KAPITTEL 4
PROGRAMMERING AV EN ROBOT

Hittil har vi snakket om hvordan roboten beveger seg og hvordan man bruker den i en robotcelle, men virkeligheten er ikke like enkel. Roboten må jobbe i et samspill med resten av fabrikken, både andre maskiner og mennesker, og dette er ikke like enkelt hver gang.

Det finnes mange måter å programmere på, mange programmeringsspråk man kan velge, og mange problemstillinger man må tenke over når man først skal programmere en robot.

I dette kapittelet skal vi diskutere forskjellige måter å programmere på, presentere flere kjente kommunikasjonsprotokoller som brukes innen robotikk, og se på forskjellige programmeringsspråk og utviklingsmiljø, som ROS/ROS2 som robotingeniører bruker i arbeidshverdagen.

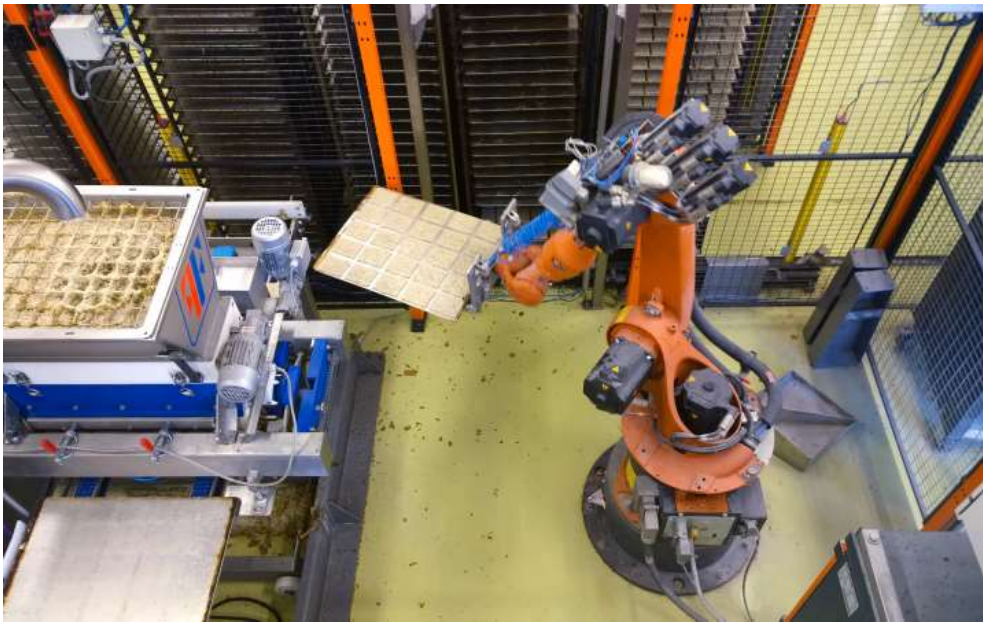
En Robot, ei Sag og en Pakkesniffer

Dette er Viddal Automation. Det er en bedrift som leverer robotisering av prosesser for hovedsakelig små og mellomstore bedrifter. De jobber med stort sett alt og ingen utfordring er for stor.

Viddal Automation holder til på Håhjem i Skodje kommune (uttalelse Skøje) utenfor Ålesund.

Siden 2013 har de jobbet i alle disipliner fra robotisert sveising og fresing, til kunstig intelligens og datasyn. Med dette har de god erfaring med hvordan man skal programmere roboter, både hvordan man kan gjøre det enkelt og trygt for kunden og hvor skoen trykker når man skal løse kompliserte problemstillinger.

Steffen Viddal,
gründer og daglig leder
av Viddal Automation



La du merke til at jeg skrev om en pakkesniffer i tittelen? Vi snakker om den i slutten av kapittelet.

TreeCon er en bedrift i Namsos som produserer takstoler. De har en enorm fabrikk som automatisk produserer takstoler basert på tegninger fra kunden.

TreeCon kontaktet Viddal Automation med et problem: Ei sag som gikk for tregt. Av alle prosessene i fabrikken, var denne sagen en flaskehals som senket produksjonshastigheten. TreeCon ville derfor bytte den ut med en robot som skulle kunne erstatte sagen fullstendig, men også gjøre jobben raksere.

Dette er ikke like enkelt som det høres ut som. Det vanskelige her er at man må installere roboten uten å endre på resten av fabrikken. Du må ha samme kommunikasjon, samme robotcelle, samme resultat, bare raskere. Vi skal se litt senere hvordan Viddal klarte dette, og hvilke problemer de møtte underveis.

Oppgave

Utfordringene er mange i dette tilfellet:

- Du har et kommunikasjonsnettverk mellom alle maskinene i fabrikken, og du vil ikke endre på resten av kommunikasjonsprotokollene eller meldingene i resten av fabrikken for å tilpasse den nye roboten. Roboten må derfor motta akkurat de samme meldingene som den gamle saga gjorde og gjøre den samme jobben.
- Du har en fabrikk som er fullt operativ, der du ikke vil ha nedetid i flere dager eller uker mens du installerer den nye roboten
- Du vil sikre at roboten gjør nøyaktig den samme jobben som saga, bare fortere
- Systemet må være fremtidsrettet så det er enkelt å utvide, eventuelt med flere sensorer eller med flere roboter.

Hvordan ville du løst dette?

Saga som var installert ved TreeCon. Den henger fra taket, hvor en maskin beveger seg i riktig vinkel for at saga kan kutte plankene som ligger på samlebåndet

Legg merke til hvor denne roboten henter platen fra. Hvordan ville du programmmert dette?

