0_led = millis();
  }
}


void spiHandler()
{
  switch (marker)
  {
    case 0:
      dat = SPI0.DATA; //leser inn data fra SPI-dataregister.
      if (dat == SPI_START_BYTE) //Starten av all overføring av CODE og VALUE med START_BYTE. [START_BYTE,CODE,
          VALUE]
      {
        marker++;// ker  marker slik at neste case utløses.
      }
      break;

    case 1: //LESER INN CODE
      reccived_array[1] = SPI0.DATA; //CODE innlesing fra SPI-dataregister.
      marker++; // ker  marker slik at neste case utløses.
      break;

    case 2://LESER INN VALUE
      reccived_array[0] = SPI0.DATA; //VALUE innlesning fra SPI-dataregister.
      if (dat != reccived_array[0] && dat != reccived_array[1]) //Sjekker om feil verdi er lest inn i CODE eller
          VALUE
      {
        / Printer mottatt data ut til seriellporten. /
        Serial.print(i); //Teller antall mottatte verdier
        Serial.print("\r\t Code: "); Serial.print(reccived_array[1]); //Skriver ut mottatt code.
        Serial.print("\t Verdi: "); Serial.println(reccived_array[0]); //Skriver ut mottatt verdi.
        i++; //Teller brukt til testing

        //j = 0; //Resetter teller for failsafe
        //t0_safe = millis();//Resetter teller for failsafe.
        counter = 0;
        / Hånterer mottatte verdier med handleData(CODE,VALUE) /
        handleData(reccived_array[1], reccived_array[0]);

      }

      else //Funnet feil på CODE
      {
        Serial.println("[SPI ERROR] Feil på mottatt verdi");
      }

      marker = 0; //Setter marker tilbake til null, slik at avlestning av SPI-data begynner på nytt igjen.
      break;
```

```
  }
}

void handleData(long int code, long int value) {

  switch (code)
  {
    //ratt
    case WHEEL_CODE:
      TCA0.SINGLE.CMP0BUF = SERVO_MIN + value   wheelMultiplier; //Skriver PWM signal ut på pinne 9 (
          Styringsservo)
      break;

    //Clutch
    case CLUTCH_CODE:
      TCA0.SINGLE.CMP1BUF = ESC_NULL + value   throttleMultiplier   gearing; //Skriver PWM signal ut på pinne 10
          (ESC)
      brakeCheck = false; //Deaktiverer bremsepedalen
      break;

    //gass
    case ACCELERATION_CODE:
      TCA0.SINGLE.CMP1BUF = ESC_NULL - value   throttleMultiplier   gearing; //Skriver PWM signal ut på pinne 10
          (ESC)
      brakeCheck = true; //Aktiverer bremsepedalen
      break;

    //brems
    case BRAKE_CODE:
      if (brakeCheck){
        TCA0.SINGLE.CMP1BUF = ESC_MIN; //Skriver PWM signal ut på pinne 10 (ESC)
        brakeCheck = false; //Deaktiverer bremsepedalen
      }
      break;

    case PLUSS_BTN_CODE:
      gearing = 1;
      break;

    case MINUS_BTN_CODE:
      gearing = 0.50;
      break;
  }
}

void setup_done_LED(int led_pin)
{
  for (int timer = 0;timer<500;timer+=50)
  {
    digitalWrite(led_pin,HIGH);
    delay(timer);
    digitalWrite(led_pin,LOW);
    delay(timer);
  }
}

void testServo_ESC()
{

  TCA0.SINGLE.CMP0BUF = SERVO_MIN;
  delay(500);
  TCA0.SINGLE.CMP0BUF = SERVO_MAX;
  delay(500);
  TCA0.SINGLE.CMP0BUF = SERVO_NULL;
  delay(1000);
  TCA0.SINGLE.CMP1BUF = ESC_NULL - 44   throttleMultiplier   gearing;
```

```
  delay(500);
  TCA0.SINGLE.CMP1BUF = ESC_NULL - 0   throttleMultiplier   gearing;
  delay(500);
  TCA0.SINGLE.CMP1BUF = ESC_NULL - 44   throttleMultiplier   gearing;
  delay(500);

  TCA0.SINGLE.CMP1BUF = ESC_NULL;
}
```

Vedlegg E

Glass-to-glass kode C Arduino

```
/*Programkode glass to glass testing */
/*BSc-gruppe E2118*/
#define signalPin 8
#define sensPin A0

float sensRead;
float lysTerskel=185; //Bruk kalibreringskoden for å finne verdi

int sigRead;


bool risingEdge = 0;
int lastSignal = 0;
int sensReady = 0;

unsigned long startTime;
unsigned long endTime;
unsigned long delta_t;

void setup() {
  // put your setup code here, to run once:
  pinMode(signalPin,INPUT);
  pinMode(sensPin,INPUT);
  Serial.begin(115200);
}

void loop() {
  sensRead = analogRead(sensPin);
  sigRead = digitalRead(signalPin);

  if(sigRead !=lastSignal)
  {

    //Serial.println("CHANGED");
    if((sigRead==1)&&(lastSignal==0)) //Sjekker om RISING EDGE.
    {

        //Serial.println("RISING EDGE");
        risingEdge = 1;
        startTime = millis();
        sensReady = 1;

    }
    else
    {
      risingEdge = 0;
    }
    lastSignal = sigRead;

  }

  if((sensRead>lysTerskel)&&(sensReady))
  {
    sensReady=0;
    //Serial.println("FIRST LIGHT MESSURE");
    endTime=millis();
    delta_t = endTime - startTime;
```

