

Vanessa Berg
Tobias Dybvad
Henrik Christoffer Kran
Jardar Slaatten Lysebo
Kamilla Snekestad-Bjønnes
Ida Brattøy Ustad

Verdens mest avanserte modelljernbane

Et sommerprosjekt om utbygging av modelljernbane med integrasjon av avansert sensorikk og teknologi



Prosjekteier: Elisabeth Cabrinetti
Prosjektleder: Vanessa Berg & Ida Brattøy Ustad
August 2026

Norges teknisk-naturvitenskaplige universitet - NTNU
Institutt for bygg- og miljøteknikk

Sammendrag

Denne rapporten dokumenterer sommerprosjektet 2026 på Konnekts modelljernbane, en fysisk modell av et jernbaneanlegg som brukes som læringsplattform innen jernbanefag og til å vise fram faget på messer og utstillinger. Prosjektet er en videreføring av sommerprosjektet 2025, der selve modellen ble bygget, og har i hovedsak dreid seg om teknologien og programvaren rundt anlegget.

Det ble utviklet en løsning for maskinlæring og kameradeteksjon som gjenkjenner togene visuelt. Arbeidet startet med innsamling og annotering av et datasett på 1216 bilder i Roboflow, og gikk gjennom flere faser der både kamera, modellarkitektur og maskinvare ble byttet. Den endelige løsningen bruker en YOLOv11-modell som kjører direkte på sensoren i et Raspberry Pi AI Camera, slik at deteksjonen skjer i sanntid uten å belaste datamaskinen som samtidig styrer togene.

Rapporten dokumenterer også design og oppsett av banen, inkludert et understell med vippefunksjon som gjør at modulen kan flyttes gjennom vanlige dører, sporplanlegging i SCARM, landskapsarbeid, og inndelingen av anlegget i seksjoner overvåket av fire Z21 Detector X8-bokser.

Programvaren for togstyring ble utviklet fra en enkel kjørekommando til et komplett system. Kommunikasjonen med Roco Z21-sentralen skjer over nettverk, og posisjonen til hvert tog leses fra belegningsdetektorer og RailCOM. Brukergrensesnittet er utviklet i fire generasjoner, fra et enkelt program for en testbane til dagens løsning som kjører i nettleseren og kan betjenes fra flere enheter samtidig. Systemet har en kollisjonskontroll som nødstopper tog før de kjører inn i opptatte spor, og en funksjon for autonom kjøring der togene finner vei selv, stiller sporvekslene i forkant og reserverer sporet foran seg slik at flere tog kan kjøre samtidig uten å kollidere.

Videre ble det utviklet en informasjonsnettside som gjør prosjektet tilgjengelig for besøkende utover selve utstillingen. Nettsiden er bygget med Next.js og Tailwind CSS etter et mobile-first-prinsipp, og nås ved å skanne en QR-kode ved modellen. Det ble lagt vekt på universell utforming og på en struktur som er enkel å videreutvikle.

Det ble også arbeidet med en Train Gate, en sensorportal for tilstandskontroll av togmateriell. Portalen er ferdig som fysisk modell, og de aktuelle sensorene og løsningene er kartlagt, men det er ikke gjennomført elektronisk arbeid innenfor prosjektets ramme.

Rapporten inneholder til slutt en praktisk veiledning for demontering, frakt og montering av anlegget, slik at det kan tas med på utstillinger av personer som ikke har vært med på å bygge det.

Et gjennomgående mål har vært at arbeidet skal kunne videreføres av framtidige studentgrupper. Valgene som er tatt er derfor begrunnet, og hvert kapittel avsluttes med konkrete anbefalinger for videre arbeid. De viktigste er å la kollisjoner løses automatisk framfor manuelt, å innføre prioritering mellom tog som skal bruke samme sporveksel, å utvide kameradeteksjonen med flere kameraer og kraftigere maskinvare, og å ta i bruk elektronikken i Train Gate.

Figurer

1	Gammel og ny modul ovenfra.	3
2	SCARM bilde	4
3	Detaljer og landskap	5
4	Seksjoner	6
5	Annotering av tog i Roboflow.	7
6	Resultater fra modellene trent i Roboflow.	8
7	AI-kameraet som detekterer rødt og grått tog	10
8	Første versjon av grensesnittet, laget til testbanen.	18
9	Begynnelsen på å lage hele grensesnittet	19
10	Andre versjon av grensesnittet, utvidet til hele anlegget.	19
11	Tredje versjon, med fargekoding per spor og oversikt over detektormodulene.	20
12	Fjerde og nåværende versjon: togstyringen som en side i nettleseren.	20
13	Raspberry PI kjører programmet på skjermen	24
14	Nettsidens design i Figma.	27
15	Train Gate-portalen.	31
16	Train Gate-portalen.	32
17	Utstyr til stand	34
18	Jord, spenning og CAN må kobles mellom modulene.	35
19	Gjengestaver og muttere holder modulene sammen.	36
20	Bein festest/løsnes med bolter og muttere øverst og nederst.	36
21	Montering/Demontering av hjul	37

Tabeller

1	Inndeling av det annoterte datasettet.	8
2	Parametre benyttet under trening av YOLO-modellen.	9
3	Innstillinger for deteksjon på Raspberry Pi AI Camera.	11
4	Utviklingen av maskinlæringsløsningen.	12
5	Oversikt over teknologiene som ble benyttet i utviklingen av nettsiden.	29

Innhold

Figurer	ii
Tabeller	ii
1 Innledning	1
1.1 Mål	1
1.2 Arbeidsmåte	1
1.3 Rapportens oppbygning	1
2 Design og oppsett av banen	3
2.1 Design og bygging av understell	3
2.2 Design av togspor i ny modul	3
2.3 Landskap	3
2.4 Seksjoner	4
2.5 Reverse loop/vendelsløyfe	5
3 Maskinlæring og kameradeteksjon	6
3.1 Formål	6
3.2 Datagrunnlag og annotering i Roboflow	6
3.2.1 Innsamling av bilder	7
3.2.2 Annotering og klasser	7
3.2.3 Datasettets inndeling	7
3.2.4 Lagring av datasettet	8
3.3 De første modellene og OpenMV	8
3.4 Egentrent modell: Faster R-CNN	9
3.5 YOLOv11 med Ultralytics	9
3.5.1 Treningsoppsett	9
3.5.2 Konvertering fra polygoner til bokser	9
3.5.3 Live deteksjon fra webkamera	10
3.6 Flytting av inferens til AI-kamera	10
3.6.1 Konvertering av modellen	10
3.6.2 Tilpasning av koordinater	11
3.6.3 Innstillinger	11
3.6.4 Integrasjon i anlegget	11
3.7 Oppsummering og videre arbeid	11

3.7.1	Videre arbeid	12
4	Kameradeteksjon med OpenMV	13
4.1	Maskinvare: OpenMV N6	13
4.2	Programvaremiljø og oppsett	13
4.3	Arbeidsflyt for deteksjon	14
4.4	Utfordringer og videre arbeid	15
5	Programvareutvikling og automatisert togkjøring	16
5.1	Z21-sentralen og hvordan den styres	16
5.1.1	Kommunikasjon over nettverk	16
5.1.2	Å kjøre et lokomotiv	16
5.1.3	Sporveksler	16
5.1.4	Å lese hvor togene er	17
5.2	Fra første kommando til fungerende styring	17
5.3	Utvikling av brukergrensesnittet	17
5.3.1	Versjon 1: testbanen	18
5.3.2	Versjon 2 og 3: hele anlegget	18
5.3.3	Versjon 4: styring i nettleseren	19
5.3.4	Fartsbegrensning	21
5.4	Kollisjonskontroll	21
5.4.1	To former for fare	21
5.4.2	Hvilken vei er framover?	21
5.4.3	Å ikke stole blindt på detektorene	21
5.4.4	Betjening	22
5.5	Autonom togkjøring	22
5.5.1	Grunnprinsippet	22
5.5.2	Reservasjoner	22
5.5.3	To stegs framsyn	22
5.5.4	Serialisering av stegene	23
5.5.5	Vaktbikkje	23
5.5.6	Samspill med manuell styring	23
5.6	Veien videre	23
5.6.1	Automatisk løsning av låsesituasjoner	23
5.6.2	Prioritering ved felles sporveksel	24
5.6.3	Mer variert kjøremønster	24

5.7	Raspberry Pi	24
5.7.1	Videreutvikling av Raspberry Pi-oppsettet	25
6	Nettside	26
6.1	Formål	26
6.2	Design og planlegging	26
6.2.1	Behovsanalyse	26
6.2.2	Designvalg	26
6.2.3	Informasjonsarkitektur	27
6.3	Innhold	28
6.4	Implementasjon	28
6.4.1	Teknologivalg	28
6.4.2	Struktur	29
6.5	Hosting	30
6.6	Hvordan videreføre prosjektet	30
6.6.1	Videre drift	30
6.6.2	Oppdatering av innhold	30
7	Train Gate	31
7.1	Kartlagte sensorer og løsninger	31
7.1.1	Termisk kamera	31
7.1.2	Vektmåling og hjuldiagnostikk	31
7.1.3	Kameraovervåking	32
7.1.4	Identifikasjon av tog	32
7.1.5	Hastighet	32
7.1.6	Værstasjon	32
7.1.7	Akustisk måling	33
7.2	Veien videre	33
8	Steg-for-steg: demontering, frakt og montering	34

1 Innledning

Denne rapporten dokumenterer arbeidet som ble utført på Konnekts modelljernbaneprojekt gjennom sommerprosjektet i 2026. Modelljernbanen er en fysisk modell av et jernbaneanlegg som fungerer som en læringsplattform innen jernbanefag, og som samtidig brukes til å vise fram faget for besøkende på messer og utstillinger.

Prosjektet er en videreføring av arbeidet fra sommerprosjektet i 2025, der selve modellen ble bygget. Der det foregående prosjektet i hovedsak handlet om det fysiske anlegget, har årets arbeid i større grad dreid seg om teknologien og programvaren rundt det, samt om å gjøre prosjektet tilgjengelig for andre.

1.1 Mål

Prosjektet hadde flere delmål, som til sammen utgjør kapitlene i denne rapporten.

Et hovedmål var å ta i bruk maskinlæring og kameradeteksjon på anlegget, slik at systemet kan gjenkjenne togene visuelt og ikke bare gjennom den elektriske tilbakemeldingen fra sporet. Et annet var å utvikle programvaren for togstyring videre, fra enkel manuell kjøring til et system med kollisjonskontroll og tog som kjører helt av seg selv.

Videre skulle det utvikles en informasjonsnettside som gjør prosjektet tilgjengelig for besøkende også utenfor selve utstillingen, og det skulle arbeides med en Train Gate, en sensorportal som demonstrerer tilstandskontroll av togmateriell i modellskala.

Et gjennomgående mål på tvers av alle delene har vært at arbeidet skal kunne videreføres av framtidige studentgrupper. Prosjektet er ikke ferdig, og er heller ikke ment å bli det. Løsningene er derfor dokumentert, og de valgene som er tatt er begrunnet, slik at de som overtar forstår hvorfor systemet ser ut som det gjør. Rapporten inneholder av samme grunn konkrete anbefalinger for videre arbeid i hvert enkelt kapittel.

1.2 Arbeidsmåte

En erfaring som har preget arbeidet er verdien av å starte i det små og bygge oppover, framfor å sette sammen hele systemet på én gang. Et anlegg som dette består av mange ledd som alle må fungere: skinner og strømforsyning, en digitalsentral, detektorer i sporet, programvaren som tolker signalene, og til slutt brukergrensesnittet. Når alt kobles sammen samtidig og noe ikke virker, er det nesten umulig å si hvor feilen ligger.

Ved i stedet å verifisere ett ledd av gangen, og beholde de enkle testprogrammene underveis, blir feilsøking langt raskere. Flere av de mest tidkrevende feilene i prosjektet ble funnet nettopp fordi det fantes et enkelt program som kunne isolere om problemet lå i maskinvaren eller i koden. Denne framgangsmåten går igjen i flere av kapitlene.

Datablader og protokollspesifikasjoner har vært en viktig kilde gjennom hele arbeidet. De fleste komponentene i anlegget har dokumentasjon tilgjengelig, og for de digitale delene av systemet er den ofte nødvendig for å komme videre i det hele tatt. Der dokumentasjonen ikke har vært tilstrekkelig, er detaljene kartlagt ved å måle og observere hva utstyret faktisk gjør.

1.3 Rapportens oppbygning

Rapporten er bygget opp slik at hvert kapittel dekker en avgrenset del av prosjektet.

Kapittel 2 beskriver design og oppsett av selve banen: understell, sporplan, landskap og inndelingen i seksjoner. Kapittel 3 beskriver arbeidet med maskinlæring og kameradeteksjon, fra innsamling og annotering av bilder til den løsningen som i dag kjører på anlegget. Kapittel 4 tar

for seg et parallelt spor med OpenMV-kameraet. Kapittel 5 tar for seg programvaren for togstyring, hvordan systemet kommuniserer med digitalsentralen, hvordan brukergrensesnittet er utviklet, og hvordan kollisjonskontroll og autonom togkjøring fungerer.

Kapittel 6 dokumenterer utviklingen av informasjonsnettsiden, inkludert design, teknologivalg, hosting og hvordan den kan videreføres. Kapittel 7 beskriver Train Gate, med de sensorene og løsningene som er kartlagt for videre arbeid.

Til slutt inneholder kapittel 8 en steg-for-steg-veiledning for demontering, frakt og montering av anlegget. Denne delen er skrevet som en praktisk bruksanvisning framfor som dokumentasjon av utviklingsarbeid, ettersom anlegget skal fraktes til og fra utstillinger av personer som ikke nødvendigvis har vært med på å bygge det.

2 Design og oppsett av banen



Figur 1: Gammel og ny modul ovenfra.

2.1 Design og bygging av understell

Skal en modul kunne flyttes gjennom vanlige dører, må understellet være under 90 cm bredt. Det var derfor hovedmodulen ble redesignet og fikk en vippefunksjon. Den nye modulen designet vi selv fra bunnen av i sommer, og kunne dermed tilpasse målene underveis, så den er smal nok i seg selv og trenger ikke samme løsning. Nye moduler bør uansett ikke være lengre enn 2 m. Byggingen av understellet er gjort med enkle metoder. Ønsker man å forstå hvordan det er satt sammen holder det å se under bordet.