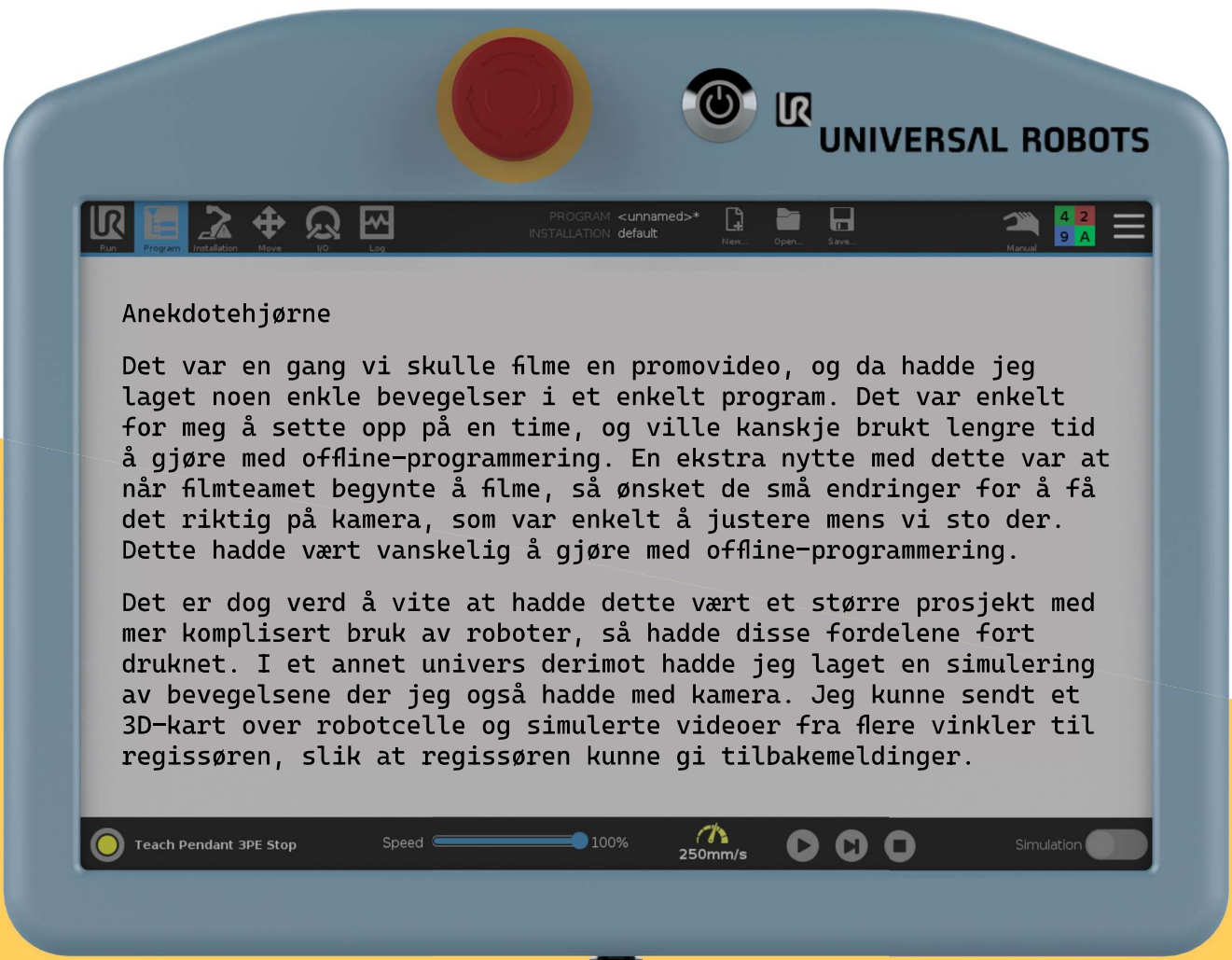
OFFLINE- VS ONLINE-PROGRAMMERING

Når man først snakker om programmering av roboter, skiller vi først mellom *offline-* og *online-*programmering. Dette er skillet mellom hvordan man skal programmere roboten fysisk. Programmerer man roboten online, så står man foran roboten, gjerne ved å bruke en kontrollenhet, og ser de fysiske bevegelsene av roboten mens man programmerer. Med offline-programmering jobber man uten roboten, og heller gjerne med et simuleringsverktøy så man kan se robotens bevegelser.

Online-programmering

Online-programmering er når man bruker selve roboten til å programmere hvor den skal. Man står med en kontrollenhet i hånden, flytter robotarmen til den neste ønskede posisjonen, lagrer posisjonen, og gjør dette for hvert steg. Dette var den måten som Unimate ble programmert på.

Når man programmerer på denne måten vil man være helt sikker på hvordan roboten beveger seg, og man kan gjøre nødvendige småjusteringer som trengs på hver individuelle robotcelle.



Kontrollenheten til e-serien fra UR

Fordeler og ulemper med online-programmering

Det er ikke overraskende at jeg ikke er entusiastisk for online-programmering. Det er i mine øyne en gammeldags måte å jobbe på. Men det er ofte den eneste måten som leveres med robotene. De fleste robotleverandørene har en simulator eller visualiseringsverktøy, men disse koster også ekstra, og mange har lisenser som må fornyes. Det kan derfor hende at du blir nødt til å programmere online for å få automatisert en prosess.

Den største ulempen med online-programmering er sikkerhet. Du må stå nært, og av og til inni robotcellen for å programmere, og man får heller ingen tilbakemelding om hva roboten planlegger å gjøre når den beveger seg fra A til B. Det er sjeldent at roboten får problemer med selve bevegelse, men det største skaden er at roboten kræsjer i omgivelsene rundt seg.

En annen ulempe er at roboten ikke kan være operasjonell mens du programmerer. I enkelte bedrifter er det veldig kostbart om man må stanse produksjonen for å omprogrammere roboter, så man prøver å unngå det så mye som mulig.

Det er dog noen fordeler med denne typen programmering, og det er at man kan ta i bruk alle funksjonene som roboten tilbyr. Det fins mange kommandoer som et simulatorporgram ikke kan generere, fordi de som lagde simulatorprogrammet ikke visste eller tenkte over at den kommandoen eksisterte. Mange simulatorporgram støtter flere robotspråk, men det betyr også at de legger seg på mer generelle implementasjoner heller enn de kommandoene som er spesifikke for den roboten.

For at dette skal telle som en fordel, så må selvfølgelig du være flinkere å programmere den roboten enn et helt team med lang erfaring med å programmere simulatorer som genererer optimalisert kode... Så, mest sannsynlig ikke. Sorry.

Bilde av promovideoen med en KUKA KR6 R900 som skrur i en lyspære.

Offline-programmering

Når man skal programmere offline, så trenger man ikke en fysisk robot. Man er bokstavlig talt koblet fra roboten, derav navnet. Dette er den mest vanlige måten å programmere på om man jobber med store kompliserte automatiseringssystemer.