```
    Serial.print("latency:");Serial.println(delta_t);
  }




}void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Vedlegg F

Glass-to-glass kalibrer Arduino

```
/*Programkode for kalibrering av
*glass to glass testing
Kjøres først test for å finne lysterskel
*/
/*BSc-gruppe E2118*/
#define BAUDRATE 115200
#define sensPin A0
#define ledPin
float sensValue = 0;
float sensTerskel = 25;
float maxValue=1;
float minValue=9999;
unsigned long t0 = millis();
unsigned long t1 = millis();
unsigned long delta_t = 0;

void setup() {

  Serial.begin(BAUDRATE);
  Serial.println("Kalibreringsavlesning startet...");
  pinMode(sensPin,INPUT);
  //digitalWrite(ledPin,LOW); //En skriving digitalWrite tar 3,4ms ifølge
https://roboticsbackend.com/arduino-fast-digitalwrite/. kan måles. Vi kan også
heller bruke AVR-kode for optimalisering.

}

void loop() {
  sensValue = analogRead(sensPin);
  Serial.print("VALUE:");Serial.println(sensValue);
  //Serial.print("MIN:");Serial.println(minValue);
  //Serial.print("MAX:");Serial.println(maxValue);
  if (sensValue>sensTerskel)
  {


  }

  if (sensValue>=maxValue)
  {
    maxValue = sensValue;
    Serial.print("NEW MAX:");Serial.println(maxValue);
  }
  if (sensValue<=minValue)
  {
    if (sensValue>50) //Ignorere usannsynlige lave verdier.
    {
    minValue = sensValue;
    Serial.print("NEW MIN:");Serial.println(minValue);
    }
  }
  t1=millis();
  delta_t = t1-t0;

  Serial.println(delta_t/1000);
  Serial.println("Venligst vent....");
  if (delta_t>15000)
```

```
  {
  Serial.println("_____________________________");
  Serial.print("MIN:");Serial.println(minValue);
  Serial.print("MAX:");Serial.println(maxValue);
  }
  while(delta_t>15000)
  {
    //DO nothing
  }


}
```

Vedlegg G

Signal test Arduino

```
/*Programkode for signalatency testing*/
/*BSc-gruppe E2118*/
//Definerer pinneutganger
#define signalPin 8 //digital input kobles til PILOT
#define sensPin 7 //digital input Kobles til BIL
//Deklarerer variabler
int sigRead;    //for avlesning av signalPin
int sensRead; //for avlesning av sensPin

bool risingEdge = 0;  //Start tilstand 0
int lastSignal = 0;   //Start tilstand 0
int sensReady = 0;    //Start tilstand 0

//Variabler for måling av tid.
unsigned long startTime; //start tid. t0
unsigned long endTime;   //slutt tid. t1
unsigned long delta_t;   //delta_t, for tidsdifferanse.

void setup() {
  // put your setup code here, to run once:
  pinMode(signalPin,INPUT); //Setter pinne som inngang.
  pinMode(sensPin,INPUT);   //Setter pinne som inngang.
  Serial.begin(115200);     //Setter baud-rate til 115200. Høy baud for å
unngå forsinkelser.
}

void loop() {
  sensRead = digitalRead(sensPin); //Leser av sensor-pinne 2. Sluttsignal for
timer
  sigRead = digitalRead(signalPin); //Leser av sensor-pinne 1. Startsignal for
timer

  //Håndtering av SIGNAL t0.
  if(sigRead !=lastSignal) //ved startsignal for timer
  {
    if((sigRead==1)&&(lastSignal==0)) //Sjekker om RISING EDGE.
    {
        //Serial.println("RISING EDGE"); utkommentert for å unngå og bremse
programmet.
        risingEdge = 1; //Setter rising edge til 1. Boolsk True.
        startTime = millis();//micros();//millis();
        sensReady = 1; //Setter sensReady til 1. Boolsk True.

    }
    else
    {
      risingEdge = 0; //setter rising edge til 0. Boolsk false.
    }
    lastSignal = sigRead; //Mellomlagrer siste verdi for å sammenligne og
sjekke RISING EDGE.
  }

  if((sensRead==1)&&(sensReady)) //ved sluttsignal for timer. SIGNAL t1.
  {
    endTime=millis();//micros();//Setter sluttid for måling. t1.
    sensReady=0; //tilbakestiller slik at sensor-avlesning kun skjer hvis
rising-edge er satt.
```

```
    //Serial.println("FIRST SENSOR MESSURE"); utkommentert for å unngå og
bremse programmet.

    delta_t = endTime - startTime; //beregner tidsdifferansen og dermed
forsinkelsen i ms.
    Serial.print("latency:");Serial.println(delta_t); //skriver ut resultat av
måling.
  }
}
```