Hjul er en viktig del av oppsettet. Understellene er bygget slik at bordene har samme høyde både med og uten hjul montert, men med hjulene på blir det ganske høyt, så da er det greit å kunne ta dem av til utstilling. Hjulene er avtakbare, og krever ikke verktøy: løft bordet, så faller hjulene ut av seg selv.

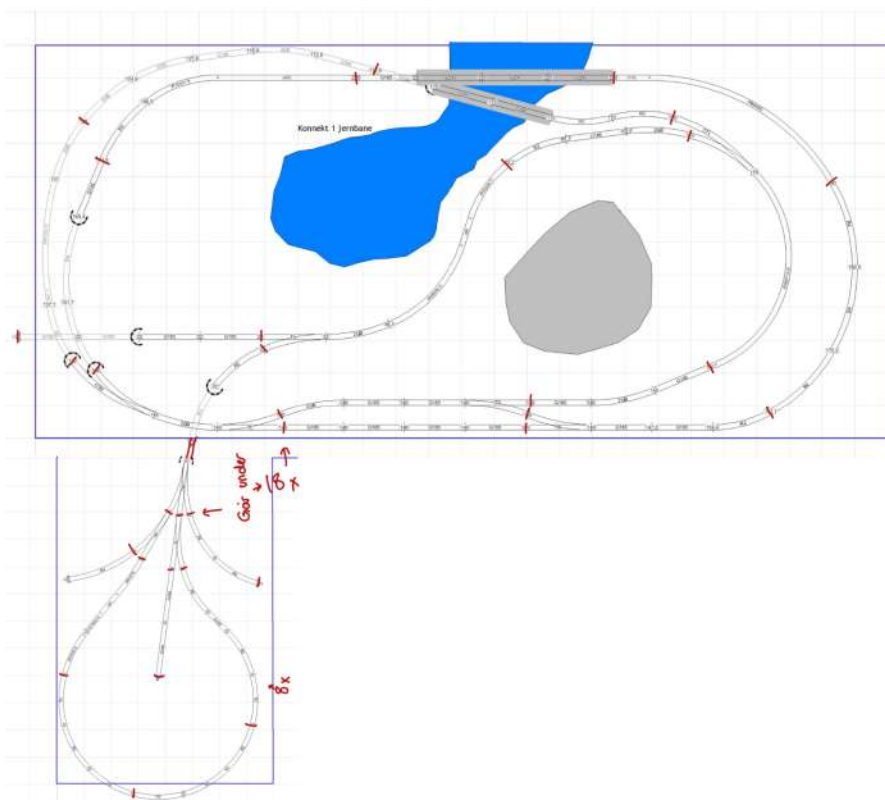
2.2 Design av togspor i ny modul

Sporplanen til den nye modulen er tegnet i SCARM. Programmet er fint å bruke fordi man kan prøve seg frem med forskjellige skinner og layout før man bestemmer seg, og det lager automatisk en handleliste over hvilke skinner man må kjøpe.

Viktig ved oppsett av nytt prosjekt i SCARM: velg H0-skala og Rocco GeoLine som skinnetype. Ellers blir hverken målestokk eller skinnevalg riktig.

2.3 Landskap

Til selve landskapet brukes hobbylim, trelim, eller en limblanding, som ofte er trelim fortynnet med vann (gjør gjerne med en dråpe oppvaskmiddel i for å bryte overflatespenningen). Denne blandingen er spesielt praktisk til å feste ballast/pukk og annet løst materiale, siden den suger seg lettere ned mellom kornene enn ufortynnet lim. Siden den store modulen har en vippefunksjon, så det kan ikke ligge løse gjenstander eller elementer på den.



Figur 2: SCARM bilde

Fjellpartiene er bygget opp med avispapir og pappmasjé som grunnform, før det legges sparkelmasse oppå for å gi selve overflaten og strukturen til fjellet. Når sparkelen har tørket, kan man spraye på litt vann for å myke den opp igjen, og deretter skrape og forme den for et mer realistisk preg. Fjellene er grunnmalt grå og deretter bygget opp med flere nyanser av grå og litt hvitt tilfeldig fordelt, for å gi et naturlig utseende fremfor en jevn gråfarge.

Steinbruddet er bygget i en trekasse med isopor som base under. Noen mindre detaljer er 3D-printet og malt i etterkant. Det ligger mye gode tips og fremgangsmåter for slikt landskapsarbeid å finne på nett.

Det viktigste å huske ved videre landskapsarbeid: skinnene må aldri bli sperret eller dekket av landskapet, og de må fortsatt kunne demonteres eller byttes ut ved behov. De er skjøre, så vær forsiktig rundt dem.

2.4 Seksjoner

Banen deles opp i seksjoner ved å sette inn en isolerende lask på spenningsskinne på hver side av seksjonen. Deretter loddes det på en kabel fra spenningsskinne, som trekkes inn i en av inngangene på detektorboksen. Mer detaljert beskrivelse av denne fremgangsmåten finnes i databladet til Z21 Detector.

Anlegget er delt opp i seksjoner som overvåkes av fire Z21 Detector X8-bokser: tre på den store modulen og én på den nye. X8 er brukt fremfor X16 fordi X8 har CAN-buss, som de andre boksene kommuniserer gjennom. CAN-bussen er nødvendig for at RailCom-informasjonen fra togene skal nå frem til Z21-sentralen.

Den store modulen har 18 seksjoner totalt, fordelt på de tre detektorboksene. Bordet er i praksis delt i tre soner: alle seksjoner på høyre side hører til én boks, midtseksjonene til en annen, og venstre side til den tredje som vist i Figur 4. Alle fire boksene er koblet sammen via CAN, og den



(a) Landskap ny modul



(b) Stasjonsbygg



(c) Steinbrudd



(d) 3D-printet vanntårn

Figur 3: Detaljer og landskap

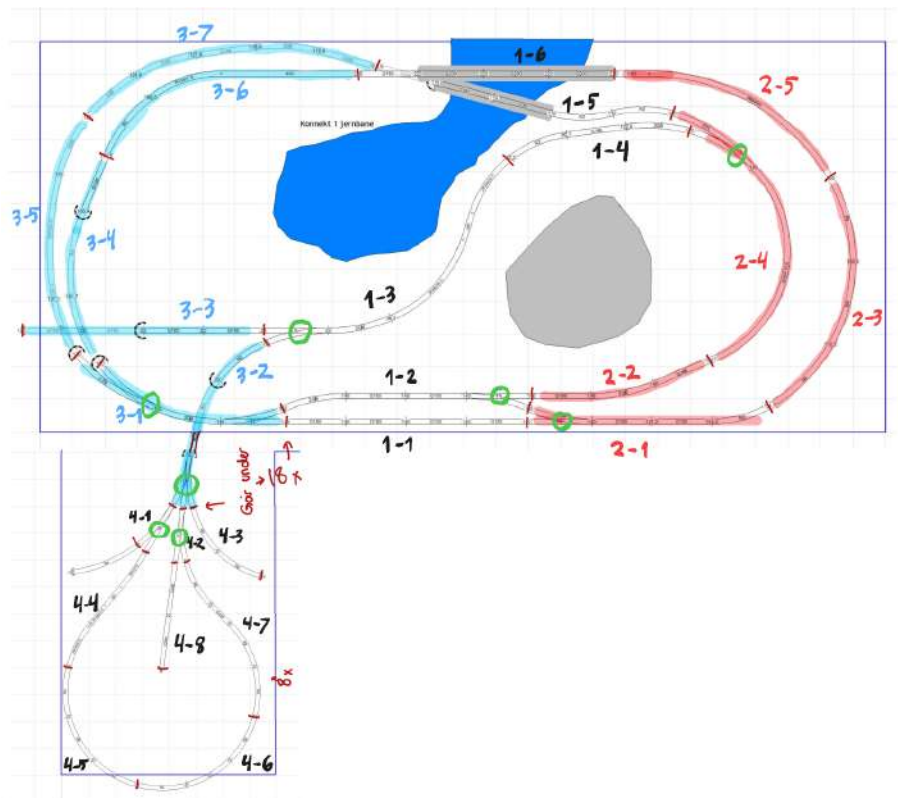
første boksen i rekken er koblet videre til Z21-sentralen, også via CAN.

2.5 Reverse loop/vendelsløyfe

Vendesløyfe skaper et elektrisk problem: spenningen følger høyreskinnen gjennom hele sporet, men når toget kommer ut igjen på den andre siden av loopen, møter skinnen jordskinnen med feil polaritet. Uten en løsning på dette blir det kortslutning.

Løsningen er en Z21 Multi Loop-boks, kombinert med to korte isolerte skinnebiter i hver ende av loopen. Multi Loop-boksen kjenner igjen når toget kjører inn i loop-seksjonen og bytter polaritet automatisk, slik at spenning og jord alltid stemmer overens når toget kommer ut igjen.

Seksjonen som styres av Multi Loop-boksen kan aldri være kortere enn det korteste strømførende kjøretøyet (lok eller vogn) som skal kjøre gjennom den. Er den for kort, mister toget kontakt med riktig polaritet midt i bryteøyeblikket.



Figur 4: Seksjoner

3 Maskinlæring og kameradeteksjon

Som en del av sommerprosjektet i 2026 ble det arbeidet med å ta i bruk maskinlæring i modelljernbanen. Målet var at systemet skulle kunne gjenkjenne togene på anlegget ved hjelp av kamera, som et supplement til den elektriske tilbakemeldingen fra R-Bus-detektorene.

Arbeidet ble gjennomført trinnvis, der flere løsninger ble prøvd før den endelige kom på plass. Underveis ble det byttet både kamera, modellarkitektur og maskinvare, og hvert bytte hadde en konkret årsak. Kapittelet følger denne utviklingen kronologisk, fra de første bildene til løsningen som i dag kjører på anlegget.

3.1 Formål

Hovedformålet med kameradeteksjonen er å gi systemet informasjon om togene som ikke kan hentes ut fra sporet alene. R-Bus-detektorene gir en pålitelig tilbakemelding på om en seksjon er belagt, men ikke helt presist hvor i seksjonen toget befinner seg.

I tillegg hadde arbeidet et pedagogisk formål. Modelljernbanen fungerer som en læringsplattform innen jernbanefag, og maskinlæring er en teknologi som i økende grad tas i bruk i jernbanesektoren, blant annet til overvåking og tilstandskontroll. Ved å implementere gjenkjenning av tog i modellskala kan prinsippene bak slik teknologi demonstreres i praksis for besøkende og studenter.

3.2 Datagrunnlag og annotering i Roboflow

Det ble først bestemt å lage et proof of concept. Til dette ble Roboflow valgt, en nettbasert plattform for annotering, versjonering og eksport av bildedatasett. Roboflow ble valgt fordi

plattformen dekker hele veien fra opplasting av bilder til ferdig datasett, og fordi den støtter eksport til mange ulike formater. Det siste viste seg å bli viktig, ettersom det underveis ble byttet modellarkitektur flere ganger, og hver arkitektur krever sitt eget format.

3.2.1 Innsamling av bilder

De første bildene ble tatt med et OpenMV-kamera, omtrent 200 stykker av togene på anlegget. Bildene ble tatt fra ulike vinkler og med togene plassert forskjellige steder, slik at modellen skulle lære å kjenne igjen selve toget framfor én bestemt situasjon.

Kvaliteten fra OpenMV-kameraet viste seg raskt å bli for lav. Det ble derfor byttet til et webkamera med 1080p, og bildene ble tatt automatisk med et enkelt Python-skript. På denne måten ble det samlet inn omtrent 1200 bilder til den ordentlige modellen.

3.2.2 Annotering og klasser

Bildene ble annotert i Roboflow, der hvert tog ble markert og tilordnet en klasse. Det ble definert to klasser, **grey** og **red**, etter togenes farge. Antallet klasser ble bevisst holdt lavt, både fordi anlegget kjøres med to lokomotiver, og fordi færre klasser krever mindre treningsdata.

Til selve markeringen ble Roboflows eget AI-verktøy benyttet, som foreslår omrisset av objektet automatisk. Dette fungerte veldig bra og sparte mye tid, men forbruket av tokens gikk fortere enn ventet, slik at det måtte kjøpes flere underveis.

Annoteringen ble utført som polygoner, altså ved å tegne omrisset av toget framfor et enkelt rektangel. Dette gir en mer presis markering, men som beskrevet i avsnitt 3.5 måtte annotasjonene senere gjøres om til rektangulære bokser.



Figur 5: Annotering av tog i Roboflow.

3.2.3 Datasettets inndeling

Det ferdige datasettet endte på 1216 bilder, fordelt på tre deler som vist i tabell 1.

Tabell 1: Inndeling av det annoterte datasettet.

Del	Formål	Antall bilder
train	Trening av modellen	852
valid	Validering underveis i treningen	243
test	Slutttest på ukjente data	121
Totalt		1216

Denne tredelingen er vanlig praksis. Treningssettet brukes til å justere modellen, valideringssettet til å kontrollere underveis at den faktisk generaliserer framfor å memorere treningsbildene, mens testsettet holdes helt utenfor treningen og gir et uavhengig mål på hvordan modellen presterer på bilder den aldri har sett.

3.2.4 Lagring av datasettet

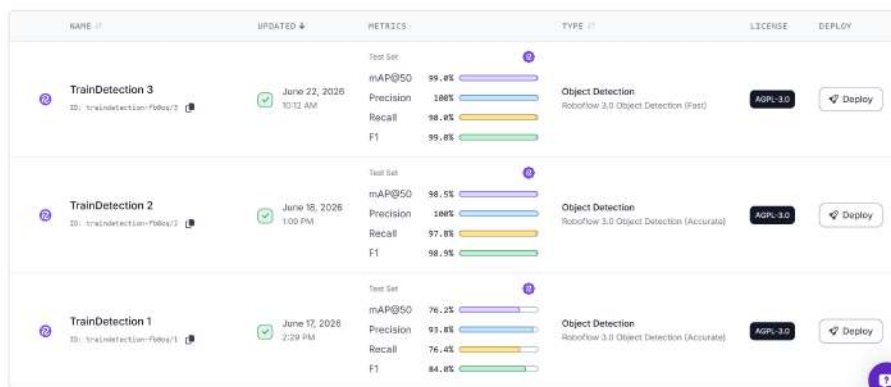
Det annoterte datasettet lagres ikke i prosjektets Git-repositorium, ettersom bildefilene er for store for et kodearkiv. Roboflow fungerer i stedet som kilden datasettet hentes fra ved behov, og framgangsmåten er dokumentert i `computer_vision/README.md`. En fordel med dette er at datasettet er versjonert, slik at det er mulig å gå tilbake til en tidligere versjon dersom en endring skulle gi dårligere resultater.