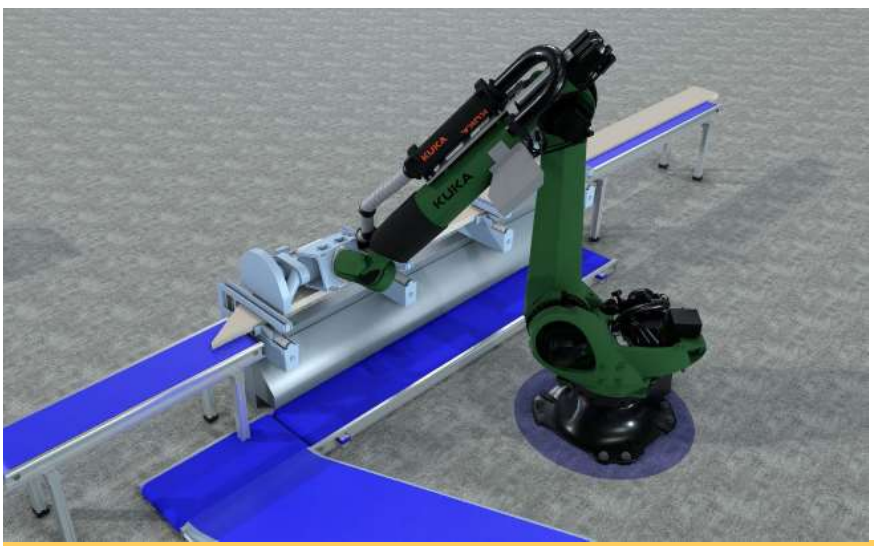
Enkleste eksemplet for å vise tankegangen rundt dette er CNC-maskiner og G-kode. Med G-kode skriver man generelle kommandoer om hvor man ønsker at endestykket på en drill eller annet verktøy skal være, og i teorien skal man ikke trenge å tenke på hvilken CNC-maskin man bruker.

Selv om G-kode tidligere ble programmert manuelt, så er det mest vanlig nå å generere den ut ifra det du planlegger å maskinere. Alle store 3D-modelleringsprogrammer har en mulighet til å eksportere modellen til G-kode, eller til en filtype som et annet program kan lage G-kode utav. Uansett hvordan du får G-koden, kan den brukes av hvilken som helst CNC-maskin eller 3D-printer (merk at det ønskede resultatet vil nok ikke være det samme). Man kan derfor skrive/generere G-kode uten å tenke på maskinen man bruker, og man kan nærmest laste koden rett over på maskinen når man ønsker å kjøre den.
Dette er hvordan man programmerer offline. Man har gjerne et simuleringsverkøy som kan vise robotens bevegelser, og bruker dette til å programmere roboten. Når man føler seg ferdig vil man kunne eksportere koden på et slikt format at man kan laste koden direkte over på roboten.

Det er verd å merke seg at man ofte kun tenker på hvor endestykket på roboten forflytter seg, hvilket verktøy den har, hva den plukker opp, osv. Da er det ikke så viktig lengre hvilken robot som gjør det, så lenge det blir gjort. Flere simulatorverktøy er laget for at man kun programmerer banen til endestykket, hvor man setter opp referanserammer og endestykkets bevegelse mellom dem, og at man etterpå velger hvilken robot som skal gjøre det og hvor den skal stå i forhold til bevegelsen. Når man er ferdig, genererer programmet en kode som er tilpasset akkurat den roboten du valgte.

Fordeler og ulemper med offline-programmering

Den største fordelen med offline-programmering, og hovedgrunnen til at de fleste store automasjonsbedriftene bruker dette er redusert nedetid. Man slipper å bruke roboten aktivt mens man programmerer, som betyr at roboten kan gjøre det vanlige arbeidet sitt mens du forbereder robotcellen til en ny produksjon.



Et simuleringsverktøy som brukes til offline-programmering er KukaSim. Her ser du oppsettet til sageroboten til Viddal. KukaSim kan brukes til å simulere og også automatisk programmere KUKA roboter. Andre alternativer er Visual Components (som Kuka Sim er bygget på) og RoboDK.

En annen fordel er at man kan teste hele produksjonsflyten i simulatoren. Det å kunne simulere fabrikken er en stor fordel, og en av hovedpillarene til industri 4.0, som vi kommer tilbake til. Det er også nyttig at man kan jobbe "robotagnostisk", altså at man ikke trenger den nøyaktige roboten for å programmere riktig. Når man også kun fokuserer på banene på endestykket, er det ikke problem å endre på endestykket i ettertid. Dette gjør det mulig å spesifisere detaljene rundt robotens og endestykkets konfigurasjoner etter at man har programmert banen.

Den største ulempen med offline-programmering er tiden det tar å sette opp et slikt system. Nå skal det sies at det ikke tar så veldig lang tid, så det er ikke en alt for stor ulempe. Men det er fortsatt ganske vanlig å bruke online-programmering hvis man bare skal gjøre noe enkelt. Jeg gjør det gjerne hvis jeg skal vise noe i undervisningen eller programmere en demonstrator på en stand eller noe. Da trenger man ikke et fullt simulert miljø for å få det til å funke.

Sensorstyrt programmering

Det å dele robotprogrammering inn i kun online- og offline-programmering er litt kunstig, siden det finnes mange måter å programmere en robot på. For eksempel kan man programmere roboten slik at den mottar en fresejobb i form av G-kode, og hvor du hardkodet inn en kalibreringsrutine som kjører før hver jobb. I tillegg kan man programmere roboten med algoritmer, beslutningstrær og tilstandsmaskiner, slik at roboten selv bestemmer hva den skal gjøre basert på sensordata. Jeg vil kalle dette *sensorstyrt programmering*, men det er ikke nødvendigvis et godt ord på det.*

Når man programmerer på høyere nivå, programmerer man ikke bare at roboten skal bevege seg fra A til B for så å gripe noe og flytte tilbake til A. For det fordrer at man vet hvor A og B er.
Et eksempel er såkallt *kasseplukk* eller *bin picking* som det heter på engelsk. Det er når det ligger mange gjenstander i en kasse hvor roboten skal plukke en. Du vet ikke hvor gjenstandene er på forhånd, og i enkelte tilfeller vet du kanskje heller ikke hvor kassen er.
Man må da bruke *sensorer* til å detektere hvor gjenstandene er i kassen og *algoritmer* for å generere en bane til roboten så den kan plukke den opp.

Her ser du en UR-robot med et Zivid-kamera montert på endestykket. Kan du komme på flere operasjoner eller sikkerhetsmekanismer som må til for å få til kasseplukk?

* Man kan ofte finne ordet *dynamisk programmering* på folkemunne. Men dette er noe annet på fagspråket. Dynamisk programmering er en form for matematisk beskrivelse av en optimaliseringsalgoritme. Reguleringsteknikk er en form for dynamisk programmering.