Vedlegg H

G29 wheel input header

```
///
/*
Versjon 2.0.0
Ferdig 11.03.2021
Utarbeidet av Lars Markus Lerdahl E2118
Bygget rundt https://archives.seul.org/linuxgames/Aug-1999/msg00107.html
Endringslogg for endringer etter 11.03.2021:->
11.03.2021>>  Void return på setup og read.
15.03.2021>>  rettet på void og kommenterte ut alle fprints.
04.04.2021>>  Reverserte deklarerte verdier som short ints da det ga feil
output. (blanding av int og short ints ble tull.
04.04.2021>>  La til to knappetrykk. Knappetrykk pluss og minus.
15.04.2021>>  Gjorde om setup til å returne verdi 1 ved suksess.
02.05.2021>>  Omstrukturering slik at det er en public og en private. samt
fjernet printf tilfordel for cout.
*/
/*
problemer som må utbedres:
-Problem
+løsning
- Filtering av knapper fungerer, men verdi for knapp leses inn uavhengig av om
den er definert eller ikke.
+ Løses ved å sette m_value =js.value inn i ifsetningene, istedenfor før if.
- Bruke else if, istedenfor if. eventuelt bruke CASE.
+
*/

/*###Burde få¨inn ifdef for å sjekke om disse er definerte fra før i koden
header-filen brukes i.*/
#include <stdio.h>
#include <stdlib.h> //calloc
#include <fcntl.h>
#include <unistd.h> //read.
#include <sys/ioctl.h> // IO - kontroll input/output control
#include <linux/joystick.h> //MÅ LASTES NED PÅ PI FØR BRUK
#include <iostream> //cout
#include <math.h>//pow for å finne kvadratrot.
#include <typeinfo>
#include<unistd.h>//delay:

///
//using namespace std; Kommentert ut 30.04.2021. Tror ikke den brukes noen
plass. Vi bruker heller std::cout.
//
class JoystickInput
{
public: /*Public variabler.*/
  #define JOY_DEV "/dev/input/js0" //Filbanen peker på default Linux USB-
inndata for joysticks. jsx.
  /*RATT-CODES definert*/
  #define WHEEL  0      //evdev code for rattrotasjon
  #define CLUTCH  1       //evdev code for clutchpedal
  #define ACCELERATOR 2     //evdev code for gasspedal/akseleratorpedal
  #define BRAKE 3       //evdev code for bremsepedal
  #define FIRST 5       //evdev code for ukjent verdi som kommer ved oppstart.
  #define MINUS_BTN_CODE 20   //evdev code for knappetrykk minus.
  #define PLUSS_BTN_CODE 19 //evdev code for knappetrykk pluss.
```

```
  int wheel_bit_resolution = 16; //Oppløsning på ratt og pedaltråkk
  int wheel_deg_max = 860;  //860 graders rotasjon. Logitech oppgir 900grader.
  int wheel_range = 180;    //180 graders utslag default. Hvor mye av rattets
rotasjonsområde vi velger å bruker

/*Private variabler.*/
private:
  // Diverse variabler
**************************************************************************
**************
  unsigned int microsecond = 1000000; //ett sekund i mirosekunder. brukes i
usleep.
  int oldRange,newRange; //deklarerer oldrange og newRange til senere bruk i
map-funksjonen.
  int m_value,m_code; // Deklarerer  variabler for utlesning av data. Return-
verdi fra funksjonene eventCode og eventValue.
  // Diverse
END*********************************************************************
************************

  //Wheel
values:*****************************************************************
***************************
   int m_wheel_max = (pow(2, 16) - 1) / 2;//fordi rattet går fra -32xxx til
32xxx
   int m_wheel_center = 0;// Gammel type
   int m_wheel_inc_deg = (m_wheel_max*2 / wheel_deg_max); //kalkulerer antall
grader rotasjon pr int-inkrement
   int m_wheel_range_int = m_wheel_inc_deg * wheel_range;   //Kalkulerer range
i skalert til returnverdi fra ratt.
   /*finner maksimalutslag i vær retning basert på wheel_range_int*/
   int m_Newwheel_min = m_wheel_center - m_wheel_range_int;
   int m_Newwheel_max = m_wheel_center + m_wheel_range_int; //Setter nye max
og min med default 360 graders total rotasjon
  //Wheel END
**************************************************************************
**********************

  //Variabler som brukes i avlesning av fil. Satt opp slik som i
linuxdokumentasjonen til joystick.h***************/
  int joy_fd, * axis = NULL, num_of_axis = 0, num_of_buttons = 0, x;
//Deklarerer flere variabler som int. og axis som en pointer-int
  char* button = NULL, name_of_joystick[80]; //Deklaerer char-pointer for
knapper og joystick-navn.
  struct js_event js; // lager en struckt med event. js_event arver struckt-
strukturen til js.
  //Avlesningsvariabler
END*********************************************************************
*************

//PRIVATE FUNKSJONER INNE I KLASSEN JoyStickInput.
private:

  int map(int oldMax, int oldMin, int newMax,int newMin)
  {
    //Remapping av verdier:
```

```
    oldRange = (oldMax - oldMin);   // Kalkulerer Område basert på argumenter
gitt til map()
    newRange = (newMax - newMin);   // Kalkulerer Område basert på argumenter
gitt til map()
    m_value = (((m_value - oldMin) *newRange)/oldRange)+newMin;//mapper om
m_value til nytt område basert på argumenter gitt.
    return m_value; //Returnerer ny m_value. return fra map brukes ikke, da
m_value er tilgjengelig i hele klassen.
  }

  void Wheel_rotation(void)
  {
    //m_Newwheel_min er en negativ verdi. grensene er signed int før
remapping.
    if (m_value < m_Newwheel_min)   // setter alle verdier over definertmaks
til maksverdi
    {
      m_value = m_Newwheel_min;   //redefinerer m_value
      //std::cout<<"[MIN]"<<std::endl;
    }
    //m_newwheel_max er en positiv verdi. grensene er signed int før
remapping.
    if (m_value > m_Newwheel_max)   // setter alle verdier over definertmaks
til maksverdi
    {
      m_value = m_Newwheel_max;   //redefinerer m_value
      //std::cout<<"[MAX]"<<std::endl;;
    }
    /*m_newwheel_max og min er høyeste verdi i bruk fra rattrotasjon. Disse
grensene er utledet fra og bestemt av wheel_range.*/
    map(m_Newwheel_max, m_Newwheel_min, 180, 0); //Mapper om verdiene til
grader. 0 til 180 grader unsigned 8-bit størrelse lastet inn i en 32-bit int-
variabel m_value.
  }

  void Pedal_Push8bit(void) //Endrer område for pedaltråkk
  {
    /*+-32657 er høyeste verdi returnert dynamiske value returnert fra
ratt/pedal-avlesning.*/
    map(-32767, 32767, 255, 0); //Mapper om data fra 16-bit signed til 8-bit
unsigned lagret i en 32-bit int-variabel m_value.
  }

/*PUBLIC FUNKSJONER INNE I KLASSEN JoyStickInput.*/
public:
/*Funksjon for oppsett av avlesning av det virituelle linux-filsystemet.*/
  int setup(void)
  {
    //Vi definerer joy_fd som open (js0,read only) og sjekker om den er -1 for
å se om fila ble åpnet eller ikke.
    if ((joy_fd = open(JOY_DEV, O_RDONLY)) == -1) //JOY_DEV er filbane til js0
i linux virituelle filsystem. O_RDONLY= read only. hvis return er -1, ble ikke
fila åpnet.
    {
      //std::cout<<("Couldn't open joystick. Check USB
connection")<<std::endl;
```