3.3 De første modellene og OpenMV

Med datasettet på plass ble de første modellene trent direkte i Roboflow. Den første ble trent på de omtrent 200 bildene fra OpenMV-kameraet, og fungerte som en bekreftelse på at framgangsmåten var riktig. Den andre ble trent på det utvidede datasettet fra webkameraet, og ga tydelig bedre resultater.

Planen var å legge modellen inn på selve OpenMV-kameraet, slik at deteksjonen kunne kjøre uten en datamaskin tilkoblet. Modellen viste seg imidlertid å være for stor for kameraets lagringsplass. Det ble derfor trent en tredje og mindre modell tilpasset denne begrensningen.

Her oppstod likevel et hinder som avgjorde veien videre. Roboflow tillot kun eksport til OpenMV-format uten ekstra kostnad, mens de øvrige formatene krevde betalt abonnement. Siden prosjektet ikke var tjent med å låses til ett bestemt kamera, ble det besluttet å beholde det annoterte datasettet og i stedet trene modellene selv. Datasettet var på dette tidspunktet det mest verdifulle resultatet av arbeidet, og det kunne gjenbrukes uavhengig av hvilken modell som ble valgt videre.



Figur 6: Resultater fra modellene trent i Roboflow.

3.4 Egentrent modell: Faster R-CNN

Den første modellen som ble trent lokalt var en Faster R-CNN, implementert med PyTorch og torchvision. Dette er en såkalt tostegs-detektor, som først foreslår områder i bildet der det kan finnes seg et objekt, og deretter klassifiserer hvert område. Arkitekturen er kjent for god nøyaktighet, og var derfor et naturlig utgangspunkt.

Framfor å trene fra bunnen av ble det benyttet *transfer learning*. Det innebærer at det lastes inn en modell som allerede er trent på et stort, generelt datasett, hvorpå kun det siste laget byttes ut slik at det passer prosjektets klasser. Modellen har da allerede lært å gjenkjenne former, kanter og strukturer, og trenger bare finjustering. Uten dette ville et datasett på rundt tusen bilder vært altfor lite.

Løsningen fungerte teknisk, men var uegnet til formålet. Faster R-CNN er en tung arkitektur, og utviklingsmaskinen er en Windows ARM64-maskin uten NVIDIA-grafikkort, slik at all trening måtte kjøres på prosessoren. Trening over fem epoker tok mange timer, noe som gjorde det upraktisk å prøve ut endringer. Viktigere var det at modellen brukte for lang tid per bilde til å kunne følge togene mens de er i bevegelse. Det ble derfor byttet til en lettere arkitektur.

3.5 YOLOv11 med Ultralytics

For å få opp hastigheten ble det byttet til YOLO i versjon 11, gjennom biblioteket Ultralytics. YOLO er en enkeltstegs-detektor, som finner og klassifiserer objektene i én operasjon framfor å foreslå kandidatområder først. Det gir noe lavere nøyaktighet i teorien, men er langt raskere, og er derfor det som vanligvis brukes til sanntidsdeteksjon. Ultralytics håndterer i tillegg datalasting, treningslønke og lagring, slik at kodemengden ble betydelig mindre enn i den forrige løsningen.

Det ble valgt varianten `yolo11n.pt`, der `n` står for *nano* og angir den minste og raskeste utgaven. Den er lett nok til å kjøre flytende på prosessoren alene, noe som var en forutsetning med den maskinvaren som var tilgjengelig.

3.5.1 Treningsoppsett

Treningen ble satt opp med parametrene i tabell 2.

Tabell 2: Parametre benyttet under trening av YOLO-modellen.

Parameter	Verdi	Begrunnelse
<code>epochs</code>	50	Godt utgangspunkt; YOLO konvergerer raskt
<code>imgsz</code>	640	Standard bildestørrelse for YOLO-trening
<code>device</code>	<code>cpu</code>	Ingen NVIDIA-grafikkort tilgjengelig
<code>patience</code>	15	Avbryter treningen ved manglende forbedring

Parameteren `patience` gjør at treningen stopper automatisk dersom modellen ikke blir bedre over 15 påfølgende epoker. Det sparer tid og motvirker samtidig at modellen overtilpasses treningsdataene. Full trening tok med dette oppsettet omtrent 30 til 60 minutter, mot flere timer for den forrige løsningen. De beste vektene lagres som `best.pt` i katalogen `runs/detect/train/weights/`.

3.5.2 Konvertering fra polygoner til bokser

Som nevnt i avsnitt 3.2 var annotasjonene laget som polygoner, og eksporten fra Roboflow inneholdt i tillegg en blanding av polygoner og bokser. Dette fikk YOLOs segmenteringstrening til å avbryte med feil.

Løsningen ble et lite hjelpeprogram, `seg_to_box.py`, som regner om hvert polygon til den minste boksen som omslutter det. Programmet skriver resultatet til en ny katalog, slik at det opprinnelige datasettet forblir urørt. For vårt formål er en rektangulær boks god nok. Det er hvor toget er som er interessant, ikke det nøyaktige omrisset, og bokser er samtidig raskere å beregne.

3.5.3 Live deteksjon fra webkamera

Resultatet ble programmet `yolo_webcam.py`, som kjører deteksjon i sanntid fra et tilkoblet webkamera. Programmet åpner kameraet i 1280×720 , leser ett bilde av gangen, kjører det gjennom modellen og tegner boksene med klassenavn oppå originalbildet. Konfidensterskelen ble satt til 0,8, slik at bare deteksjoner modellen er minst 80 prosent sikker på vises.

Denne modellen fungerte veldig bra, men forutsatte at anlegget var koblet til en Windows-maskin. Siden målet var å få alt til å kjøre på Raspberry Pi-en, måtte løsningen tilpasses videre.



Figur 7: AI-kameraet som detekterer rødt og grått tog

3.6 Flytting av inferens til AI-kamera

Anlegget styres av en Raspberry Pi 4, som har langt mindre regnekraft enn utviklingsmaskinen. Den håndterer allerede togstyring og webgrensesnitt, og å legge bildeanalysen til de samme oppgavene ville gitt for lav bildefrekvens til sanntidsdeteksjon.

Løsningen ble å flytte selve beregningen ut av datamaskinen og inn i kameraet. Til dette ble det brukt et Raspberry Pi AI Camera med Sony IMX500-sensor. Det spesielle med denne sensoren er at den har en innebygd neural prosesseringsenhet, slik at modellen kjører på sensoren framfor på datamaskinens prosessor. Raspberry Pi-en mottar dermed ferdig beregnede deteksjoner sammen med bildet, og bruker ingen regnekraft på selve analysen.

Denne tilnærmingen er også interessant utover dette prosjektet. Å utføre beregningen der dataene oppstår, framfor å sende alt til en sentral enhet, brukes i industrielle overvåkingssystemer, og er i jernbanesammenheng relevant der båndbredde eller responstid er begrensende.

3.6.1 Konvertering av modellen

Sensoren kan ikke kjøre YOLO-modellen direkte, så modellen måtte konverteres i to trinn:

1. Eksport fra YOLO-format til IMX-format med `YOLO(best.pt).export(format="imx")`
2. Pakking til sensorens kjørbare format med `imx500-package`, som gir filen `network.rpk`

Modellfilen legges sammen med klasselisten `labels.txt` i katalogen `runs/detect/train/rpi/`, hvor programmet `yolo_rpi.py` finner dem. Klasselisten er nødvendig fordi sensoren returnerer klassene som tall, som programmet må slå opp for å finne riktig navn.

3.6.2 Tilpasning av koordinater

Overgangen ga en ny runde med feilsøking rundt hvordan sensoren oppgir posisjonen til de detekterte objektene. Boksene kom ut i sensorens interne koordinatsystem, og måtte både normaliseres mot modellens inngangsstørrelse og få byttet om x- og y-aksene før de lå riktig oppå bildet. Begge tilpasningene styres av egne innstillinger i programmet, slik at de enkelt kan slås av dersom en framtidig utgave av sensorens programvare endrer dette.

Det ble også observert at boksene ikke fulgte toget jevnt, men hoppet mellom posisjoner. Årsaken var at sensoren ikke leverer nye deteksjoner for hvert eneste bilde, slik at boksene forsvant i bildene som manglet data. Dette ble løst ved at programmet gjenbraker forrige bildes deteksjoner når nye ikke er tilgjengelige.

3.6.3 Innstillinger

De endelige innstillingene er gjengitt i tabell 3.

Tabell 3: Innstillinger for deteksjon på Raspberry Pi AI Camera.

Parameter	Verdi	Begrunnelse
Konfidensterskel	0,50	Balanse mellom falske og manglende deteksjoner
IoU-terskel	0,65	Slår sammen overlappende bokser for samme tog
Maks antall deteksjoner	2	Anlegget kjøres med to lokomotiver
Oppløsning	1280 × 960	Økt fra utgangspunktet for bedre deteksjon

Konfidensterskelen ble satt lavere enn på utviklingsmaskinen, fra 0,8 til 0,50. Det skyldes at modellen komprimeres under konverteringen til sensorens format, noe som gir litt lavere sikkerhetsverdier for samme deteksjon. IoU-terskelen angir hvor stor overlapp som skal til før to bokser regnes som samme objekt, og hindrer at det tegnes flere bokser rundt ett og samme tog. I visningen tegnes de to klassene med hver sin farge, slik at det er lett å se hvilket tog modellen har gjenkjent.

3.6.4 Integrasjon i anlegget

Deteksjonsprogrammet er integrert i oppstartsskriptet `start.sh`, som beskrives nærmere i kapittel 5. Skriptet starter både togserveren og deteksjonsprogrammet automatisk når anlegget skrus på, åpner nettleseren mot betjeningsgrensesnittet, og avslutter alle prosessene ryddig ved nedstenging. Anlegget kan dermed startes fra et skrivebordsikon uten kjennskap til programmene under, noe som var nødvendig for at løsningen skal kunne brukes på utstillinger.

3.7 Oppsummering og videre arbeid

Maskinlæringsdelen gikk gjennom flere faser, der hvert bytte hadde en konkret årsak. Tabell 4 oppsummerer utviklingen.

Den viktigste erfaringen er at valg av modell ikke kan vurderes uavhengig av hvilken maskinvare den skal kjøre på. Faster R-CNN er ikke en dårlig modell, men den var feil modell gitt kravet om sanntid uten grafikkort. YOLO på en Windows-maskin var riktig i utviklingsfasen, men feil for et anlegg som skal kunne stå og kjøre alene. Ved å flytte beregningen inn i kamerasensoren ble begge kravene innfridd.

Tabell 4: Utviklingen av maskinlæringsløsningen.

Fase	Løsning	Beregning	utfø- res	Årsak til endring
1	Roboflow-modell, OpenMV	OpenMV-kameraet		Modellen for stor; eksport låst bak betaling
2	Faster R-CNN (Py-Torch)	Proseszor, Windows-maskin		For treg til sanntid
3	YOLOv11-nano (Ultralytics)	Proseszor, Windows-maskin		Krevde tilkoblet Windows-maskin
4	YOLOv11 på IMX500	Nevral enhet i kamerasensoren		Endelig løsning

Datasettet viste seg samtidig å være det mest gjenbrukbare arbeidet. Det annoterte datasettet i Roboflow overlevde samtlige bytter av arkitektur, og kunne eksporteres på nytt i det formatet den nye løsningen krevde.

3.7.1 Videre arbeid

Dagens løsning bruker ett kamera, koblet til Raspberry Pi-en. En naturlig utvidelse er å koble til flere kameraer, slik at større deler av anlegget dekkes. Dette lar seg gjøre med YOLO på en Windows-maskin, men forutsetter at en slik maskin er tilgjengelig og koblet til under drift. Alternativt kan det brukes flere AI-kameraer, slik at løsningen fortsatt kjører på Raspberry Pi-en alene.

Det anbefales også å ta nye bilder med det kameraet som faktisk står på anlegget, og trene opp modellen på nytt. Landskapet på modellen er endret siden bildene ble tatt, og en modell trent på oppdaterte bilder fra riktig kamera og riktig plassering vil gjenkjenne togene mer pålitelig.

Til slutt kjører deteksjonen i dag som et selvstendig program som viser resultatet på skjerm. Å koble deteksjonene inn i styringslogikken, slik at posisjonen fra kameraet kan brukes sammen med tilbakemeldingen fra R-Bus-detektorene, er en nærliggende videreføring.

4 Kameradeteksjon med OpenMV

Strømbasert blokkdeteksjon forteller oss om en blokk er opptatt eller ledig, men ikke noe mer. Den sier ingenting om *hva* som står i blokken, eller hvilken vei toget kjører. For å få mer ut av deteksjonen ville vi prøve å kjenne igjen toget direkte i kamerabildet. Til dette brukte vi et OpenMV N6, et kamera med innebygd maskinlæring.

Tanken er at kameradeteksjon kan være et supplement til den strømbaserte blokkdeteksjonen. Mens strømmålingen sier at *noe* er der, kan kameraet i tillegg fortelle *hva* som er der. Det åpner for funksjoner som å finne kjøreretning, skille mellom ulike togtyper og på sikt unngå kollisjoner.

4.1 Maskinvare: OpenMV N6

OpenMV N6 er en liten maskinsynsmodul bygget rundt mikrokontrolleren STMicroelectronics STM32N6. Brikken har en Arm Cortex-M55-prosessor på 800 MHz og en egen neural akselerator (ST Neural-ART) på 600 GOPS (INT8). Akseleratoren gjør at modulen kan kjøre objektdeteksjon, for eksempel YOLOv8, i sanntid direkte på enheten. Den trenger altså verken en PC eller en server koblet til.