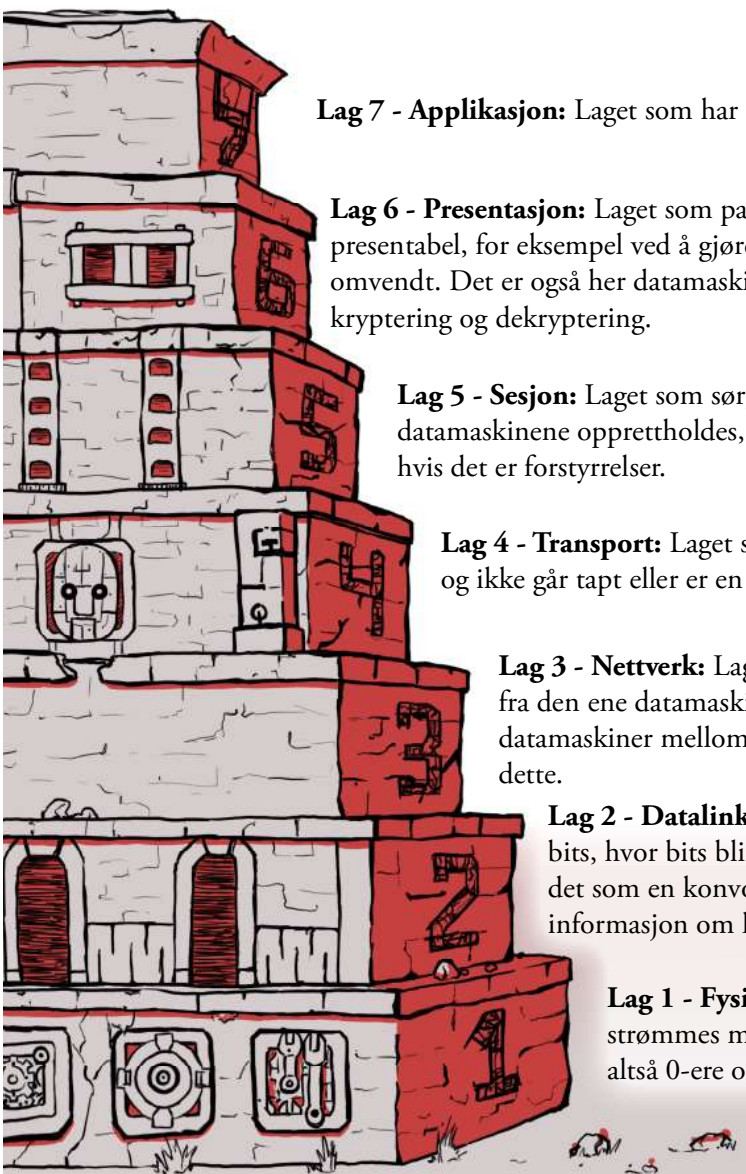
OSI- OG TCP/IP-MODELLEN

En av elementene som kan avgjøre om et prosjekt lykkes er hvor god egnet kommunikasjonen mellom maskiner. Krever du datastrømmer på under millisekunder eller er det viktigere at datapakker ikke forsvinner?

En kommunikasjonsprotokoll er et fastsatt sett med regler for hvordan to datamaskiner skal utveksle data. Dette sikrer at informasjonen blir både sendt og mottatt riktig, og at datamaskinene vet hvem og når det ble sendt og mottatt.

Kommunikasjonsprotokoller standardiserer alt fra hvilken data som sendes, hvordan dataen er innpakket, hvordan elektroniske signaler skal tolkes og hvilke kabler som skal brukes for å overføre signalene.

OSI-MODELLEN



Lag 7 - Applikasjon: Laget som har grensesnitt mellom forskjellige applikasjoner.

Lag 6 - Presentasjon: Laget som pakker eller pakker opp data så den er presentabel, for eksempel ved å gjøre bits om til tall eller bokstaver eller omvendt. Det er også her datamaskinen gjør komprimering, dekomprimering, kryptering og dekryptering.

Lag 5 - Sesjon: Laget som sørger for at kommunikasjonen mellom de to datamaskinene opprettholdes, og som kan gjenopprette kommunikasjonen hvis det er forstyrrelser.

Lag 4 - Transport: Laget som sørger for at pakken mottas uforandret og ikke går tapt eller er en duplikat.

Lag 3 - Nettverk: Laget som foreslår hvordan pakken skal sendes fra den ene datamaskinen til den andre. Det kan være flere datamaskiner mellom sender og mottaker, og dette laget håndterer dette.

Lag 2 - Datalink: Laget som sørger for en feilfri sending av bits, hvor bits blir pakket inn i en *ramme*. Man kan tenke på det som en konvolutt, som beskriver avsender og mottaker og informasjon om hvordan mottakeren skal håndtere innholdet.

Lag 1 - Fysisk: De elektriske eller optiske signalene som strømmes mellom datamaskiner og konverteres til *bits*, altså 0-ere og 1-ere.

OSI-MODELLEN

Det første som er lurt å snakke om er Open Systems Intercommunication Basic Refererance Model, altså OSIBR-modellen, som har blitt forkortet videre til OSI-modellen.
Denne modellen er standard for all kommunikasjon mellom datamaskiner, og deler inn kommunikasjonen i 7 lag.

Det er ikke så viktig å dykke så dypt inn i OSI-modellen, og det er også verd å vite at det som er beskrevet over er en grov forenkling og sikkert feil. Det viktigste er at du vet at kommunikasjon mellom datamaskiner og roboter ikke er helt rett frem. Det er også viktig å vite at ikke alle protokollene som er brukt i industrien implementerer alle lagene. Disse protokollene har derfor ikke de egenskapene i lagene over, og det er derfor opp til deg å implementere disse lagene selv hvis du

TCP/IP-MODELLEN

TCP/IP-modellen er også en modell som er sidestilt med OSI-modellen. Den er kjent for å være en praktisk implementasjon av OSI-modellen og er det vi kaller "internett". Den forenkler OSI-modellen til fire lag:

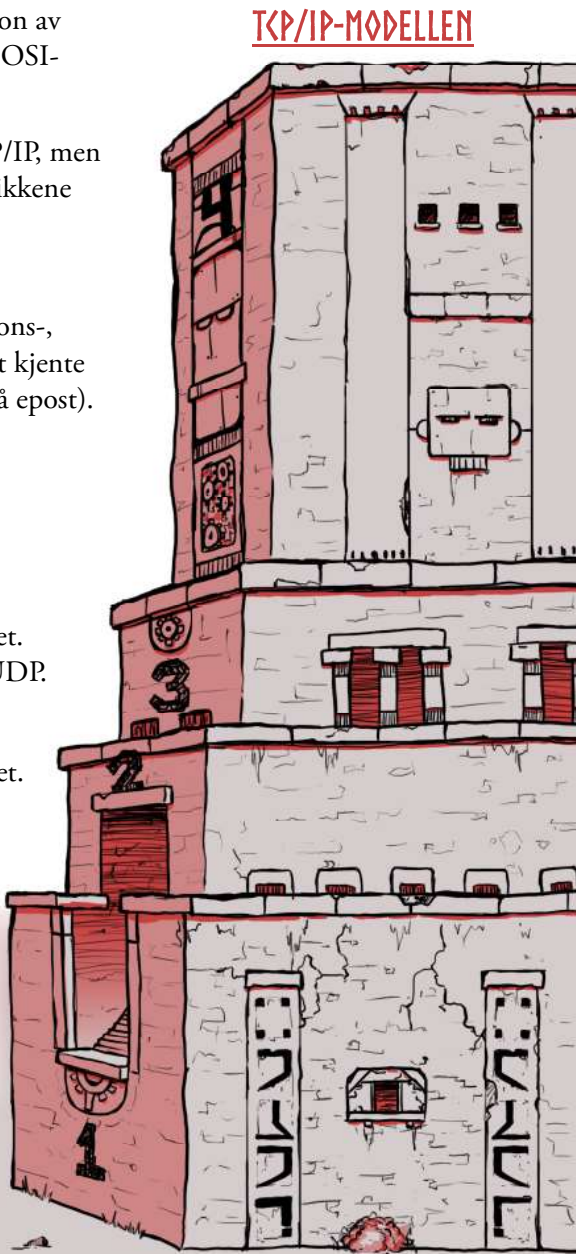
Det er ikke mange protokoller i industrien som bruker TCP/IP, men det blir flere og flere ettersom det blir vanligere å koble fabrikkene

Lag 4 - Applikasjon: Sammenslåing av sesjons-, presentasjons- og applikasjonslaget. De mest kjente protokollene er HTTP, FTP og SMTP (altså epost).

Lag 3 - Transport: Samme som transportlaget. Her dukker opp protokollene som TCP og UDP.

Lag 2 - Internett: Samme som nettverkslaget. Det er her IP-adressene blir brukt til å kartlegge hvor pakkene skal gå.

Lag 1 - Link: Sammenslåing av det fysiske og datalink-laget. Dette brukes av nettverkskort.



Ethernet-pluggen

Ok, jeg vil gjerne dedikere en liten del av boken til RJ45, også kjent som *Ethernet-pluggen*. Den er ekstremt vanlig i industrien og alle de protokollene nevnt i dette kapittelet enten bruker denne eller kan bruke den. Det er også den som var brukt til hustelefon og telefaks, som var veldig kult før i tiden.

> *En gang da jeg var liten og manglet en brikke i et nytt Legosett, tegnet jeg hvordan den så ut og fakset bildet til Lego i Danmark. Det er rart å tenke på at en kopi av tegningen min kanskje ligger i en arkivkjeller en plass der.*

Den har 8 kabler som er tvinnet i 4 par, hvor hvilket par som brukes til hva er litt avhengig av litt forskjellig ting (men det er verd å vite at man kan splitte en Ethernet-kontakt til to kabler, men med dårligere hastighet. Nyttig i en nødsituasjon.)

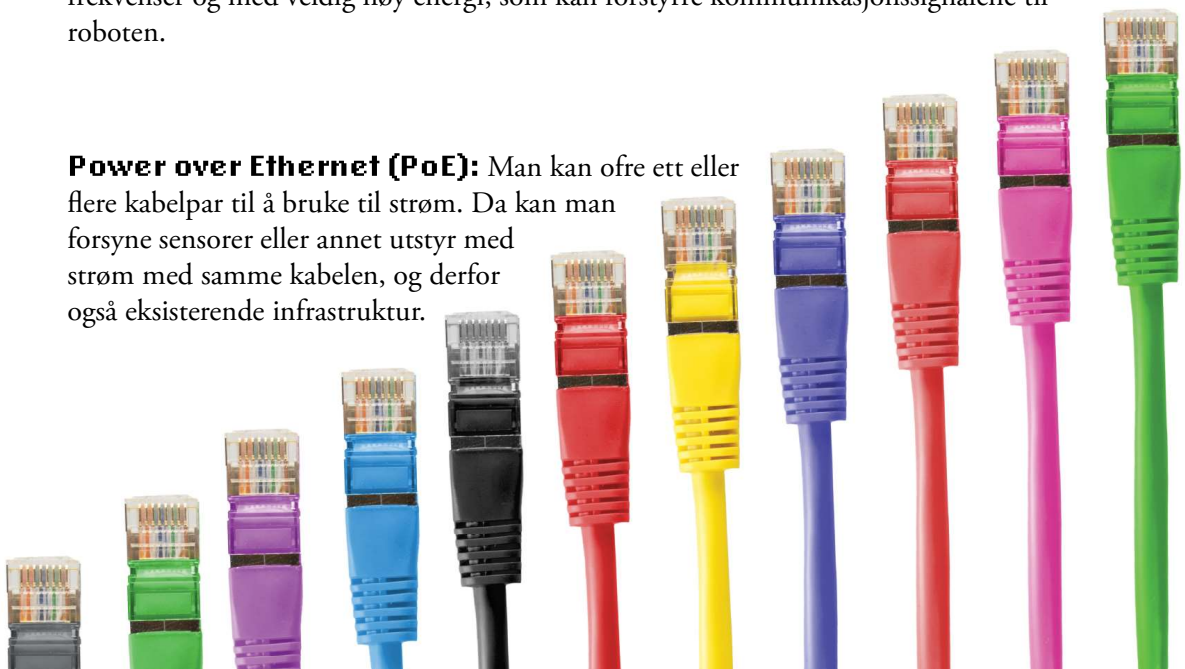
Det er også viktig å vite at det er enkelte egenskaper ved Ethernet-kabelen er nyttig i industrien, men som man ikke finner i vanlige kabler man kjøper i butikken:

Låsemekanisme: Ethernet-pluggen har vanligvis en låseklips, men disse er ikke veldig robust mot slitasje. Det finnes derfor låsemekanismer i metal, som sikrer at pluggen ikke ryker eller ødelegger datamaskinen hvis du napper i kabelen. De kan også fås med en gitt IP-grad.

Noen fortalte meg en gang at på Grillstad smøres kontaktene inn med vaselin, fordi det salte miljøet eirer kontaktene uavhengig av IP-grad.