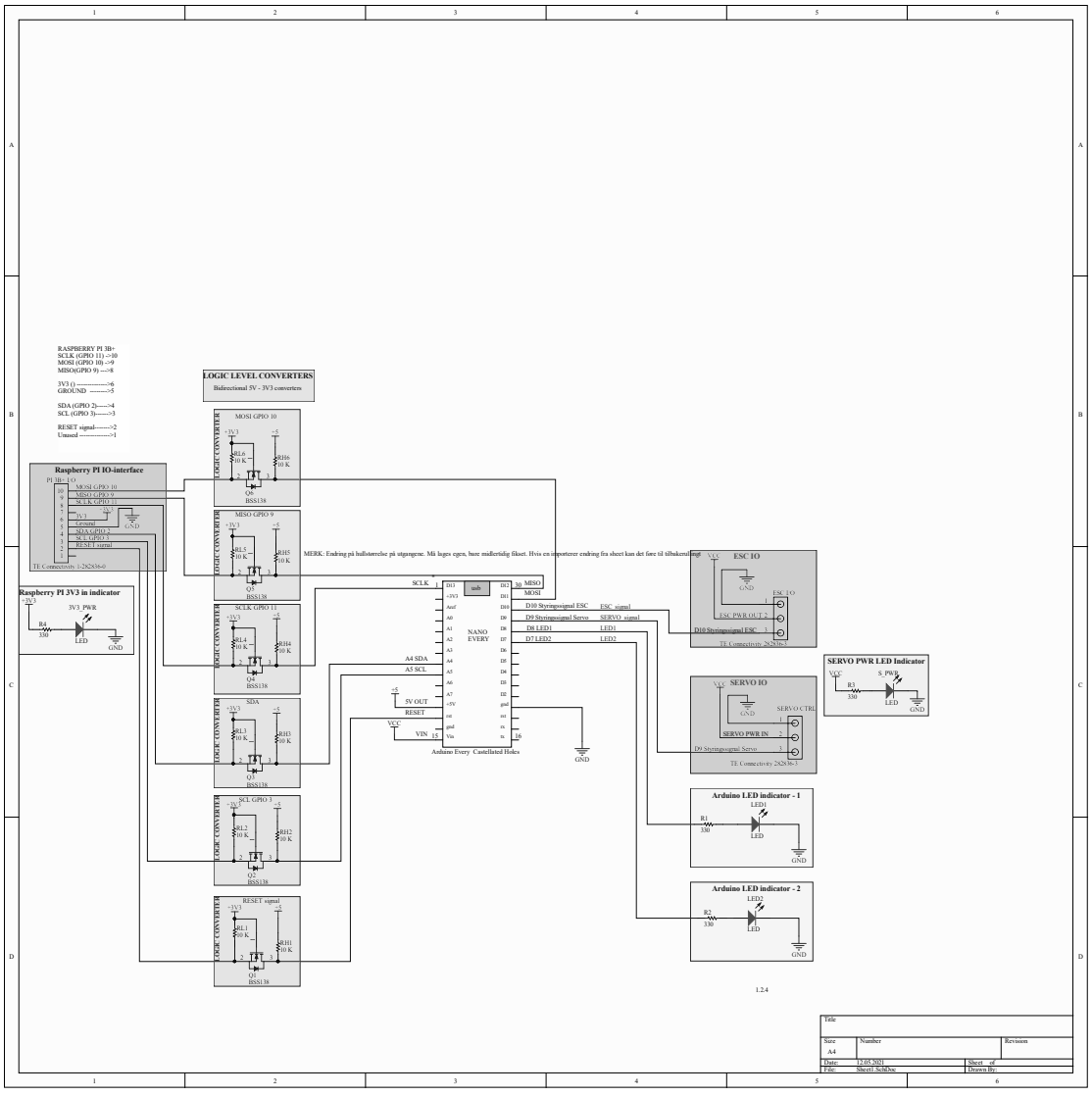
```
      std::cout<<("Kunne ikke åpne joystick-datafil. Sjekk USB-
tilkoblingen!")<<std::endl;
      return -1;
    }
    //Ved suksess av åpning av fil:
    ioctl(joy_fd, JSIOCGAXES, &num_of_axis);        //joy_fd er filen js0
åpnet.. JSIOCGAXES returnerer antall akser inn i num_of_axis
    ioctl(joy_fd, JSIOCGBUTTONS, &num_of_buttons);      //JSIOCGBUTTONS
returnerer antall buttons inn i num_of_buttons.
    ioctl(joy_fd, JSIOCGNAME(80), &name_of_joystick);     //JSIOGNAME
returerner identifier string, enhetsinformasjon/navn på enhet
    axis = (int*)calloc(num_of_axis, sizeof(int));      //calloc (size_t num,
size_t size); Allokerer minneplass til arrayet axis
    button = (char*)calloc(num_of_buttons, sizeof(char)); //calloc:Allocates a
block of memory for an array of num elements, each of them size bytes long,
and initializes all its bits to zero.

    /*Skriving til terminalvinduet.*/
    //std::cout <<"Joystick detected:"<<name_of_joystick<<std::endl;
    std::cout <<"Joystick detected:"<<name_of_joystick<<std::endl;  //skriver
ut til terminal: navn på joystick.
    //std::cout<<"Axis:"<<num_of_axis<<std::endl;
    std::cout<<"Axis:"<<num_of_axis<<std::endl;           //skriver ut til
terminal: Antall akser.
    //std::cout<<"Buttons:"<<num_of_buttons<<std::endl;
    std::cout<<"Buttons:"<<num_of_buttons<<std::endl;       //skriver ut til
terminal: Antall knapper.
    fcntl(joy_fd, F_SETFL, O_NONBLOCK); /* Setter avlesningen til ikke
blokkerende modus */
    return 1; //Setup returnerer 1 ved suksess

  }

/*Funksjon for avlesning av det virituelle linux-filsystemet.*/
  void readEvent(void)
  {
    //Pollingrate på USB-HID er 1000Hz. Nyquist-samplingsrate blir da 2000Hz,
eller hvert 0.5ms.
    usleep(0.0005 * microsecond);//0.5ms delay for å avlaste CPU. Kan byttes
ut med millis-timer for å unngå blocking, men ikke behov.
    read(joy_fd, &js, sizeof(struct js_event)); //leser in fra joy_fd inn i
structen js som har arvet struktur fra js_event fra joystick.h.
    //std::cout<<("  \r")<<std::endl; //Setter print pointer til start igjen.
skriver over printen fra forrige.
    fflush(stdout); //Flusher buffer for cout.

  }

  void setWheelRange(int deg_angel)
  {
    if (deg_angel <= wheel_deg_max) //Sjekker om gitt argument er utefor range
av rattets rotasjonsområde.
    {
      wheel_range = deg_angel; //setter ny wheel_range.
      std::cout <<"Nytt utslag satt til: "<<wheel_range<<std::endl;
    }
    else
```