Kameraet er en 1 MP global shutter-modell. Det er en fordel når vi skal oppdage tog i bevegelse, fordi hele bildet tas opp samtidig. Dermed unngår vi bevegelsesuskarphet (*rolling shutter*-forvrengning). Siden hele utregningen skjer lokalt og strømforbruket er svært lavt (under 0,75 W), kan en slik enhet i prinsippet bygges fast inn i anlegget på samme måte som de andre ESP32-sensorene.

Kameraet er modulært, så både linse og sensor kan byttes ut ved behov. Vi har blant annet en IR-linse tilgjengelig. Den kan brukes for å gjøre deteksjonen mindre følsom for vekslende lysforhold på anlegget. Ved infrarødt lys jobber man i gråtoner i stedet for farger. Fargene gir lite mening under IR-lys uansett, og gråtonebilder krever mindre data og mindre prosessering.

4.2 Programvaremiljø og oppsett

Kameraet programmeres i MicroPython gjennom utviklingsmiljøet OpenMV IDE. Der kan vi kjøre skript rett på enheten og samtidig se kamerabildet i sanntid. Som et første steg kjørte vi et enkelt skript, bare for å sjekke at kameraet i det hele tatt fungerte og for å måle bildefrekvensen (FPS). Skriptet i Listing 1 gjør ikke noe mer enn å starte kameraet og skrive ut FPS. Det er verdt å merke seg at bildefrekvensen er lavere når kameraet er koblet til IDE-en, fordi bilde-data da sendes til PC-en for visning. Den faktiske ytelsen er høyere når enheten kjører for seg selv.

```
1 import sensor
2 import time
3
4 sensor.reset() # Nullstill og start kameraet
5 sensor.set_pixformat(sensor.RGB565) # Pikselformat: farge (RGB565)
6 sensor.set_framesize(sensor.VGA) # Bildestørrelse VGA (640x480)
7 sensor.skip_frames(time=2000) # Vent til innstillingene virker
8
9 clock = time.clock() # Klokke for FPS-måling
10
11 while True:
12     clock.tick() # Oppdater FPS-klokken
13     img = sensor.snapshot() # Ta et bilde
14     print(clock.fps()) # FPS oker naar koblet fra IDE-en
```

Listing 1: Enkelt testprogram for å verifisere kameraet og måle bildefrekvensen.

I de senere skriptene gikk vi over til den nyere `csi`-modulen for å styre kameraet. Den erstatter den eldre `sensor`-modulen. Begge metodene er vist i kodeeksemplene under. Forskjellen ligger først og fremst i hvordan kameraobjektet startes opp.

4.3 Arbeidsflyt for deteksjon

For å kjenne igjen togene måtte vi lage vår egen objekt-detektor. Arbeidsflyten besto av fire steg: først samle inn bilder, så annotere dem i RoboFlow, deretter trene en YOLOv8-modell, og til slutt eksportere modellen til .tflite og kjøre den lokalt på N6.

Innsamling av treningsdata

Først samlet vi inn bilder av togene på anlegget. Listing 2 viser skriptet vi brukte for å ta bilder og lagre dem fortløpende på microSD-kortet. Hvert bilde får et eget løpenummer, slik at vi ikke overskriver bilder vi allerede har tatt. Bildene lagres i full kvalitet, så detaljene er bevart når de senere skal annoteres.

```
1 import sensor
2 import os
3 import time
4
5 sensor.reset()
6 sensor.set_pixformat(sensor.RGB565)
7 sensor.set_framesize(sensor.HD)
8 sensor.skip_frames(time=2000)
9
10 folder_path = "/sdcard/bilder"
11 try:
12     os.mkdir(folder_path)                # Lag mappen hvis den ikke finnes
13 except OSError:
14     pass
15
16 files = os.listdir(folder_path)
17 file_count = len(files)
18 print("Antall filer:", file_count)
19
20 img = sensor.snapshot()
21 filename = folder_path + "/img_%03d.png" % file_count
22 img.save(filename, quality=100)         # Lagre i full kvalitet
23 print("Lagret:", filename)
24
25 sensor.flush()
26 time.sleep(1)
```

Listing 2: Innsamling og lagring av bilder på microSD-kort, til bruk som treningsdata.

Her er også skriptet for å kjøre kameraet med IR-belysning.

```
1 import csi
2 import time
3
4 csi0 = csi.CSI()
5 csi0.reset()
6 csi0.pixformat(csi.GRAYSCALE)          # Graatoner passer godt for IR
7 csi0.framesize(csi.HD)
8
9 clock = time.clock()
10 while True:
11     clock.tick()
12     img = csi0.snapshot()
13     print(clock.fps())
```

Listing 3: Oppsett av kamera for gråtoneopptak, egnet for IR-belysning.

Annotering, trening og eksport

Bildene ble annotert og brukt til å trene en YOLOv8-modell i RoboFlow. OpenMV samarbeider med RoboFlow og har en ferdig arbeidsflyt for å trene modeller og eksportere dem til N6. Den

ferdige modellen ble eksportert til TensorFlow Lite-format (.tflite) og lagt på enhetens skrivebeskyttede /rom-filsystem. Derfra lastes modellen rett fra flash, uten å kopieres til RAM. Det gjør at selv ganske store modeller kan kjøre samtidig med det levende kamerabildet.

Inferens på enheten

Inferensskriptet er vist i Listing 4. Kamerabildet beskjæres med `window((192, 192))` slik at det passer med størrelsen modellen forventer. Modellen lastes med YOLOv8-etterprosessering og en deteksjonsterskel på 0,4. Resultatene tegnes som rammer oppå bildet i sanntid.

```
1 import csi, time, ml
2 from ml.postprocessing.ultralytics import YoloV8
3
4 # Start kameraet
5 csi0 = csi.CSI()
6 csi0.reset()
7 csi0.pixformat(csi.RGB565)
8 csi0.framesize(csi.VGA)
9 csi0.window((192, 192)) # Match modellens input-størrelse
10
11 # Last inn modellen fra ROM
12 model = ml.Model("/rom/openmv_traindetection-fb0oq_v2.tflite",
13                 postprocess=YoloV8(threshold=0.4))
14 print(model)
15
16 clock = time.clock()
17 while True:
18     clock.tick()
19     img = csi0.snapshot()
20     boxes = model.predict([img])
21     for i, class_detections in enumerate(boxes):
22         rects = [r for r, score in class_detections]
23         labels = [model.labels[i] for j in range(len(rects))]
24         ml.utils.draw_predictions(img, rects, labels,
25                                 [(0, 255, 0)] * len(rects), format=None)
26     print(clock.fps(), "fps")
```

Listing 4: Sanntidsinferens med egentrent YOLOv8-modell på OpenMV N6.

4.4 Utfordringer og videre arbeid

Underveis møtte vi to praktiske utfordringer.

Den første dukket opp da vi skulle legge modellen på enheten. På Windows fikk vi ikke åpnet /rom-filsystemet på kameraet. Problemet lå i driverne. Vi løste det ved å gå over til Linux, der enheten ble tilgjengelig som den skulle.

Den andre utfordringen kom etter at dette var på plass. Bildefrekvensen på selve enheten ble for lav til at sanntidsdeteksjonen fungerte godt nok. Det gjorde at kjøring direkte på N6 ikke ble en god nok løsning i denne omgang.

Selv om kjøringen på enheten ikke ble bra nok, ga arbeidet oss verdifull erfaring. Vi fikk samlet inn og annotert treningsdata, og vi fikk testet maskinsyn gjennom Roboflow. Det viste at selve idéen fungerer, og ga oss et godt grunnlag for å jobbe videre.

Veien videre er å flytte inferensen vekk fra mikrokontrolleren og over til en PC med dedikert GPU. Kameraet brukes da bare som bildekilde, mens vi trener og kjører modellen selv med PyTorch. Det gir mer frihet i valg av modell og gjør feilsøkingen enklere. Ulempen er at systemet da ikke lenger er en frittstående enhet med lavt strømforbruk. Uansett hvilken plattform vi ender på, legger dette arbeidet et grunnlag for videre utvikling av visuell togdeteksjon, som på sikt kan supplere blokkdeteksjonen med informasjon om togtype, posisjon og retning.

5 Programvareutvikling og automatisert togkjøring

Denne delen av prosjektet handler om programvaren som styrer selve anlegget: hvordan datamaskinen snakker med togene, hvordan den vet hvor de befinner seg, og hvordan togene til slutt kan kjøre helt av seg selv uten å kolliderer. Arbeidet ble bygget opp trinnvis, fra én enkelt kommando til én lokomotiv, til et komplett system med kollisjonskontroll og autonom kjøring.

En erfaring som gikk igjen gjennom hele arbeidet er verdien av å starte i det små og bygge oppover, framfor å sette sammen alt på én gang. Det er langt enklere å feilsøke når man vet at hvert ledd under fungerer, og det gir samtidig en bedre forståelse av hele systemet.

5.1 Z21-sentralen og hvordan den styres

Anlegget styres av en Roco Z21-sentral, nærmere bestemt modellen «z21 start». Sentralen er bindeleddet mellom datamaskinen og skinnene: den legger et digitalt DCC-signal oppå kjørestrømmen, slik at hvert lokomotiv kan styres individuelt selv om alle står på samme spor. Hvert lokomotiv har en egen DCC-adresse, og dekodere i lokomotivet reagerer kun på kommandoer som er adressert til det.

5.1.1 Kommunikasjon over nettverk

Z21-en har et innebygd trådløst nettverk og kommuniserer med datamaskinen over UDP på port 21105. Protokollen er dokumentert i *Z21 LAN Protocol Specification*, som kan lastes ned gratis fra z21.eu. Datablader og protokollspesifikasjoner var gjennomgående en viktig kilde i dette arbeidet, og de fleste komponentene i anlegget har tilsvarende dokumentasjon tilgjengelig.

Selve meldingsformatet er enkelt, men har noen detaljer man må ha på plass før noe som helst fungerer. Hver pakke består av en lengdeangivelse, en header som sier hva slags melding det er, og selve dataene. Tall lagres med minst signifikante byte først. Den viktigste undergruppen av meldinger, kalt LAN_X, avsluttes i tillegg med en sjekksum beregnet som XOR over alle bytene i meldingen. Sentralen forkaster meldinger med feil sjekksum uten å gi tilbakemelding, noe som gjør feilen vanskelig å oppdage om man ikke vet om den.

5.1.2 Å kjøre et lokomotiv

For å sette fart på et lokomotiv sendes en kjørekommando med lokomotivets adresse, ønsket fartstrinn og retning. Systemet bruker 128-steps modus, der farten angis fra 0 til 126. Fartsbyten har noen særtrekk som måtte håndteres: verdien 1 er reservert til nødstop, slik at reelle fartstrinn starter på 2, og det høyeste bitet i byten angir kjøreretningen. Adressen deles i to byte, der lokomotiver med adresse over 127 markeres særskilt.

I tillegg må sporstrømmen slås på før noe skjer, og forbindelsen må holdes i live. Z21-en kobler fra klienter som har vært stille i mer enn ett minutt, slik at programmet sender en forespørsel hvert 30. sekund for å beholde forbindelsen.

5.1.3 Sporveksler

Sporveksler styres som DCC-tilbehør med egne adresser. Her måtte det gjøres to tilpasninger som ikke var åpenbare fra dokumentasjonen alene. For det første er adressene nullbaserte i protokollen, mens de vises fra 1 og oppover i Z21-appen og på håndkontrolleren, slik at veksler 1 sendes som adresse 0. For det andre er det ikke nok å sende én kommando: vekselen må først aktiveres og deretter deaktiveres igjen, med en kort pause imellom. Uten deaktiveringspulsen står vekselmotoren under spenning og kan brenne seg ut.

Koblingen mellom vekselnummer på anlegget og DCC-adresse er lagt i datafilen `turnout_addresses.json`, slik at den kan endres uten å røre koden.

En egen utfordring var veksler V7, som er en treveis sporveksler av typen Roco 61160. Denne inneholder i realiteten to vekselmotorer og opptrer derfor som to adresser i systemet. For å kaste den riktig må begge motorene styres i rekkefølge, slik at den ene slippes fri før den andre kastes. Dette er løst med et eget felt i adressefilen for den andre adressen.

5.1.4 Å lese hvor togene er

Togstyring uten tilbakemelding er blind styring. For å vite hvor togene faktisk befinner seg brukes belegningsdetektorer av typen Roco 10808, som melder tilbake over R-Bus. Anlegget er delt inn i seksjoner, og hver seksjon har sin egen detektorinngang. Når et tog står i en seksjon, melder detektoren at den er belagt.

Programmet abonnerer på tilbakemeldinger fra sentralen ved oppstart, og mottar deretter en melding hver gang noe endrer seg. Meldingen inneholder én byte per detektormodul, der hvert bit representerer én inngang. Programmet leser ut bitene, sammenligner med forrige kjente tilstand, og melder videre kun de inngangene som faktisk har endret seg. R-Bus-tilbakemeldingen ble bekreftet fungerende 25. juni 2026.

Belegningsdetektorene forteller at *en* seksjon er opptatt, men ikke *hvilket* tog som står der. Til dette brukes RailCOM, en utvidelse av DCC der lokomotivdekoderen svarer tilbake med sin egen adresse. Detektorene på anlegget leser dette og sender adressen videre over CAN-bussen. Dekodingen av disse meldingene måtte kartlegges mot en ekte sentral, ettersom det nøyaktige byte-oppsettet ikke framgikk av dokumentasjonen. Dette ble gjort med et eget diagnoseprogram, `railcom_sniff.py`, som dumper rådataene fra sentralen. Med RailCOM på plass vet systemet ikke bare at et spor er opptatt, men også hvilket lokomotiv som står der og hvilken vei det kjører.

5.2 Fra første kommando til fungerende styring

Arbeidet startet i det små. Det første programmet gjorde ikke annet enn å slå på sporstrømmen og gi ett lokomotiv fart, uten sensorer og uten grafisk grensesnitt. Dette er bevart som `drive_trains.py`, og er fortsatt den enkleste måten å kontrollere at man i det hele tatt når fram til Z21-en.

Deretter ble tilbakemeldingen tatt i bruk. Programmet `test_sensors.py` kjører ikke togene i det hele tatt, men abonnerer på alt sentralen sender og skriver det ut. Det ble brukt til å finne ut hvilke detektorinnganger som svarer til hvilke steder på anlegget, ved å kjøre et lokomotiv rundt og notere hva som lyste hvor. Programmet er fortsatt det første stedet man går når detektorene ikke ser ut til å gi utslag, ettersom det skiller mellom to helt ulike feil: enten endrer bytene seg, og da ligger problemet i tolkningen av dataene, eller så gjør de det ikke, og da ligger problemet i kablingen.