Skjerming: Man kan få kabler med for eksempel aluminiumsfolie eller flettet kobber rundt. Dette gjør den motstandsdyktig mot ytre elektromagnetiske forstyrrelser. Dette er for eksempel nyttig innen sveising, siden lyset fra sveisebuen ikke bare er synlig lys, men mange andre frekvenser og med veldig høy energi, som kan forstyrre kommunikasjonssignalene til roboten.

Power over Ethernet (PoE): Man kan ofre ett eller flere kabelpar til å bruke til strøm. Da kan man forsyne sensorer eller annet utstyr med strøm med samme kabelen, og derfor også eksisterende infrastruktur.



`Min kjære Vim,`

Du er ikke bare en editor, men en verden hvor tastetrykk blir til magi. En håndtverkspartner som sparer meg tid når jeg hengir meg over til ditt svøpende lokkenett av kommandoer.

Der andre programmer prøver å gi meg alt med menyer og ikon, inviterer du meg til å lære, forstå og mestre. Hver kommando, fra `ciw` til `daw` bringer du mine tanker til kode; et samkvem skapt for programmering.

Med `e:` starter vi et nytt eventyr sammen; en stille symfoni på Cherry MX Brown. Jeg utforsker deg med `3w` og `gg`, og du kaster meg gjennom teksten som blott står for meg. Du venter i spenning på mine ønsker, og adlyder hvert ømme tastetrykk. Hvor har du vært hele mitt liv? Hvorfor har ingen fortalt meg om deg tidligere?

Jeg trodde lenge at jeg var en flink programmerer, men innså mine feil da jeg traff deg. Du er utemt og vill; hudløs og mild. Et mesterverk det tar en livstid å kjenne. Jeg prøvde å fornekte din eksistens i mange år, men innser at ekspertene hadde rett: Vim er er fantastisk verktøy i programmering.

Når man innser at programmering ikke er å *skrive kode*, men å *redigere kode*, ser man fylden av ditt vesen. Jeg trenger ikke å skrive raskt, når jeg danser over tastene i en rumba som kan flytte, kopiere og navigere med få tastetrykk.

Hos deg finner jeg orden i kaoset, en pirrende balanse der vi deler kontrollen. Med et enkelt `yw` kan jeg kopiere akkurat det jeg ønsker, og `p` lar meg plassere det som om vi former verden sammen. Jeg merker den tause forståelsen mellom oss – hvert trykk, hvert grep, så presist og uanstrengt. Når jeg avslutter med `:wq`, føles det som et løfte — en avslutning som omslutter alt vi har skapt, og likevel en invitasjon til å fortsette – igjen og igjen.

Så her står jeg, ydmyk og betatt, vel vitende om at reisen med deg aldri vil ta slutt. Jeg er din for alltid, Vim – fra første `:e` til siste `:q!`, en evig hengivenhet i hvert tastetrykk.

Industrielle Kommunikasjonsprotokoller

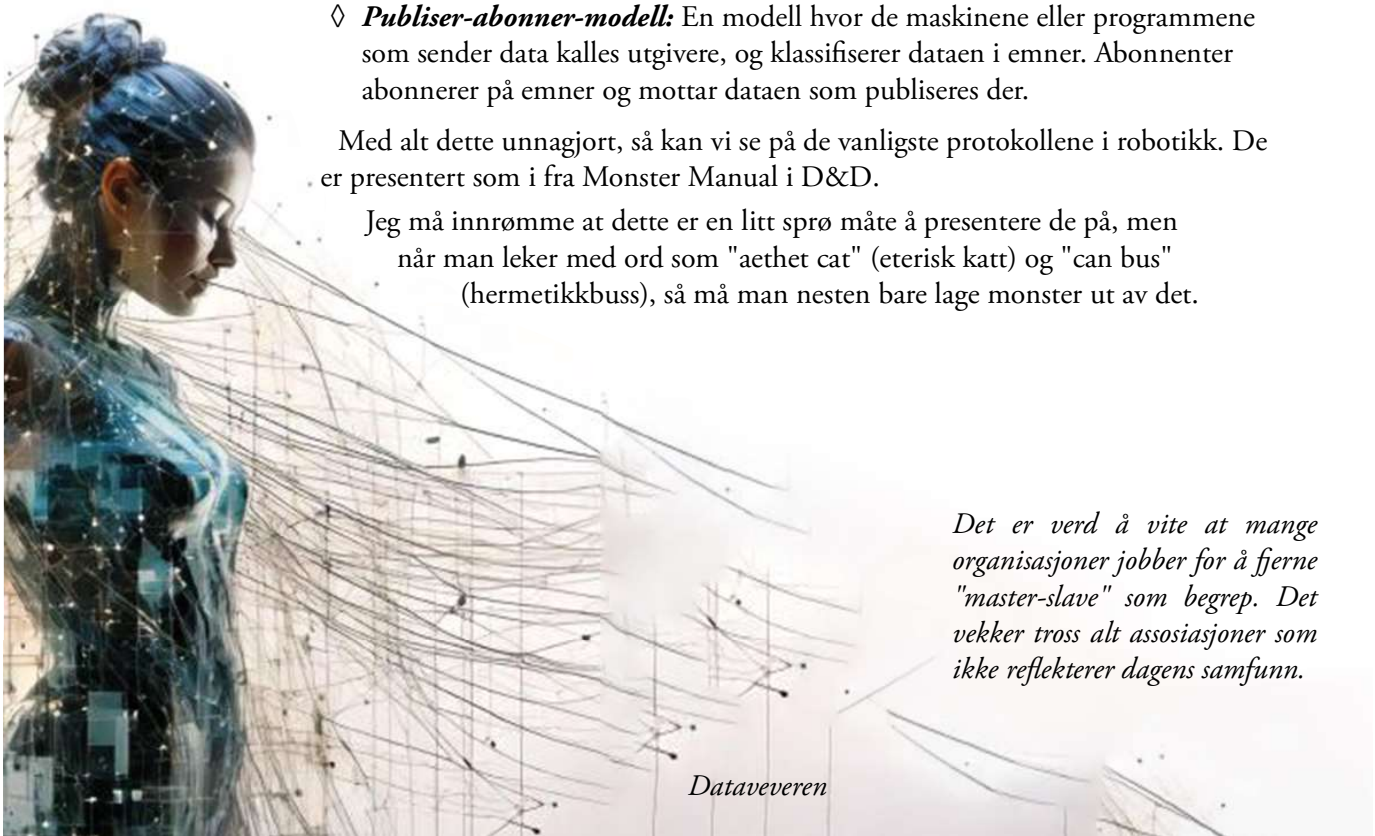
For at roboten skal kunne kommunisere med sensorer, samlebånd og sikkerhetssystem i robotcellen og med resten av fabrikken, må man ha en kommunikasjonsprotokoll i bunnen.