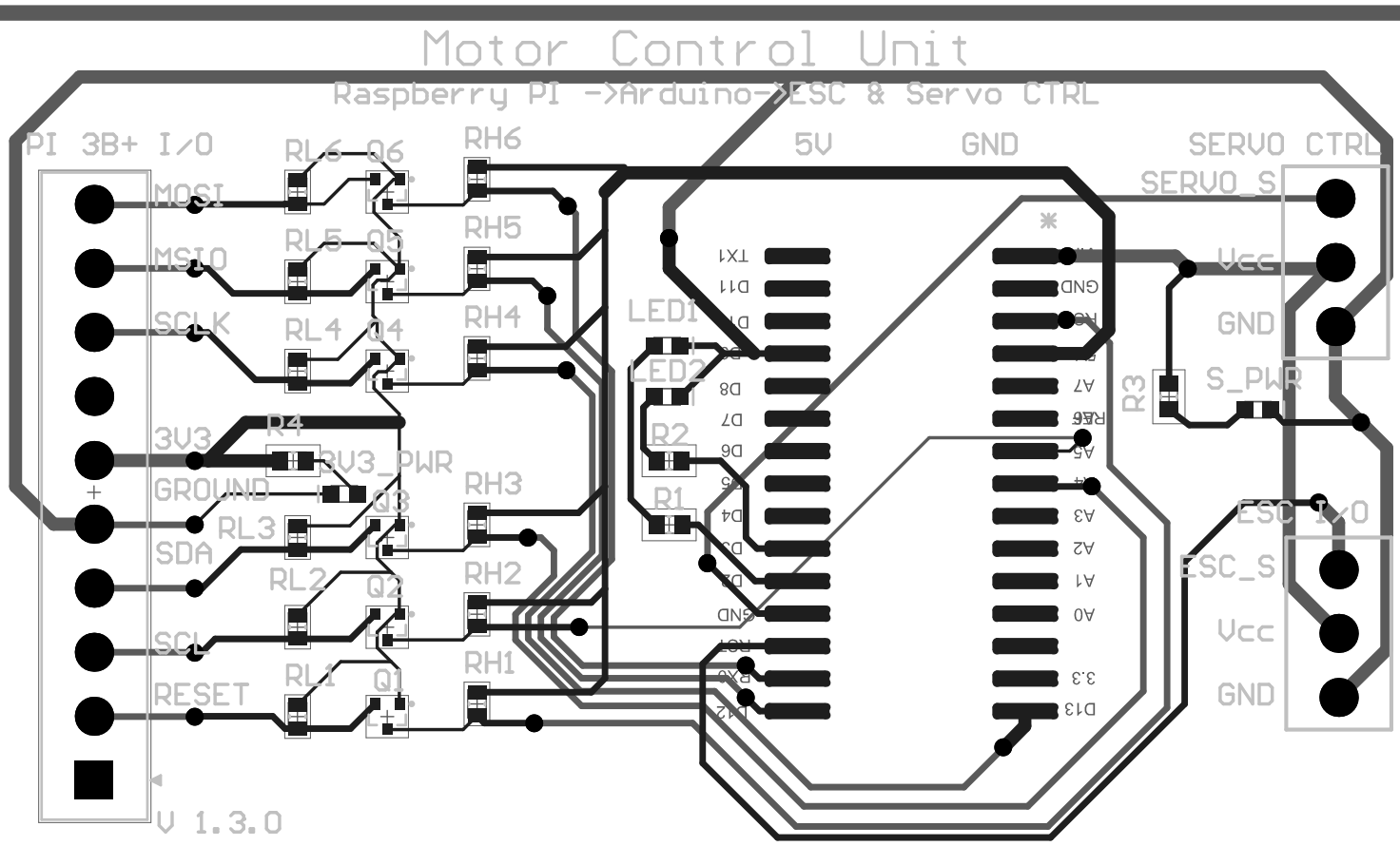
```
    {
      std::cout << "Vennligst velg ett valid omraade... (0-900). Omraade satt
til : 180 grader."<<std::endl;
      wheel_range = 180; //Setter område til default.
    }
  }

  int eventValue()
  {
    //Kanskje laste inn i m_value her, istedenfor i alle under. Ingen utslag
på kjøring, men færre linjer kode.
    switch (js.type & ~JS_EVENT_INIT)
    {
    case JS_EVENT_AXIS: //Ved case aksedata: dynamisk data. feks ratt eller
pedal.
      m_value = js.value;//laster js.value inn i privat variabel m_value, som
er tilgjengelig for hele klassen.

      if (js.number == FIRST)//Håndterer utventet verdi FIRST=5.
      {
        //do nothing...
      }
      //js.number er rattdata.
      if (js.number == WHEEL)
      {
        Wheel_rotation();
        //m_value konverteres ned til 8bit. 0- 180.
        //std::cout<<"[WHEEL]:"<<std::endl;
      }
      //js.number er clutchpedal-data
      if (js.number == CLUTCH)
      {
        Pedal_Push8bit();
        //m_value konverteres ned til 8bit. 0- 255.
        //std::cout<<"[CLUTCH]:"<<std::endl;
      }
      //js.number er Gasspedal-data
      if (js.number == ACCELERATOR)
      {
        Pedal_Push8bit(); //remapper verdi
        //m_value konverteres ned til 8bit. 0- 255.
        //std::cout<<"[ACCELERATOR]:"<<std::endl;
      }
      //js.number er bremsepedal-data
      if (js.number == BRAKE)
      {
        Pedal_Push8bit();
        //std::cout<<"[BRAKE]:"<<std::endl;
        //m_value konverteres ned til 8bit. 0- 255.
      }
      break; //Avslutter case.
    case JS_EVENT_BUTTON: //ved case knappetrykk.
      //Hvis js.number er ett av de to definerte knappetrykkene.
      if ((js.number==MINUS_BTN_CODE) || (js.number==PLUSS_BTN_CODE))
      {
        m_value = js.value; //laster verdi inn i m_value.
      }
```