Med begge deler på plass kunne de kombineres. Resultatet var et program som satte farten automatisk ut fra hvilken seksjon toget kjørte inn i, styrt av en enkel tabell som koblet detektorinngang til fart. Dette var den første formen for automatikk i systemet, og la grunnlaget for alt som kom etter.

5.3 Utvikling av brukergrensesnittet

Brukergrensesnittet ble utviklet i fire generasjoner, der hver bygget på erfaringene fra den forrige.

5.3.1 Versjon 1: testbanen

Den første versjonen ble laget til en enkel testbane, en oval sløyfe med to sporveksler på toppen som gjorde at toget kunne ta enten den ytre buen eller en rett snarvei. Grensesnittet tegnet banen som fem seksjoner og lyste opp den seksjonen toget befant seg i, basert på tilbakemeldingen fra detektorene.

Det viktigste med denne versjonen var at sporvekslene faktisk fungerte. Programmet valgte vei tilfeldig hver gang toget passerte et bestemt punkt, kastet vekslene i begge ender av snarveien, og viste den planlagte ruten i grensesnittet før toget kom fram. Dermed var både styring, tilbakemelding og veksling bekreftet på en oversiktlig bane før det ble forsøkt på hele anlegget.



Figur 8: Første versjon av grensesnittet, laget til testbanen.

5.3.2 Versjon 2 og 3: hele anlegget

Neste steg var å utvide til hele anlegget, som er vesentlig mer komplisert enn testbanen med sine ni sporveksler og over tjue seksjoner. Her ble det innført en tydelig oppdeling av programvaren i tre lag, som har vist seg å holde gjennom resten av prosjektet:

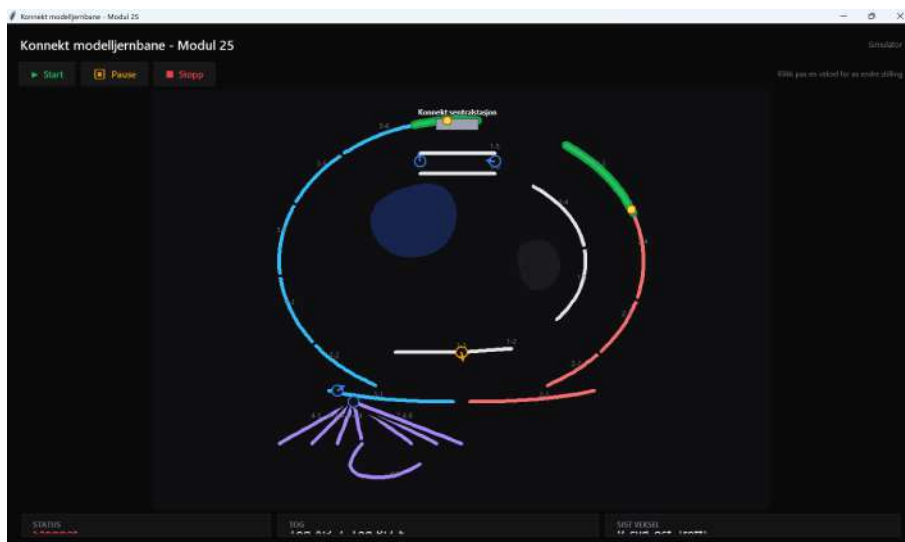
- **Geometri** – hvor sporene ligger på skjermen, altså hvordan banen tegnes.
- **Topologi** – hvordan seksjonene henger sammen, og hvilke sporveksler som ligger mellom dem.
- **Beleggskilde** – hvor informasjonen om opptatte spor kommer fra.

Denne oppdelingen gjør at det samme kartet kan mates med data fra ulike kilder. Under utviklingen ble det brukt en simulert kilde som spilte av en forhåndsdefinert kjøring, slik at

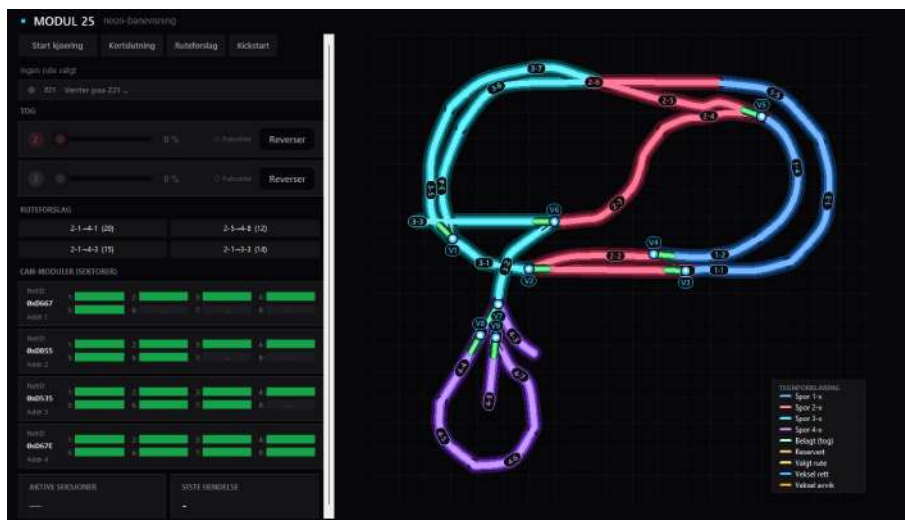
grensesnittet kunne testes uten å ha anlegget tilgjengelig. Når man så koblet til den ekte sentralen, var det bare beleggskilden som ble byttet ut, mens resten av programmet var uendret.

Topologien er beskrevet i datafilen `topology.json`, der hver seksjon oppgir hva som ligger i hver ende. Enden kan enten være en annen seksjon eller en sporveksel, og for sporveksler angis det hvilken seksjon som nås ved rett stilling og hvilken som nås ved avvik. Dette gjør at programmet kan regne ut hvilke seksjoner et tog faktisk kan kjøre til, ut fra hvordan vekslene står akkurat nå. Denne beregningen er kjernen i både kollisjonskontrollen og den autonome kjøringen.

Versjon 2 og 3 var begge skrivebordsprogrammer laget med tkinter, der versjon 3 hadde en visuelt tydeligere framstilling med fargekoding per spor og en oversikt over detektormodulene.



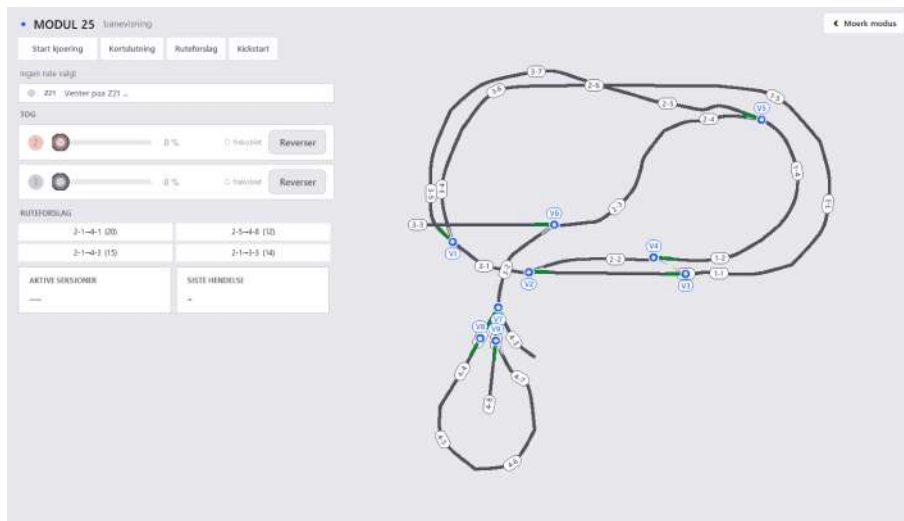
Figur 9: Begynnelsen på å lage hele grensesnittet



Figur 10: Andre versjon av grensesnittet, utvidet til hele anlegget.

5.3.3 Versjon 4: styring i nettleseren

Den fjerde og nåværende versjonen flytter hele grensesnittet inn i nettleseren. Bakgrunnen var praktisk: et skrivebordsprogram kan bare betjenes fra den maskinen det kjører på, mens en nettside kan åpnes fra flere enheter samtidig. På en utstilling er dette en klar fordel, ettersom besøkende kan styre fra sin egen mobiltelefon.

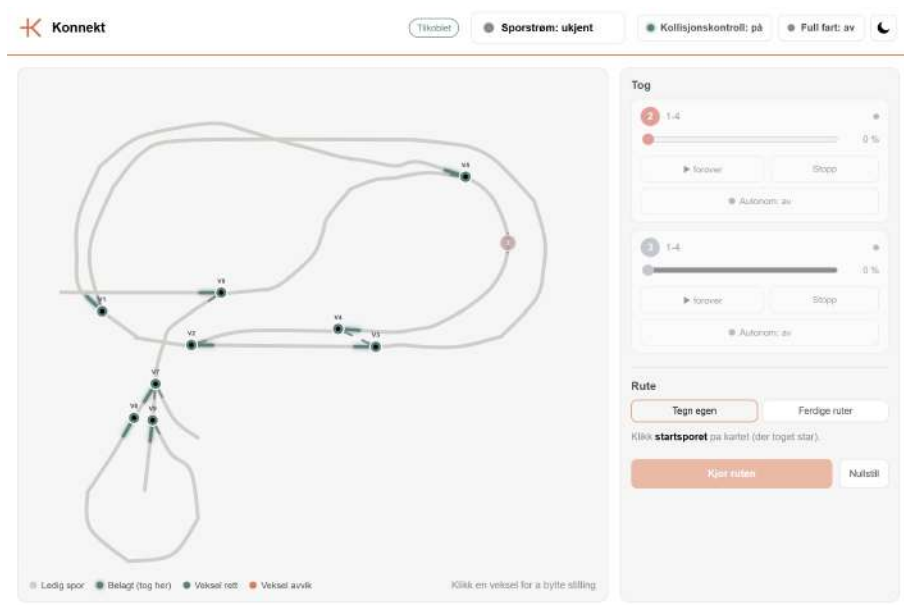


Figur 11: Tredje versjon, med fargekoding per spor og oversikt over detektormodulene.

Løsningen er bygget med FastAPI på serversiden. Kommandoer som fartsendring, vekselstilling og sporstrøm sendes fra nettleseren som vanlige forespørsler til serveren, som oversetter dem til Z21-kommandoer. Motsatt vei pushes all tilstand ut til nettleserne over WebSocket, slik at kartet oppdateres i det øyeblikket noe skjer framfor at nettleseren må spørre gjentatte ganger. Alle tilkoblede enheter ser dermed det samme til enhver tid.

Et viktig prinsipp i denne versjonen er at serveren eier sannheten om tilstanden. Nettleseren viser bare det serveren forteller, og all logikk om kollisjoner og autonom kjøring ligger på serversiden. Dette gjør at flere brukere kan styre samtidig uten at bildet spriker, og at sikkerhetsfunksjonene ikke kan omgås ved å manipulere nettsiden.

Selve backend-koden ble gjenbrukt uendret fra de tidligere versjonene. Både Z21-kommunikasjonen, beleggskilden, sporgeometrien og adressefilene er de samme filene som tkinter-versjonene bruker, slik at det kun er presentasjonslaget som er nytt.



Figur 12: Fjerde og nåværende versjon: togstyringen som en side i nettleseren.

5.3.4 Fartsbegrensning

En praktisk detalj i den siste versjonen er at farten normalt er begrenset til halvparten av det lokomotivene faktisk klarer. Full fart er for fort for anlegget, og med barn som betjener grensesnittet på utstillinger er risikoen for avsporing reell. Fartsregulatoren i grensesnittet viser fortsatt 0 til 100 prosent, men skalerer mot det gjeldende taket. Grensen kan låses opp med et passord for den som vet hva hen gjør.

Begrensningen håndheves på serveren, ikke i nettleseren. En klient som sender en for høy fart får den klippet uansett, noe som er nødvendig ettersom nettsiden kan manipuleres fra nettleserens utviklerverktøy.

5.4 Kollisjonskontroll

Med flere tog på samme bane oppstår behovet for å hindre sammenstøt. Kollisjonskontrollen kjører på serveren og vurderer situasjonen på nytt hver gang et tog melder seg i en ny seksjon eller et spor endrer belegg. Den fungerer uavhengig av om toget kjøres manuelt eller autonomt.

5.4.1 To former for fare

Vakten ser etter to situasjoner. Den første er at to tog befinner seg i samme seksjon, noe som aldri skal forekomme. Skjer det likevel, nødstoppes begge og det gis et tydelig varsel.

Den andre og viktigste er at et tog i bevegelse har et annet tog i seksjonen foran seg. Da stoppes det innkommende toget før det når fram. Ett-blokks framsyn er nødvendig nettopp fordi detektorene melder per seksjon: hadde systemet ventet til togene var i samme seksjon, ville det vært for sent.

Et tog som står stille regnes ikke som en fare, slik at varselet ikke blir hengende på et tog som allerede er stoppet.

5.4.2 Hvilken vei er framover?

En sentral utfordring er at systemet ikke uten videre vet hvilken retning et tog kjører i forhold til banens geometri. Det avhenger av hvordan lokomotivet er satt på sporet.

Løsningen er å bruke seksjonen toget nettopp forlot som referanse. Den ligger per definisjon bak toget, og filtreres derfor bort når naboene vurderes. Uten dette ville vakten stoppe toget for et opptatt spor det allerede hadde passert trygt, noe som i praksis gjorde at togene stoppet langt oftere enn nødvendig.

Dette får en direkte konsekvens når brukeren snur kjøreretningen. Et opptatt spor som lå foran toget ligger nå bak det, og er ikke lenger en fare. Systemet flytter derfor bak-referansen når retningen snus, slik at toget slippes fri framfor å bli nødstoppet på nytt så snart det får fart.

5.4.3 Å ikke stole blindt på detektorene

En feil som kostet betydelig feilsøking var falske kollisjonsvarsler. Årsaken var at systemet i utgangspunktet stolte på beleggsinformasjonen alene. Problemet er at en detektor også slår ut på togets eget bakre vognsett når nabosporet deler vekselende med sporet toget står på, i tillegg til at støy kan gi utslag.