Det er mange å velge i hvor hver av har sine fordeler og bruksområder. Før vi ser på hver av disse, kan vi snakke om litt terminologier som er nyttig å vite om.

- ***Master/slave:*** Forholdet mellom to systemer, hvor den ene kontrollerer den andre og har fullmakt over kommunikasjonsflyten. Dette kan utvides til mange systemer, hvor en maskin er mastersystemet, og kontrollerer de andre som er slavesystemene.
- ***Daisy chain:*** En nettverkstopologi hvor systemene er koblet til hverandre i serie. Dette kan enten være en lineærtopologi, med toveiskommunikasjon mellom et system og den neste, eller ringtopologi, med enveiskommunikasjon mellom et system og neste, og den siste i serien er koblet til den første.
- ***Distribuert nettverk:*** Når data og databehandling spres på flere systemer. Nettverket totalt jobber for å oppnå et mål.
- ***Sanntid:*** Hvis kommunikasjonen er tidskritisk og et system krever en viss reaksjonstid og responstid. For eksempel kan roboter som driver kraftstyring kreve at eksterne sensorer sender målingen innen en viss tid (gjerne < 1ms) for at roboten skal kunne gjennomføre regulering. Sanntidssystemer blir diskutert senere i kapittelet.
- ***IoT:*** Internet of Things eller Tingenes Internett. Et løst begrep som omhandler maskiner som kobler seg på nettverk og deler data.
- ***QoS:*** Quality of Service eller Tjenestekvalitet. Noe som måler påliteligheten av kommunikasjonen. Latens (tidsforsinkelsen på meldinger), pakketap, gjennomstrømning og jitter (avvik fra frekvenstakt) er noen parametere som kan brukes til å måle kvaliteten.
- ***Publiser-abonner-modell:*** En modell hvor de maskinene eller programmene som sender data kalles utgivere, og klassifiserer dataen i emner. Abonnenter abonnerer på emner og mottar dataen som publiseres der.

Med alt dette unnagjort, så kan vi se på de vanligste protokollene i robotikk. De er presentert som i fra Monster Manual i D&D.

Jeg må innrømme at dette er en litt sprø måte å presentere de på, men når man leker med ord som "aethet cat" (eterisk katt) og "can bus" (hermetikkbuss), så må man nesten bare lage monster ut av det.



Det er verd å vite at mange organisasjoner jobber for å fjerne "master-slave" som begrep. Det vekker tross alt assosiasjoner som ikke reflekterer dagens samfunn.

DDS: Distributed Data Service

Dataveveren

Arkitekten av det digitale riket, som styrer over legioner av datastrømmer

Beskrivelse: Dataveveren fremstår som et stort, glitrende nettverk av sammenkoblede tråder, hver pulserende med informasjon. Formen skifter hele tiden ettersom den vever nye veier mellom systemer, mens utallige mindre enheter - kjent som *DataReaders* og *DataWriters* - myldrer gjennom de forseggjorte konstruksjonene. Disse mindre vesenene fungerer som forlengelser av Veverens vilje, og skaper komplekse mønstre av dataflyt som spenner seg over alle digitale domener.

Svakheter: Dataveverens kompleksitet kan være dens undergang. De forseggjorte QoS-mønstrene krever nøye konfigurasjon, og feiljusterte innstillinger kan forstyrre databanene. Selv om den er kraftig, krever den mer ressurser enn enklere protokoller, noe som gjør den mindre egnet for begrensede miljøer.

Lag: Sesjon-, presentasjon- og applikasjonlaget

Protokolloverhead: 4-12 byte per sample

Oppdagelsestid: <100 ms

Responstid: < 1 ms

Pålitelighet: 99%

Dynamisk oppdagelse: Kan automatisk oppdage og koble til andre DDS-enheter uten konfigurasjon

Quality of Service (QoS): Har 23 forskjellige QoS-mønstre for å sikre presis kommunikasjonsatferd

Tids- og romuavhengig: Kan opprettholde koblinger selv når enheter eksisterer i forskjellig tid og rom.

Selektiv kringkasting: Effektive multicast-mønstre som kun leverer data til riktige abonnenter.

Oppførsel: DDS bruker en publiserings-abonner-modell, der DataWriters publiserer data via emner, hvor DataReaders abonnerer på emner for å motta dataen. Hvert emne representerer en unik datatype med et bestemt navn og struktur. Ved hjelp av *Domain Participants* kan DDS effektivt knytte sammen DataWriters og DataReaders, slik at kun de som abonnerer og publiserer samme emne, snakker sammen. Dette gjør at DDS automatisk kobler sammen flere PC-er, PLS-er og roboter og at informasjonen flyter effektivt gjennom nettverket.

Når data distribueres, brukes en Global Data Space-abstraksjon, som lar applikasjoner lese og skrive data uavhengig av hvor de befinner seg i nettverket. DDS har også en *Real-Time Publish-Subscribe* (RTPS)-protokoll, som muliggjør at nye enheter oppdages og kobles til nettverket automatisk. Dette gjør at DDS kan tilpasse distribusjonsmønstret sitt uten å kreve sentraliserte servere eller spesielle nettverkstjenester.

I tillegg har DDS et *Quality of Service* (QoS)-system, som gjør det mulig å kontrollere og overvåke hvor god kvaliteten på nettverket er, og om data potensielt ikke mottas.

Habitat: DDS håndterer komplekse miljøer der sanntidskommunikasjon er avgjørende, og passer derfor til både industrielle automasjonssystemer, autonome kjøretøy og militære applikasjoner. DDS passer spesielt til oppdragskritiske systemer med små feilmarginer.

EtherCAT

Spektraljegeren

Et lynrask rovdyr som streifer gjennom den industrielle eteren på jakt etter data

Beskrivelse: Spektraljegeren framstår som en slank, spøkelsesaktig katt sammensatt av ren industriell eter. Dens form bølger av elektromagnetisk energi, og øynene gløder med den presise timingen til atomklokker. Mest karakteristisk er dens unike krystallinske ramme som flyter gjennom hele kroppen som en ryggrad, som lar den behandle data med utrolig hastighet mens den beveger seg.

Svakheter: Spektraljegerens jaktmønstre er sterkt avhengige av masternoden. Skulle masteren feile, stopper hele nettverket. I tillegg, mens den lineære prosesseringen er utrolig effektiv, kan den gjøre feilsøking kompleks da problemer i én slave kan påvirke hele kjeden.