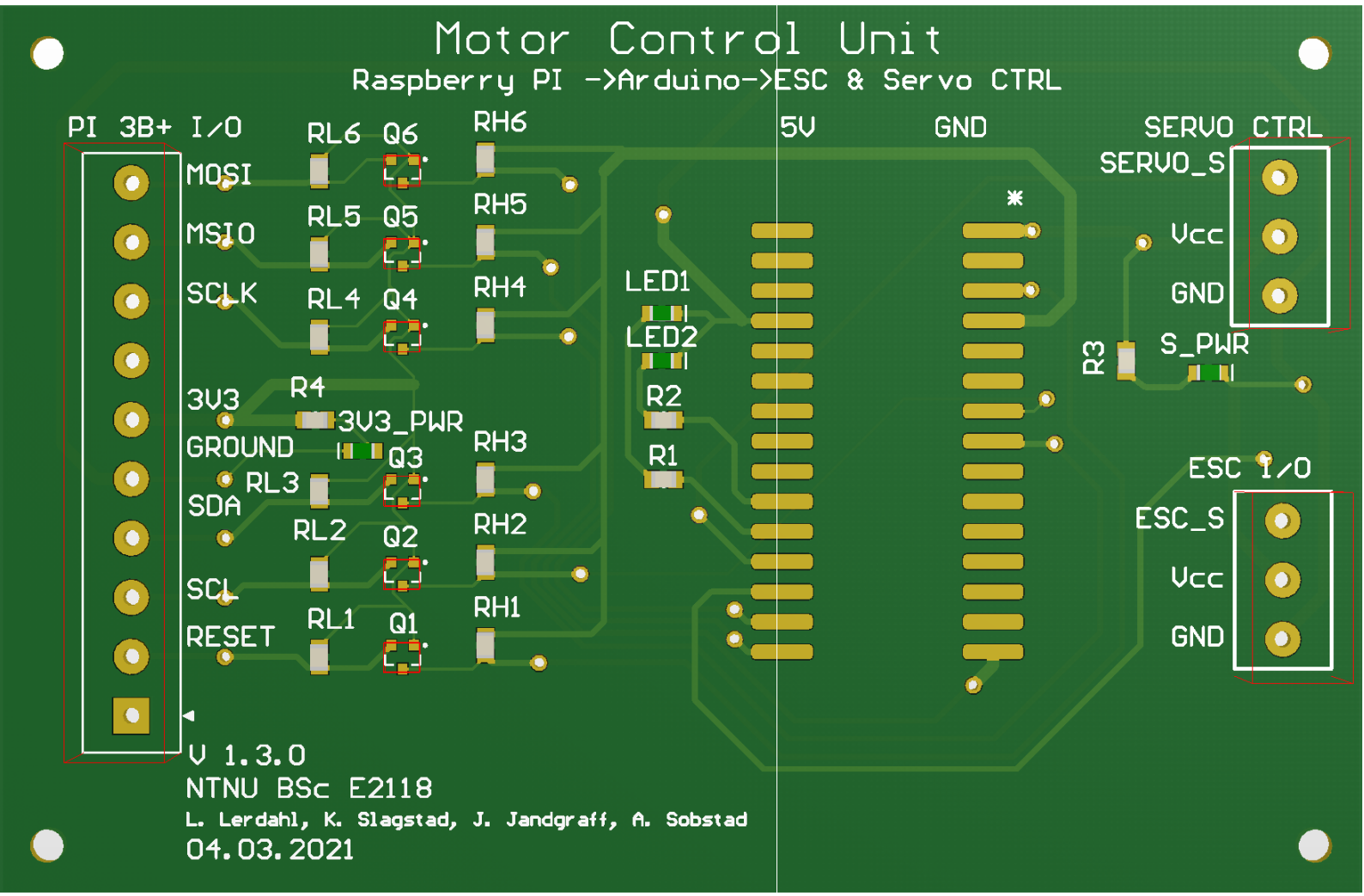
```
      else{m_value = m_value;} //ignorer verdi fra knapper som ikke er
definert.
      break; //Avslutter case.
    }
    return m_value;  //returnerer value.
  }

  int eventCode()
  {
    switch (js.type & ~JS_EVENT_INIT)
    {
    case JS_EVENT_AXIS: //Ved dynamisk data. feks ratt eller pedal.s
      m_code = js.number;

      return m_code; //returnerer code.
      break; //Avslutter case.

    case JS_EVENT_BUTTON: //Ved Boolsk data. Knappetrykk
    if ((js.number==MINUS_BTN_CODE) || (js.number==PLUSS_BTN_CODE))
    {
      m_code = js.number; //Laster inn knappeidentifikator. Enten
MINUS_BTN_CODE) || (js.number==PLUSS_BTN_CODE
    }
    else{m_code = m_code;}//ignorer code fra knapper som ikke er definert.
      return m_code; //returnerer code.
      break; //Avslutter case.
    }
  }
}; //avslutter klasse
//HEADER END.
```