Løsningen var å kreve at belegget stemmer overens med et annet togs kjente posisjon før det regnes som fare. En seksjon regnes derfor som sperret bare i to tilfeller: enten står et annet lokomotiv der bekreftet av RailCOM, eller sporet er fysisk belagt *og* det er et annet togs sist

kjente posisjon. Det siste tilfellet fanger opp et tog som står stille og har mistet RailCOM-kontakten, som ellers ville vært usynlig for systemet.

5.4.4 Betjening

Kollisjonskontrollen kan slås av og på fra grensesnittet, slik at anlegget kan kjøres fritt ved behov. Tilstanden deles mellom alle tilkoblede nettlesere.

Når et varsel kvitteres ut, gis togene tilbake farten de hadde før nødstoppen, og de skjermes mot ny nødstopp helt til de har kommet seg til en ny seksjon. Uten denne skjermingen ville de blitt stoppet igjen i samme øyeblikk. Kvittering gir gass uavhengig av om faren fortsatt er reell, ut fra at brukeren har sett situasjonen og tar ansvar for den.

5.5 Autonom togkjøring

Den mest avanserte funksjonen er autonom kjøring, der et tog kjører rundt på anlegget helt av seg selv. Autonom kjøring kan slås på for hvert enkelt lokomotiv, og flere tog kan kjøre autonomt samtidig.

5.5.1 Grunnprinsippet

Systemet fungerer som en tilstandsmaskin per lokomotiv, drevet av tilbakemeldingene fra detektorene. For hvert steg gjør programmet følgende:

1. Finn seksjonene foran toget, altså de kjørbare naboene minus den toget kom fra.
2. Velg en seksjon som er trygg, og reserver den.
3. Still sporvekslene som trengs, og gi lokomotivet fart.
4. Når detektoren melder at toget er framme, frigi sporet bak og gjenta.

Er alt foran opptatt, venter toget til noe blir ledig. Er det et blindspor uten vei videre, snur det og kjører tilbake.

Autonom kjøring er helt avhengig av RailCOM. Uten en melding om hvilket lokomotiv som står hvor, vet ikke systemet hvor toget befinner seg, og toget blir stående i ro til det melder seg.

5.5.2 Reservasjoner

Kollisjonshindringen mellom flere autonome tog løses med reservasjoner. Hver seksjon kan kun eies av ett tog om gangen, og et tog reserverer både seksjonen det står i og den det sikter mot. Et annet autonomt tog vil dermed aldri velge en seksjon som allerede er tatt, og de to togene holder seg i bevegelse uten å komme i veien for hverandre.

5.5.3 To stegs framsyn

Den enkeltendringen som gjorde autonom kjøring pålitelig var å planlegge to steg framover framfor ett. Bakgrunnen er at flere seksjoner på anlegget har sporveksel i *begge* ender. Stiller man bare vekselen som trengs for å komme inn i neste seksjon, står utgangsvekselen fortsatt feil når toget ruller inn, og på en kort seksjon rekker toget fram til den før detektoren har meldt ankomst og programmet har rukket å kaste. Resultatet er at toget kjører gjennom en sporveksel mens den stilles, med feil spor eller kortslutning som konsekvens.

Løsningen er å slå sammen kravene for begge stegene og stille alle vekslene mens toget fortsatt står i utgangsseksjonen. Når toget så ankommer neste seksjon, står vekslene videre allerede riktig, og det ruller uavbrutt gjennom. Dette er samme prinsipp som en forhåndsdefinert rute bruker, bare anvendt i et rullende vindu.

5.5.4 Serialisering av stegene

Et beslektet problem viste seg som tilsynelatende tilfeldige feilkastede sporveksler. Årsaken var ikke feil vekselkrav, men at to steg kunne starte samtidig for samme lokomotiv. Selve stegsekvensen tar et par sekunder, ettersom toget stoppes, vekslene kastes, det ventes litt, og farten gis tilbake. Kom det en ankomstmelding midt i denne sekvensen, ble et nytt steg planlagt parallelt, og vekselkommandoene fra de to stegene flettet seg inn i hverandre.

Løsningen var å serialisere stegene per lokomotiv. Mens en sekvens pågår, markeres det at lokomotivet er opptatt, og en ankomst som kommer i mellomtiden noteres i stedet for å utløse et nytt steg umiddelbart. Når sekvensen er ferdig, vurderes situasjonen på nytt.

5.5.5 Vaktbikkje

Ettersom autonom kjøring er fullstendig avhengig av tilbakemeldingene, stopper alt opp dersom en melding uteblir. Dette kan skje ved en død seksjon der detektoren ikke gir utslag, eller om RailCOM-kontakten faller ut.

For å håndtere dette er det lagt inn en vaktbikkje. Når toget får fart mot en seksjon, startes en tidtaker som forventer en ankomstmelding innen tolv sekunder. Kommer meldingen, avvæpnes tidtakeren. Uteblir den, duler systemet toget forsiktig og vurderer situasjonen på nytt, og stopper til slutt toget om det fortsatt ikke skjer noe. Uten denne funksjonen ville et tog kunne bli stående i ro på ubestemt tid uten at noe tydet på hva som var galt.

5.5.6 Samspill med manuell styring

Autonom kjøring slås automatisk av for et lokomotiv dersom brukeren griper inn, enten ved å justere farten, trykke stopp eller starte en forhåndsdefinert rute. Dette gjør at manuell styring alltid har forrang, og at brukeren ikke må huske å slå av automatikken først.

En begrensning er verdt å nevne: en ekte låsesituasjon der to tog står nese mot nese løses ikke automatisk. Begge stopper trygt, men et av dem må snus for hånd.

5.6 Veien videre

Systemet slik det står i dag håndterer normal drift godt, men det er flere områder der det naturlig kan videreutvikles.

5.6.1 Automatisk løsning av låsesituasjoner

Slik kollisjonskontrollen fungerer nå, stopper den togene trygt når de er på vei mot hverandre, men den løser ikke situasjonen. Står to tog nese mot nese, blir de stående til noen snur et av dem for hånd. Dette er en bevisst begrensning, ettersom en automatisk løsning krever at systemet tar en beslutning om hvilket tog som skal vike, og det var ikke prioritert innenfor prosjektets tidsramme.

En videreføring bør derfor la systemet løse slike situasjoner selv. I praksis innebærer det at programmet oppdager låsen, velger hvilket tog som skal rygge, finner en seksjon det kan trekke

seg tilbake til, og deretter slipper det andre toget forbi. Logikken for å finne kjørbare naboseksjoner finnes allerede i den autonome kjøringen, slik at grunnlaget for dette er på plass.

5.6.2 Prioritering ved felles sporveksel

Når to autonome tog sikter mot samme sporveksel, avgjøres det i dag av hvilket tog som rakk å reservere seksjonen først. Det fungerer, men gir et vilkårlig resultat: det ene toget venter uten at det ligger noen vurdering bak hvorfor akkurat det skulle vente.

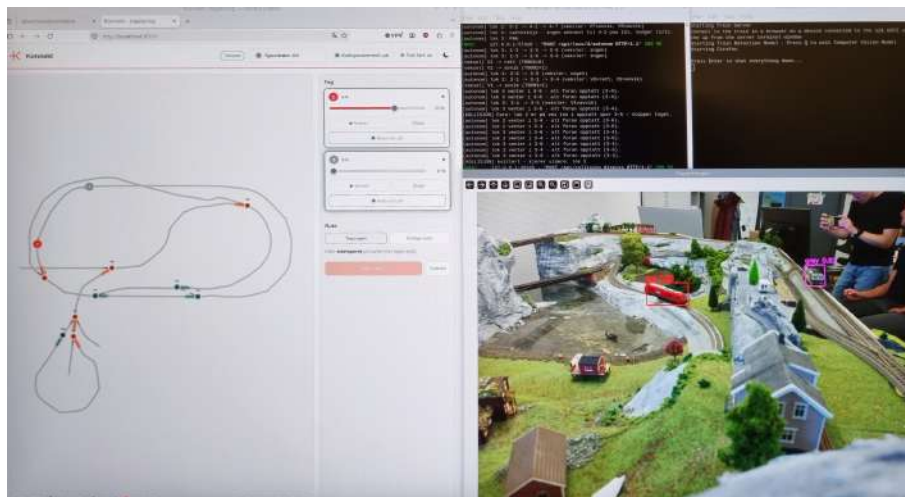
En bedre løsning er å innføre en form for prioritering, slik det gjøres på ekte jernbane. Toget med høyest prioritet får passere først, mens det andre venter i en egnet seksjon. Prioriteten kan settes fast per lokomotiv, eller beregnes ut fra situasjonen, for eksempel slik at et tog som allerede har ventet lenge går foran, eller at et tog som er i bevegelse prioriteres framfor et som står stille. Sistnevnte gir jevnere kjøring, ettersom det unngår å bremse ned et tog som allerede er i gang.

Dette ville også gjort anlegget mer interessant som demonstrasjon, siden prioritering mellom tog er en problemstilling som er direkte gjenkjennelig fra virkelig togframføring.

5.6.3 Mer variert kjøremønster

Et tredje forbedringspunkt er at togene bør snu retning oftere. I dag kjører et autonomt tog stort sett samme vei helt til det møter et blindspor og blir tvunget til å snu. Resultatet er at enkelte deler av anlegget brukes mye, mens andre knapt besøkes.

Ved å la togene snu retning med jevne mellomrom, enten tilfeldig eller etter et fastsatt mønster, vil kjøringen bli mer variert og hele anlegget tas i bruk. For besøkende gir dette et betydelig mer levende inntrykk enn et tog som kjører den samme runden om og om igjen. Funksjonen for å snu finnes allerede, ettersom den brukes ved blindspor, slik at endringen i hovedsak handler om når den skal utløses.



Figur 13: Raspberry Pi kjører programmet på skjermen

5.7 Raspberry Pi

Modellen har en Raspberry Pi 4B 8GB med Raspberry Pi AI Camera som kjører togsserveren og Computer Vision modellen. Raspberry Pi'en har repoet til koden liggende på `/home/konnekt/konnektmodelljernbane` i filstrukturen. Der kan nyeste versjon av koden hentes ned ved å kjøre kommandoen `git pull`. I samme mappe ligger start-skriptet kalt `start.sh` dette skriptet starter automatisk opp togsserveren og AI-modellen på en ryddig måte på skjermen.

Dette skriptet kan også kjøres ved å dobbeltklikke på Start Trainsikonet på skrivebordet. For å gjøre klart til togkjøring trenger man bare å være koblet på z21 wifien og kjøre Start Trains". For å avslutte togkjøring må man gå inn på terminalen øvers til høyre på skjermen der det står Press Enter to shut down"og trykke Enter på tastaturet.

5.7.1 Videreutvikling av Raspberry Pi-oppsettet

Også maskinvaren rundt Raspberry Pi-en har rom for forbedring, både når det gjelder betjening og ytelse.

Det første punktet gjelder Bluetooth. Den innebygde Bluetooth-modulen i Raspberry Pi-en fungerer for dårlig i praksis, med ustabil tilkobling til tastatur og mus. Dette er særlig merkbart på utstillinger, der oppsettet skal fungere umiddelbart uten feilsøking. En ekstern Bluetooth-modul med bedre rekkevidde og mer stabil forbindelse bør derfor anskaffes.

Det andre punktet gjelder selve datamaskinen. Anlegget kjører i dag på en Raspberry Pi 4B med 8 GB minne, som håndterer togserveren og kameradeteksjonen, men merkbart tregere enn ønskelig. Det gir en liten forsinkelse i betjeningen, som selv om den ikke er kritisk, er tydelig nok til å merkes når man styrer togene. En nyere og kraftigere modell vil gi lavere responstid og et mer behagelig grensesnitt.

Det tredje punktet henger sammen med utvidelsen av kameradeteksjonen omtalt i avsnitt 3.6. Dagens løsning bruker ett AI-kamera, som dekker en begrenset del av anlegget. Skal flere deler av banen dekkes, trengs flere kameraer, og med flere kameraer følger også behov for mer regnekraft til å ta imot og behandle dataene. En mulig løsning er å bruke flere Raspberry Pi-er med hvert sitt kamera, som deler informasjon seg imellom, framfor å belaste én enhet med alt. Det gir samtidig den fordel at systemet fortsatt fungerer delvis dersom én enhet faller ut.

6 Nettside

Som en del av sommerprosjektet i 2026 ble det utviklet en informasjonsnettside for Konnekts modelljernbaneprosjekt. Nettsiden er ment å fungere som en digital utvidelse av modellen og skal gjøre det enklere for besøkende å lære mer om prosjektet. Den er tilgjengelig ved å skanne en QR-kode som er plassert ved modelljernbanen, eller direkte på <https://konnekt-modelljernbane.vercel.app/>.

6.1 Formål

Hovedformålet med nettsiden er å gi besøkende mer informasjon enn det som er mulig å formidle gjennom den fysiske modellen alene. Nettsiden samler informasjon om prosjektets bakgrunn, utvikling og innhold på ett sted, og gjør den lett tilgjengelig både under og etter et besøk.

Besøkende kan blant annet lese om hva modellen viser, hvordan prosjektet har utviklet seg over tid, hvem som har bidratt til utviklingen, samt se bilder fra byggeprosessen og den ferdige modellen. Målet er å gjøre prosjektet mer tilgjengelig og gi en bedre forståelse av modelljernbanens rolle som læringsplattform innen jernbanefag.

Et annet viktig mål har vært å utvikle en løsning som enkelt kan videreføres av fremtidige studentgrupper. Nettsiden er derfor bygget med en oversiktlig struktur og gjenbrukbare komponenter, slik at nytt innhold og nye funksjoner kan legges til uten større endringer i den eksisterende løsningen. På denne måten kan nettsiden videreutvikles i takt med modelljernbaneprosjektet og fungere som en langsiktig informasjonsplattform.

6.2 Design og planlegging

Før utviklingen av nettsiden startet ble det gjort en rekke vurderinger knyttet til målgruppe, struktur og design. Målet var å utvikle en løsning som både er enkel å bruke for besøkende og enkel å videreutvikle for fremtidige studentgrupper.