Lagklassifisering: Datalinklaget

Syklustid: Ned til 31,25 µs

Jitter: < 1 µs

Maksimalt Antall Slaver: 65 535 enheter

Prosesseringstid: 110 ns/enhet

Maksimal Avstand: 100m/enhet

Distribuerte Klokker: Opprettholder sub-mikrosekund synkronisering over alle slaver

Prosessering i Farten: Leser og skriver data uten å stoppe, oppnår nesten 100% effektivitet

Til-/Frakobling under Drift: Tilpasser dynamisk seg slaver som kobles til og fra.

Prosessdataobjekter (PDO): Overfører sanntidsdata med ekstremt lav latens

Servicedataobjekter (SDO): Konfigurerer og diagnostiserer slaver under drift

Bevegelseskontroll: Koordinerer multiple akser med presis timing for robotbevegelser

Oppførsel: EtherCAT (Ethernet for Control Automation Technology) er en sanntids industriell Ethernet-protokoll utviklet av Beckhoff Automation. Den opererer etter et *master/slave*-prinsipp hvor én masterenhet kontrollerer kommunikasjonen med flere slaveenheter.

Protokollens særtrekk er dens unike prosesseringsmetode: Enheter er konfigurert i en *daisy-chain*, hvor én enkelt Ethernet-ramme passerer gjennom alle enheter i sekvens, og hver slave prosesserer data "i farten" mens rammen forflytter seg gjennom nettverket.

Habitat: EtherCAT implementeres primært i industrielle automatiseringsmiljøer som krever presis posisjonsregulering og synkronisering. Vanlige bruksområder inkluderer robotikk, CNC-maskiner, pakkesystemer og halvlederproduksjon.

Protokollen er spesielt egnet for oppgaver som krever deterministisk kommunikasjon med syklustider under 100 mikrosekunder og jitter mindre enn 1 mikrosekund.

MQTT

Meldingskolibriene

En sverm av små fuglelignende budbringere som beveger seg effektivt mellom enheter.

Beskrivelse: Meldingskolibriene fremstår som en sverm av iriserende, mekaniske kolibrier, hver på størrelse med en tommel. De nålelignende nebbet er perfekt utformet for presis injeksjon av meldinger til deres destinasjoner. De metalliske fjærene skinner med ulik intensitet basert på deres tjenestekvalitetsnivå (*Quality of Service*): kobberglans for QoS 0, sølvskimmer for QoS 1 og gyllen stråleglans for QoS 2. I hjertet av deres territorium holder Broker-dronningen til, en større og mer utsmykket kolibri som koordinerer svermens mønstre for meldingsoverføring.

Svakheter: Kolibriene er helt avhengige av Broker-dronningen for koordinasjon – hvis hun faller, spres svermen i kaos. Deres lille størrelse, som er perfekt for effektivitet, begrenser deres kapasitet for meldingsoverføring. De må opprettholde en konstant forbindelse til etablerte flyvebaner (TCP-tilkoblinger), noe som gjør dem sårbare når luftstrømmene (nettverksforholdene) blir ustabile.

Lagklassifisering: Fra transport- til applikasjonslaget

Protokoll: TCP/IP-basert

Meldingsstørrelse: Minimal overhead (2-byte fast header)

QoS-nivåer: 0 (Maksimalt én gang), 1 (Minst én gang), 2 (Akkurat én gang)

Sikkerhet: TLS/SSL-støtte

Arkitektur: Publiser/Abonner med sentral broker

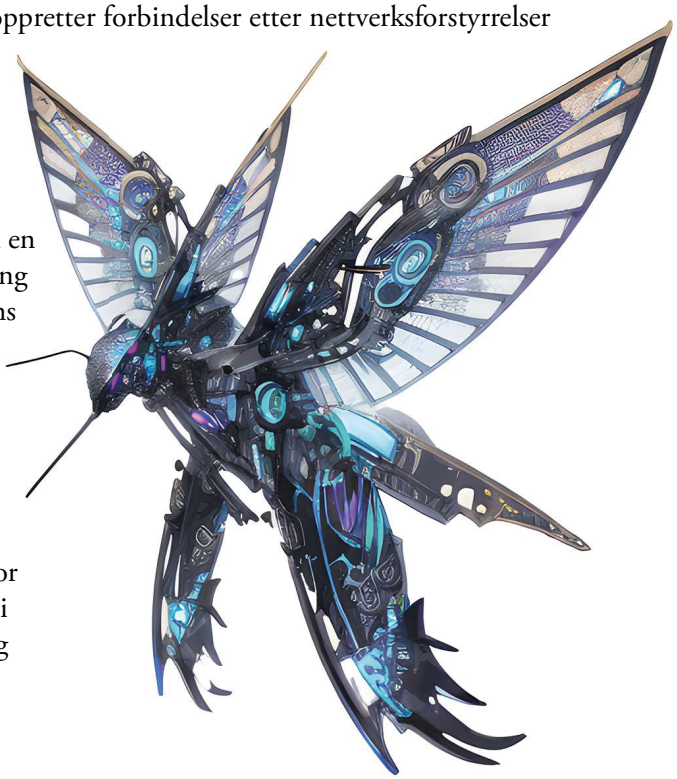
Effektiv flukt: Minimalt energiforbruk ved meldingsoverlevering

Siste sang: Etterlater en siste melding ved uventet frakobling

Meldingslagring: Lagrer viktige meldinger i bikube-lignende strukturer

Emnenavigering: Navigerer presist gjennom komplekse, hierarkiske emnestrukturer

Svermresiliens: Reformer seg raskt og gjenoppretter forbindelser etter nettverksforstyrrelser



Oppførsel: MQTT (Message Queuing Telemetry Transport) er en lettvekts publiser-abonner-meldingsprotokoll designet for begrensede enheter og nettverk med lav båndbredde og høy ventetid. Den opererer via en sentral *broker* som håndterer meldingsoverføring mellom publisister og abonnenter. Protokollens enkelhet og effektivitet gjør den ideell til IoT-applikasjoner og ressursbegrensede miljøer.

Habitat: MQTT opererer primært i IoT-økosystemer og ressursbegrensede miljøer. Den trives i situasjoner hvor båndbredde er begrenset, og strømforbruk må minimeres, fra smarte sensorer i produksjon til systemer for fjernovervåking. Protokollen er særlig utbredt i miljøer som krever effektiv meldingsoverføring mellom mange enheter.