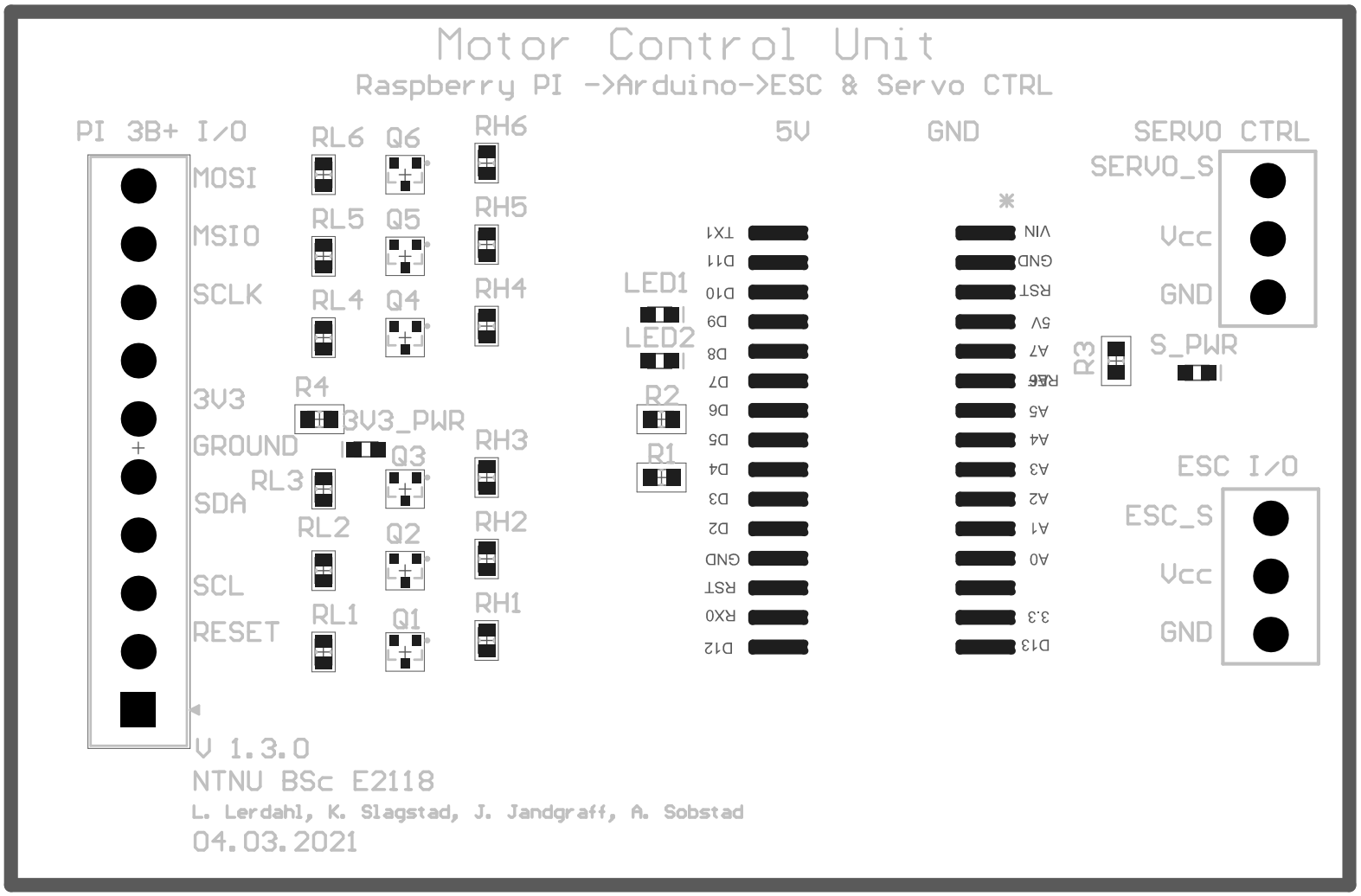
Vedlegg I

PCB - Skjematikk og utlegg


RASPBERRY PI 3B+
SCLK (GPIO 11) -->10
MOSI (GPIO 10) -->9
MISO(GPIO 9) --->8
3V3 (1 ---------->6
GROUND --------->5
SDA (GPIO 2)----->4
SCL (GPIO 3)----->3
RESET signal------>2
Unused ---------->1
LOGIC LEVEL CONVERTERS
Bidirectional 5V - 3V3 converters
Raspberry PI IO-interface
PI 3B+ I/O
MOSI GPIO 10
MISO GPIO 9
SCLK GPIO 11
3V3
Ground
SDA GPIO 2
SCL GPIO 3
RESET signal
TE Connectivity 1-2362536-0
Raspberry PI 3V3 in indicator
3V3_PWR
LED
LOGIC CONVERTER
MOSI GPIO 10
MISO GPIO 9
SCLK GPIO 11
SDA
SCL GPIO 3
RESET signal
BSS138
MERK: Endring på hallsterrelse på utgangene. Må lages egen, bare midlertidig fikset. Hvis en importerer endring fra sheet kan det føre til tilbakestilling
NANO EVERY
Arduino Every Castellated Holes
SCLK
MISO
MOSI
A4 SDA
A5 SCL
5V OUT
RESET
VIN
D10 Styringssignal ESC
D9 Styringssignal Servo
D8 LED1
D7 LED2
ESC_signal
SERVO_signal
LED1
LED2
ESC IO
ESC PWR OUT 2
D10 Styringssignal ESC
TE Connectivity 2362536-3
SERVO IO
SERVO CTRL
SERVO PWR IN
D9 Styringssignal Servo
TE Connectivity 2362536-3
SERVO PWR LED Indicator
S_PWR
Arduino LED indicator - 1
Arduino LED indicator - 2
1.2.4
Title
Size
A4
Number
Revision
Date: 12.05.2021
File: Sheet1.SchDoc
Sheet of
Drawn By:


Motor Control Unit
Raspberry PI ->Arduino->ESC & Servo CTRL
PI 3B+ I/O
MOSI
MSIO
SCLK
3V3
GROUND
SDA
SCL
RESET
RL6 Q6
RL5 Q5
RL4 Q4
R4
3V3_PWR
RL3 Q3
RL2 Q2
RL1 Q1
RH6
RH5
RH4
RH3
RH2
RH1
LED1
LED2
R2
R1
5V
GND
SERVO CTRL
SERVO_S
Vcc
GND
R3
S_PWR
ESC I/O
ESC_S
Vcc
GND
V 1.3.0
NTNU BSc E2118
L. Lerdahl, K. Slagstad, J. Jandgraff, A. Sobstad
04.03.2021


Motor Control Unit
Raspberry PI ->Arduino->ESC & Servo CTRL
PI 3B+ I/O
MOSI
MSIO
SCLK
3V3
GROUND
SDA
SCL
RESET
RL6
Q6
RH6
RL5
Q5
RH5
RL4
Q4
RH4
R4
3V3_PWR
Q3
RL3
RH3
RL2
Q2
RH2
RL1
Q1
RH1
LED1
LED2
R2
R1
5V
GND
SERVO CTRL
SERVO_S
Vcc
GND
R3
S_PWR
ESC I/O
ESC_S
Vcc
GND
V 1.3.0
NTNU BSc E2118
L. Lerdahl, K. Slagstad, J. Jandgraff, A. Sobstad
04.03.2021


Motor Control Unit
Raspberry PI ->Arduino->ESC & Servo CTRL
PI 3B+ I/O
MOSI
MSIO
SCLK
3V3
GROUND
SDA
SCL
RESET
RL6
Q6
RH6
RL5
Q5
RH5
RL4
Q4
RH4
R4
3V3_PWR
RL3
Q3
RH3
RL2
Q2
RH2
RL1
Q1
RH1
LED1
LED2
R2
R1
5V
GND
SERVO CTRL
SERVO_S
Vcc
GND
R3
S_PWR
ESC I/O
ESC_S
Vcc
GND
V 1.3.0
NTNU BSc E2118
L. Lerdahl, K. Slagstad, J. Jandgraff, A. Sobstad
04.03.2021

PCB design by L.M. Lerdahl

Design Rules Verification Report

Filename : C:\ProgramData\Altium\CircuitMaker {A4E57FE9-FE35-4287-AC6C-DAED69C9481F}\Projects\2BD7A72C-2D34-44FF-80D6-A63FB7B7E87A\fa5f0fab-c3e3-4cf7-bd92-e1c1b0f60512\PCB_E2118_122.CMPcbDoc

Warnings 0
Rule Violations 0

Warnings	
Total	0

Rule Violations	
Short-Circuit Constraint (Allowed=No) (All),(All)	0
Un-Routed Net Constraint ((All))	0
Clearance Constraint (Gap=0.254mm) (All),(All)	0
Power Plane Connect Rule(Relief Connect)(Expansion=0.508mm) (Conductor Width=0.254mm) (Air Gap=0.254mm)	0
Width Constraint (Min=0.254mm) (Max=1.016mm) (Preferred=0.508mm) (All)	0
Height Constraint (Min=0mm) (Max=25.4mm) (Prefered=12.7mm) (All)	0
Hole Size Constraint (Min=0.9mm) (Max=6mm) (All)	0
Hole To Hole Clearance (Gap=0.254mm) (All),(All)	0
Minimum Solder Mask Sliver (Gap=0.254mm) (All),(All)	0
Silk To Solder Mask (Clearance=0.254mm) (Disabled)(IsPad),(All)	0
Silk to Silk (Clearance=0.254mm) (All),(All)	0
Silk primitive without silk layer	0
Net Antennae (Tolerance=0mm) (All)	0
Unpoured Polygon (Allow unpoured: False)	0
Total	0

Vedlegg J

Informasjonsplakat

Bacheloroppgave

FJERNSTYRING AV KJØRETØY OVER 5G/4G

Jon Ola Landgraff, Lars Markus Lerdahl, Karsten Sedal Slagstad og Aksel Digre Søbstad

NTNU

BESKRIVELSE

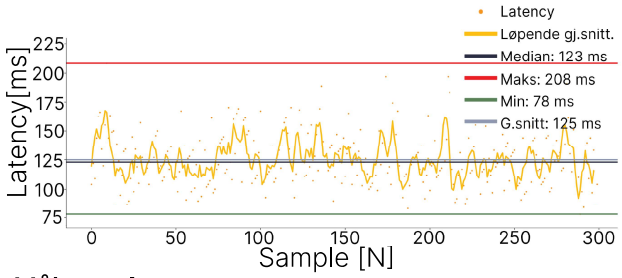
- Lage et system for fjernstyring over 5G/4G.
  - Direkte-video fra bil med vidvinkel-linse.
  - Racingstol med ratt og pedaler for styring.

- Finne ut om systemet er responsivt nok til å kunne kjøre sømløst

- Løsning basert på en kombinasjon av hyllevare og egenutviklet maskinvare og programvare.

RESULTATER

Videoforsinkelse via 5G. Glass to Glass



- **Måleresultater:**
  - Medianverdi på videoforsinkelse: 123 ms.
  - Medianverdi på styringssignal-forsinkelse: 35 ms.

- **Subjektive resultater:**
  - Piloten opplever å ha god kontroll over kjøretøyet.

KONKLUSJON

- Prosjektet har vist at fjernstyring av kjøretøy over 5G/4G er mulig.
- Fjernstyring over 5G gir både mer stabil og mer responsiv video-overføring, enn fjernstyring over 4G.
- Sluttproduktet viser stort forbedringspotensiale knyttet til video-prosessering.

Landgraff | Lerdahl | Slagstad | Søbstad

Fjernstyring av kjøretøy over 5G/4G

NTNU
Kunnskap for en bedre verden