6.2.1 Behovsanalyse

Den primære målgruppen for nettsiden er besøkende som ser modelljernbanen og ønsker mer informasjon om prosjektet. Siden mange vil åpne nettsiden ved å skanne en QR-kode ved modellen, ble det lagt stor vekt på at den skulle fungere godt på mobil. Samtidig ble nettsiden utviklet med et responsivt design som også gir en god brukeropplevelse på større skjermer, blant annet fordi den vil være tilgjengelig via lenker på Konnekts nettsider og dermed også kunne besøkes fra PC.

I tillegg til å være en informasjonskanal for besøkende, skulle nettsiden fungere som dokumentasjon av prosjektet. Det var derfor viktig å legge til rette for at fremtidige studentgrupper enkelt kan oppdatere innholdet og videreutvikle løsningen.

Nettsiden ble derfor utviklet etter et mobile-first-prinsipp, der layout og funksjonalitet først ble tilpasset mobilskjermer før de ble utvidet til større skjermer.

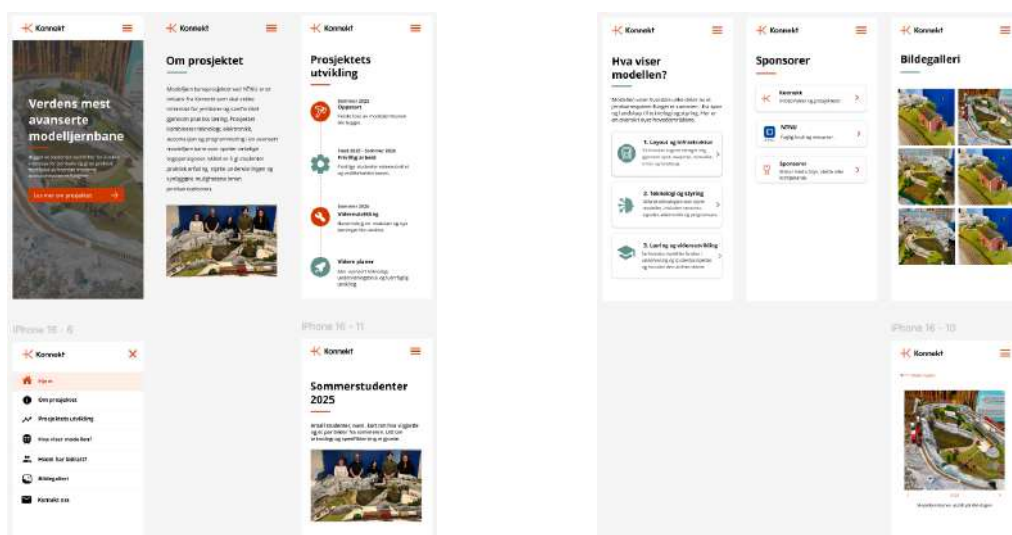
6.2.2 Designvalg

Det ble valgt et enkelt og minimalistisk design for å la innholdet og modelljernbanen stå i fokus. Under utviklingsprosessen ble det tatt kontakt med Konnekt for å diskutere innhold og få tilgang til deres designmanual. Nettsidens visuelle uttrykk er basert på denne, blant annet gjennom bruk av Konnekts fargepalett, typografi og grafiske profil. Samtidig ble antallet farger bevisst begrenset for å skape et ryddig og helhetlig uttrykk som ikke tar oppmerksomheten bort fra

innholdet. Det ble også lagt vekt på god kontrast, tydelige overskrifter og tilstrekkelig luft mellom elementene for å gjøre innholdet lett å lese.

Gjennom utviklingsprosessen ble ulike løsninger vurdert og justert, blant annet for navigasjon, presentasjon av bidragsyttere og bildegalleri. Målet var å finne løsninger som både er intuitive for brukeren og enkle å vedlikeholde.

Før implementasjonen startet ble det utarbeidet enkle skisser i Figma for å utforske ulike løsninger for layout, navigasjon og plassering av innhold (se figur 14). Skissene ga en helhetlig oversikt over nettsidens struktur og fungerte som et utgangspunkt for utviklingen. Etter hvert som nettsiden ble utviklet, ble både design og sideinndeling justert for å oppnå en mer oversiktlig og brukervennlig løsning. Den endelige nettsiden ble derfor noe annerledes enn de opprinnelige skissene.



Figur 14: Nettsidens design i Figma.

Det ble også lagt vekt på universell utforming gjennom hele utviklingsprosessen. Nettsiden ble testet med Lighthouse for å identifisere forbedringsområder knyttet til blant annet tilgjengelighet, og nødvendige justeringer ble gjort fortløpende. I tillegg ble det kontrollert at nettsiden kan navigeres ved hjelp av tastatur alene, blant annet gjennom testing av tab-rekkefølge og synlige fokusmarkeringer. Videre ble det benyttet semantisk HTML, beskrivende alternative tekster for bilder og god kontrast mellom tekst og bakgrunn. Bilder og videoer er optimalisert for ulike skjermstørrelser og enheter, slik at nettsiden fungerer godt både på mobil, nettbrett og PC, og kan brukes med mus, tastatur og berøringsskjerm. Samlet bidrar disse tiltakene til at nettsiden er tilgjengelig for flest mulig, uavhengig av funksjonsevne eller hvordan brukeren velger å navigere.

6.2.3 Informasjonsarkitektur

For å gjøre nettsiden oversiktlig ble informasjonen delt inn i egne sider med hvert sitt tema. Strukturen består av en forside med overordnet informasjon om prosjektet, en side om prosjektets utvikling, en side om hva modellen viser, et bildegalleri, en oversikt over hvem som har bidratt til prosjektet og en kontaktside.

Denne inndelingen gjør det enkelt å finne frem til ønsket informasjon uten at enkeltsider blir unødvendig lange. Under planleggingsfasen ble flere ulike måter å organisere innholdet på vurdert, blant annet løsninger med flere undersider og en mer detaljert inndeling av informasjonen. Den endelige strukturen ble valgt fordi den ga den beste balansen mellom oversikt og detaljnivå. Ved å samle beslektet informasjon på færre sider blir navigasjonen enklere, samtidig som antallet valg brukeren må ta reduseres. Dette gjør nettsiden mer intuitiv å bruke og reduserer risikoen for at brukeren går seg vill eller navigerer til feil side. Strukturen gir også et

godt utgangspunkt for videre utvikling, siden nye sider eller innhold kan legges til uten å endre den overordnede oppbygningen av nettsiden.

6.3 Innhold

Nettsiden består av seks hovedsider som til sammen gir en helhetlig presentasjon av modelljernbaneprosjektet. Innholdet er organisert slik at besøkende enkelt kan finne informasjon om prosjektet uten å måtte navigere gjennom mange undersider.

Forsiden fungerer som en introduksjon til prosjektet og gir en kort beskrivelse av modelljernbanen, prosjektets formål og hva banen skal brukes til. Siden *Prosjektets utvikling* viser hvordan prosjektet har utviklet seg gjennom ulike faser. Innholdet er organisert i klikkbare informasjonsbokser for sommerprosjektet 2025, sommerprosjektet 2026 og prosjektet i bruk, slik at brukeren enkelt kan fordype seg i den perioden eller aktiviteten som er av interesse.

Siden *Hva viser modellen?* fungerer som en oversiktsside og leder videre til undersider som beskriver modelljernbanens layout, infrastruktur og teknologiske løsninger. På siden *Om oss* presenteres Konnekts rolle i prosjektet, hvilke studenter som har utviklet banen, samt hvilke sponsorer som har bidratt.

Bildegalleriet samler bilder og videoer fra utviklingen av modelljernbanen. Ved å velge et bilde eller en video åpnes en egen side der mediet vises i større format sammen med tilhørende beskrivelse. Galleriet kan filtreres etter år, slik at man enkelt kan følge utviklingsperioden som er av interesse. Til slutt inneholder kontaktsiden informasjon om hvordan besøkende kan komme i kontakt med prosjektet dersom de har spørsmål, ideer eller ønsker å bidra.

6.4 Implementasjon

Etter at design og struktur var bestemt, ble nettsiden implementert med moderne webteknologier som legger til rette for høy ytelse, enkel videreutvikling og godt vedlikehold. Ved valg av teknologier ble det lagt vekt på å benytte løsninger som er mye brukt i bransjen, godt dokumentert og godt egnet for utvikling av responsive nettsider. Et annet viktig kriterium var at utvikleren allerede hadde erfaring med teknologiene, noe som bidro til en mer effektiv utviklingsprosess.

Flere deler av brukergrensesnittet er implementert som gjenbrukbare React-komponenter, blant annet navigasjonskort, bildegalleri, bildevisning, navigasjonsmeny, footer og bildetekster. Dette reduserer duplisert kode og gjør det enklere å vedlikeholde og videreutvikle løsningen.

Layouten tilpasses automatisk ulike skjermstørrelser ved hjelp av responsive komponenter og Tailwind CSS. På mobil vises innholdet i én kolonne, mens større skjermer benytter en to-kolonne-layout der dette gir bedre lesbarhet. Bilder, videoer og navigasjon skaleres automatisk slik at nettsiden fungerer godt på mobiltelefoner, nettbrett og PC.

6.4.1 Teknologivalg

Tabell 5 gir en oversikt over teknologivalgene for nettsiden. Nettsiden er utviklet med rammeverket Next.js, som er bygget på React og benytter App Router for organisering av sider og navigasjon. Next.js ble valgt fordi rammeverket tilbyr god ytelse, støtte for optimalisering av bilder, enkel ruting mellom sider og en oversiktlig prosjektstruktur med gjenbrukbare komponenter. Dette gjør løsningen godt egnet både for videre utvikling og fremtidig vedlikehold.

Programmeringsspråket TypeScript ble benyttet gjennom hele prosjektet. TypeScript bygger på JavaScript, men legger til statisk typing som gjør det enklere å oppdage feil under utvikling og bidrar til mer robust og lesbar kode.

Til utforming av brukergrensesnittet ble Tailwind CSS benyttet. Dette gjør det mulig å style

komponenter direkte i koden ved hjelp av forhåndsdefinerte klasser, noe som ga en effektiv utviklingsprosess og et konsistent design på tvers av hele nettsiden.

For ikoner ble biblioteket Lucide React benyttet. Dette ga tilgang til et moderne og enhetlig sett med ikoner som enkelt kunne tilpasses resten av designet.

Bilder håndteres ved hjelp av Next.js-komponenten `next/image`, som automatisk optimaliserer bildene og bidrar til kortere lastetid og bedre ytelse.

Utviklingen ble gjennomført lokalt med Visual Studio Code og Git som versjonskontroll. Prosjektet ble lagret i et GitHub-repositorium, mens hosting av nettsiden skjer gjennom Vercel, som er tett integrert med Next.js og gjør det enkelt å publisere nye versjoner etter hvert som prosjektet videreutvikles.

Tabell 5: Oversikt over teknologiene som ble benyttet i utviklingen av nettsiden.

Teknologi	Bruksområde
Next.js	Rammeverk for utvikling av nettsiden, inkludert ruting, rendering og optimalisering.
React	Komponentbasert bibliotek for utvikling av brukergrensesnitt.
TypeScript	Programmeringsspråk som gir statisk typing og bedre feilhåndtering under utvikling.
Tailwind CSS	CSS-rammeverk brukt til styling og responsivt design.
Lucide React	Bibliotek for ikoner brukt i navigasjon og innhold.
next/image	Komponent for optimalisering og effektiv håndtering av bilder.
Git	Versjonskontroll under utviklingen.
GitHub	Lagring av kildekode og samarbeid om prosjektet.
Vercel	Hosting og automatisk publisering av nettsiden.

6.4.2 Struktur

Prosjektet er organisert etter mappestrukturen til Next.js App Router, der hver mappe under `app/` representerer en egen side på nettsiden. Felles komponenter er samlet i en egen `components`-mappe, mens data som brukes flere steder er flyttet til egne datafiler. Dette gjør det enklere å finne frem i prosjektet, gjenbruke komponenter og oppdatere innhold uten å påvirke resten av løsningen.

KONNEKT-MODELLJERNBANE/

```
app/  
  components/  
  data/  
  bildegalleri/  
  hva-viser-modellen/  
  prosjektets-utvikling/  
  om-oss/  
  kontakt-oss/  
  
public/  
  images/  
  videos/  
  reports/  
  
package.json  
next.config.ts
```

README.md

...

En mer detaljert oversikt over prosjektstrukturen, inkludert beskrivelse av de viktigste filene og mappene, er dokumentert i prosjektets README.md.

6.5 Hosting

Nettsiden hostes på Vercel og er koblet til prosjektets GitHub-repositorium. Når endringer pushes til hovedgrenen (`main`), bygges og publiseres en ny versjon automatisk. Dette gjør det enkelt å videreutvikle nettsiden, ettersom fremtidige studentgrupper kun trenger å oppdatere GitHub-repositoriet før Vercel håndterer bygging og publisering.

Ved prosjektets avslutning benyttes Vercels *Hobby*-plan. Denne er tilstrekkelig så lenge nettsiden brukes til et utdanningsprosjekt og har et begrenset antall besøkende. Dersom nettsiden på et senere tidspunkt får vesentlig høyere trafikk, eller bruken vurderes som kommersiell, bør det vurderes å oppgradere til en *Pro*-plan i tråd med Vercels retningslinjer.

For å sikre langsiktig drift anbefales det at både GitHub-repositoriet og Vercel-prosjektet eies av Konnekt fremfor enkeltpersoner. På denne måten kan nye studentgrupper gis tilgang uten at nettsiden blir avhengig av personlige kontoer.

6.6 Hvordan videreføre prosjektet

Nettsiden er utviklet med tanke på at den skal kunne videreføres av fremtidige studentgrupper. Prosjektet er dokumentert gjennom kildekoden og en tilhørende README.md, som beskriver prosjektstrukturen, hvordan nettsiden startes lokalt og hvordan nytt innhold legges til.

6.6.1 Videre drift

For å sikre en langsiktig og robust løsning anbefales det at GitHub-repositoriet eies av Konnekt fremfor en personlig konto. Tilsvarende bør Vercel-prosjektet eies av organisasjonen og kobles til organisasjonens GitHub-repositorium. På denne måten kan nye studentgrupper eller ansatte enkelt gis nødvendige tilganger, samtidig som videre drift og utvikling ikke blir avhengig av den opprinnelige utvikleren.

Dersom nettsiden kun skal vedlikeholdes av én fast person i organisasjonen, kan Vercel Hobby være tilstrekkelig. En mulig løsning er da at denne personen eier både GitHub-repositoriet og Vercel-prosjektet gjennom personlige kontoer, mens øvrige bidragsyttere utvikler på egne branches og oppretter Pull Requests. Den ansvarlige personen kan deretter kvalitetssikre, merge og publisere endringene. Dette gjør det mulig å videreutvikle nettsiden uten kostnader, men innebærer samtidig at prosjektet blir avhengig av én navngitt person.

Dersom flere personer eller fremtidige studentgrupper skal kunne videreutvikle nettsiden med automatisk deploy fra et organisasjonseid GitHub-repositorium, anbefales Vercel Pro. Dette skyldes at Vercel Hobby er utviklet for personlige prosjekter og har begrenset støtte for organisasjonseid samarbeid, mens Vercel Pro er utviklet for team og gjør det mulig å gi flere utviklere tilgang til prosjektet og publisere endringer direkte.

6.6.2 Oppdatering av innhold

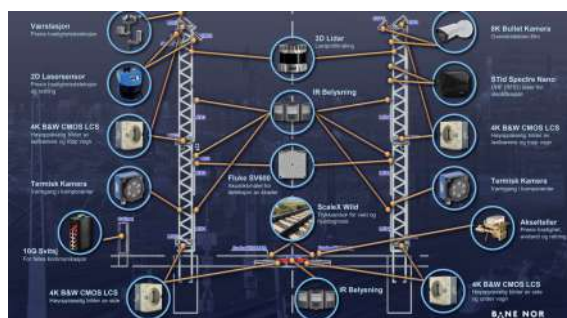
Nye bilder og videoer legges i henholdsvis `public/images` og `public/videos`, mens metadata oppdateres i `galleryImages.ts`. Bidragsyttere oppdateres i `contributors.ts`, og nye sider kan legges til ved å opprette nye mapper under `app/`. Prosjektets README beskriver denne prosessen mer detaljert.

7 Train Gate

En Train Gate er en portal togene kjører gjennom, der et sett sensorer måler tilstanden på togmateriellet i fart. Formålet i virkeligheten er å oppdage feil før de utvikler seg til driftsproblemer, ved å kontrollere hjul, lagre, last og andre komponenter uten at toget må stoppe. Dette er en teknologi som er i bruk i jernbanesektoren, og som derfor er godt egnet til å demonstreres i modellskala.

I dette prosjektet er det bygget en fysisk modell av en slik portal. Stengene er 3D-printet, limt sammen og malt, slik at portalen står ferdig på anlegget som en del av landskapet.

Det er viktig å presisere at det ikke er gjort noe elektronisk arbeid på Train Gate innenfor prosjektets ramme. Arbeidet har bestått av design og idéutvikling, samt en kartlegging av hvilke sensorer og hvilken teknologi det er realistisk å gjenskape i modellskala. Delkapittelet er derfor først og fremst en dokumentasjon av grunnlaget for videre arbeid.



Figur 15: Train Gate-portalen.

7.1 Kartlagte sensorer og løsninger

Et gjennomgående spørsmål i kartleggingen har vært hvor tett man kan komme på den virkelige teknologien, og hvor det i stedet er mer hensiktsmessig å simulere eller illustrere prinsippet. Modelltog er elektriske og oppfører seg annerledes enn virkelig materiell, slik at noen målinger ikke lar seg gjenskape direkte. I slike tilfeller er poenget å vise hva teknologien gjør, framfor å måle noe som ikke finnes.

7.1.1 Termisk kamera

I virkeligheten brukes termisk måling til å oppdage varmgang i lagre og bremses, som er et tydelig tegn på begynnende feil. På et modelltog er det lite varme å måle, ettersom togene er elektriske og går kaldt.

Prinsippet kan likevel demonstreres. En mulighet er å legge en varmekilde skjult i en av vognene, slik at det termiske kameraet faktisk får et utslag å reagere på. Da vises både målingen og varslingen på en måte som er lett å forstå for besøkende, selv om varmen er kunstig framkalt.

7.1.2 Vektmåling og hjuldiagnostikk

Vekt og hjultilstand måles i virkeligheten med trykksensorer i sporet. Tilsvarende kan gjøres i modellskala med en lastcelle, som gir et målbart utslag når toget kjører over. Dette kan brukes både til å anslå vekten av togsettet og til å se etter avvik mellom hjulene, som er utgangspunktet for hjuldiagnostikk.



Figur 16: Train Gate-portalen.

7.1.3 Kameraovervåking

Et eller flere kameraer kan gi en generell videostrøm av toget som passerer gjennom portalen. Den praktiske begrensningen her er størrelsen: kameraene må være små nok til å passe inn i portalen uten å bryte med modellens skala.

Det ble undersøkt å bruke OpenMV-kameraets infrarøde variant til dette formålet. Dette lot seg imidlertid ikke gjennomføre, ettersom det manglet en Raspberry Pi eller en Windows-maskin å koble kameraet til. All tilgjengelig maskinvare var på det tidspunktet i bruk til togstyringen og kameradeteksjonen omtalt i avsnitt 3.6.

7.1.4 Identifikasjon av tog

En Train Gate må vite hvilket tog som passerer for at målingene skal ha noen verdi. I virkeligheten løses dette ofte med RFID eller tilsvarende merking.

I dette anlegget er en slik løsning unødvendig. RailCOM gir allerede togets adresse gjennom CAN-forbindelsen til belegningsdetektorene, slik det er beskrevet i avsnitt 5.1. Informasjonen om hvilket tog som befinner seg hvor er dermed tilgjengelig i systemet fra før, og kan brukes direkte til å knytte målinger til riktig togsett.

7.1.5 Hastighet

Hastighet er en naturlig størrelse å måle i en portal, og kan løses med en egen sensor. Også her kan RailCOM benyttes i stedet, ettersom systemet kjenner både posisjon og fartstrinn til hvert lokomotiv. En egen sensor vil gi en mer direkte måling, men er ikke strengt nødvendig for å få et brukbart tall.

7.1.6 Værstasjon

En værstasjon er ikke en del av en Train Gate i seg selv, men kan bidra med tilleggsdata som temperatur, luftfuktighet og lufttrykk. I en visning gir dette flere målinger å vise fram, og setter

samtidig sensormålingene i en sammenheng, ettersom vær og temperatur påvirker togmateriell i virkeligheten.

7.1.7 Akustisk måling

Den originale Train Gate benytter akustisk måling for å oppdage avvikende lyd fra hjul og lagre, som er en av de mest brukte metodene for tidlig feildeteksjon. Dette ble undersøkt, men det ble ikke funnet en løsning som fungerte innenfor prosjektets ramme. Utfordringen ligger både i at lydbildet fra et modelltog skiller seg vesentlig fra et virkelig tog, og i at det kreves en form for referanse for å kunne skille normal lyd fra avvikende lyd.

Dette gjenstår som den mest krevende, men også den mest interessante, delen av en framtidig Train Gate på anlegget.

7.2 Veien videre

Train Gate står i dag ferdig som fysisk modell, med sensorvalgene kartlagt men ikke implementert. Neste steg er derfor å ta i bruk elektronikken.

Et naturlig utgangspunkt er å begynne med de sensorene som gir målbare resultater uten å måtte simuleres, altså lastcellen for vekt og hjuldiagnostikk, samt en enkel værstasjon. Disse gir konkrete tall å vise fram, og krever ingen kunstig framkalt tilstand for å fungere.

Deretter kan kameraovervåkingen legges til, forutsatt at det skaffes en egen Raspberry Pi til formålet. Dette henger sammen med anbefalingen i avsnitt 5.7.1 om å utvide med flere Raspberry Pi-er og kameraer, ettersom en Train Gate med kamera i praksis er nettopp et slikt ekstra kamerapunkt.

Den termiske målingen og den akustiske målingen bør tas til slutt, ettersom begge krever at det først bestemmes hvordan tilstanden skal framkalles eller simuleres. Her er det verdt å merke seg at en fungerende demonstrasjon ikke nødvendigvis krever en ekte feil, men en tydelig og forståelig visning av hva teknologien ville oppdaget.

8 Steg-for-steg: demontering, frakt og montering



Figur 17: Utstyr til stand

Pakkeliste stand

Må ha tilgang på bord på stand for pc-skjermer

Store ting:

- 1-2 stk pc-skjermer
- Mus og tastatur
- USB-C-kabel til RPI
- 1-2 HDMI-kabler og HDMI til micro-HDMI-adapter
- Krakkene

Små ting:

- Grå pussekloss
- Grå plastting for å sette på tog på sporene
- Gorillalim
- Ekstra skinner i alle typer, gjerne minst 1–2 som er loddet, i tilfelle noe går galt
- Ledninger og noen Wago-rekkeklammer
- Snipper/blå avbiter-/avisoleringstang
- Teip
- Liten skrutrekker

-
- Multimeter
 - Ekstra bolter og muttere
 - Ev. vogner o.l. som skal brukes under utstilling
 - Lok, vogner og biler som er løse på modellen

1. Fjerne løse gjenstander

Det aller meste er limt fast, men fjern alle vogner, lok og biler, de sitter ikke fast. Det er veldig viktig at lokene fjernes, siden de er svært dyre.

Legg alle løse gjenstander i plastboksen under bordet (den uten kabler og bokser). Denne boksen er ment for alt som skal være med på stand, så legg gjerne ned andre ting samtidig, for eksempel ekstra vogner og ekstra loddede skinner i tilfelle noe går galt under standen (se pakkeliste for stand).

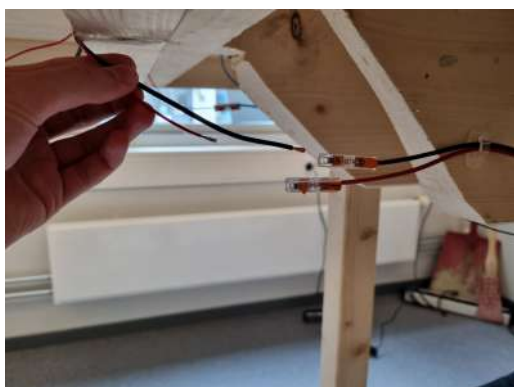
Viktigst er det å fjerne løse gjenstander fra den gamle modulen som skal tippes, men fjern gjerne fra den nye, mindre modulen også, så er man helt sikker.

2. Kabler mellom stor og liten modul

Før man skyver den lille modulen bort fra den store, må noen kabler kobles fra:

- Den blå ethernetkabelen løsnes fra den sorte detektorboksen på den lille modulen. Ved opprigg skal den kobles inn i den øverste av de to CAN-utgangene. Det er veldig viktig at den **ikke** settes inn i R-bus.
- To kabler til, en rød og en svart, må også løsnes. Dette gjøres enkelt ved å vippe opp den oransje låsen på klemmen, da løsner og faller den ene siden av kabelen ut.

Sikre disse kablene så de ikke ligger og slenger under transport.



Figur 18: Jord, spenning og CAN må kobles mellom modulene.

3. Sammenkoblingspunkt mellom ny og gammel modul

Modulene er koblet sammen med to gjengestaver og fire muttere. Løsne mutterne og ta ut disse og gjengestavene, deretter er det bare å skyve bordplatene fra hverandre. Ved sammensetting gjør man det samme, bare i motsatt rekkefølge. Ta vare på alt av muttere og gjengestaver.



Figur 19: Gjengestaver og muttere holder modulene sammen.

4. Tilting av stor modul

- Sikre at bremsene på hjulene er på. Hvis modulen ikke har hjulene på, bør de settes på før tilting. Dette gjøres ved å løfte modulen og føre hjulene med gjengstangen inn i hullet i de fire hjørnene på modellen. Dette bør gjøres på en jekktralle så man sørger for at alle hjulene settes i bakken samtidig for å fordele vekten.
- På langsiden ved stasjonen sitter to bein, markert 1 og 4. Ta av to muttere øverst på hvert bein og én nederst på hvert bein. Få noen til å løfte og støtte bordet mens boltene tas av. Man kan trenge en hammer for å slå ut de to nederste boltene.
- Når beina er av, begynner man å vippe bordet sakte ned til det treffer bunn. Ta vare på muttere, skiver og bolter, og legg dem på et lurt sted.

Vær litt forsiktig under tilting og gjør det sakte, i tilfelle noe skulle skje eller lim/figurer/gjenstander løsner.



Figur 20: Bein festest/løsnes med bolter og muttere øverst og nederst.

5. Ta av hjul (ev.)

Samme fremgangsmåte på begge modulene, men for den store kan bordet **ikke** være tiltet. Det må da løftes opp og festes på beina igjen så det står stødig. Det kan være gunstig med en jekketralle bygget opp med litt tre, eller å bære det manuelt hvis man er nok folk.

Bordet må bæres høyt nok til at hjulene faller ut. De er kun festet med gjengestenger og har ingen låsemekanisme, men kan sitte litt fast. Lirk da forsiktig til de løsner. Det er viktig at bordet ikke settes i bakken før vekten fordeles over alle hjulene.



Figur 21: Montering/Demontering av hjul

6. Krakk/platå

Vi har bygget to krakker som skal plasseres i hjørnet mellom modulene, for å gi lave tilskuere bedre utsikt over modellen. Den lengre krakken skal stå inntil den større modulen, og den mindre krakken skal plasseres inntil den mindre modulen.

Disse krakkene er laget slik at de kan plasseres mellom beina til den mindre modellen under frakting.

7. Prepping av modellen før visning

- Sett på bremsene på hjulene
- Hev stor modul igjen og fest bein o.l. med bolter og muttere
- Ev. ta av hjul
- Fest stor og liten modul sammen
- Fest kablene mellom stor og liten modul igjen
- Gå over alle skinner med pusseklossen for å fjerne støv, smuss og andre partikler. Det er veldig viktig å gå over de to små skinnene på halvsirkelen på den nye modulen, siden disse ikke leder strøm, renser de seg ikke selv, så de må jevnlig gås over med klossen.
- Sett tilbake løse gjenstander
- Sjekk at toget kjører, at alle seksjoner/skinner fungerer, og at alle bokser lyser og er ok