

# 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

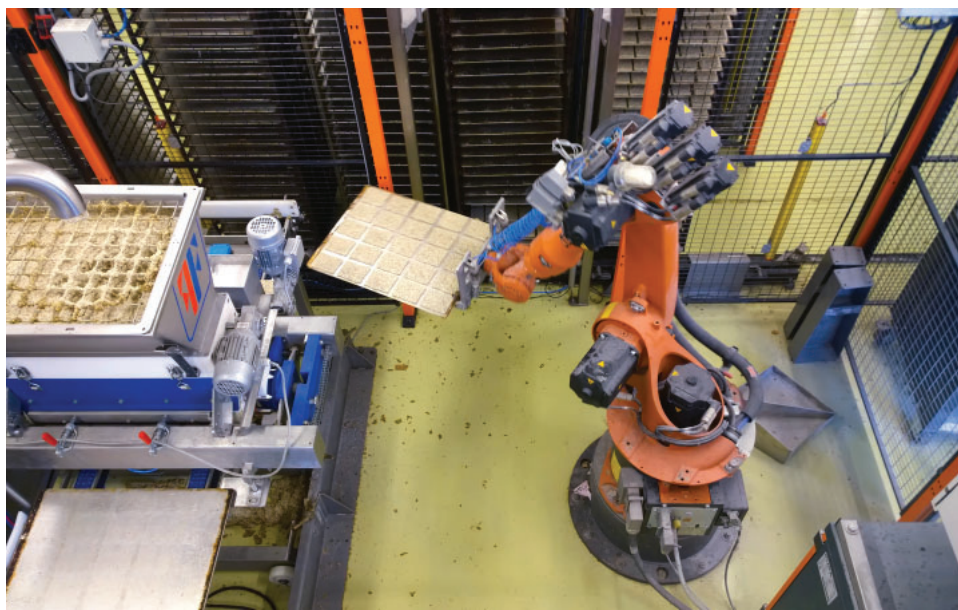
Dette er Viddal Automation. Det er en bedrift som leverer robotisering av prosesser for hovedsakelig små og mellomstore bedrifter. De jobber med stort sett alt og ingen utfordring er for stor.

Viddal Automation holder til på Håhjem i Skodje kommune (uttaelse 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 raskere.

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.

*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 samlebandet*

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

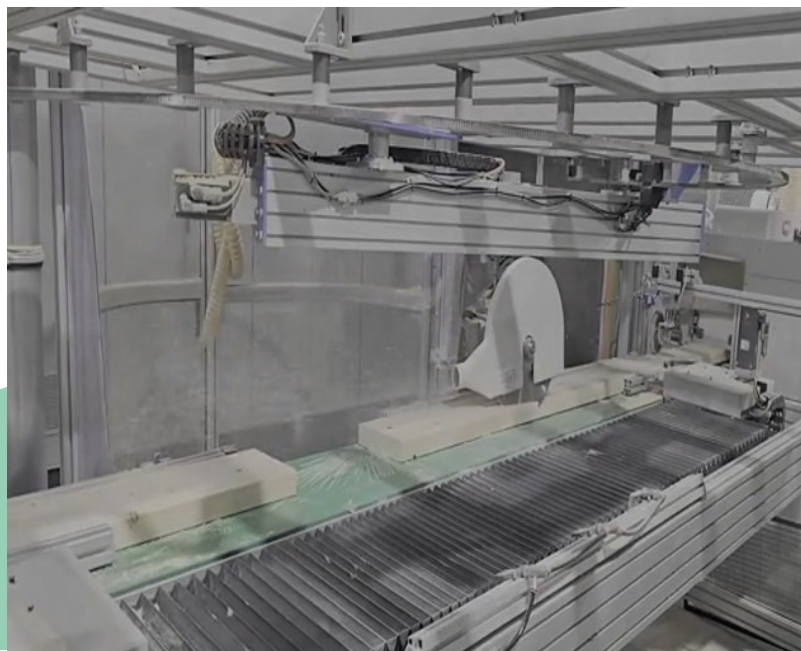


## Oppgave

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?



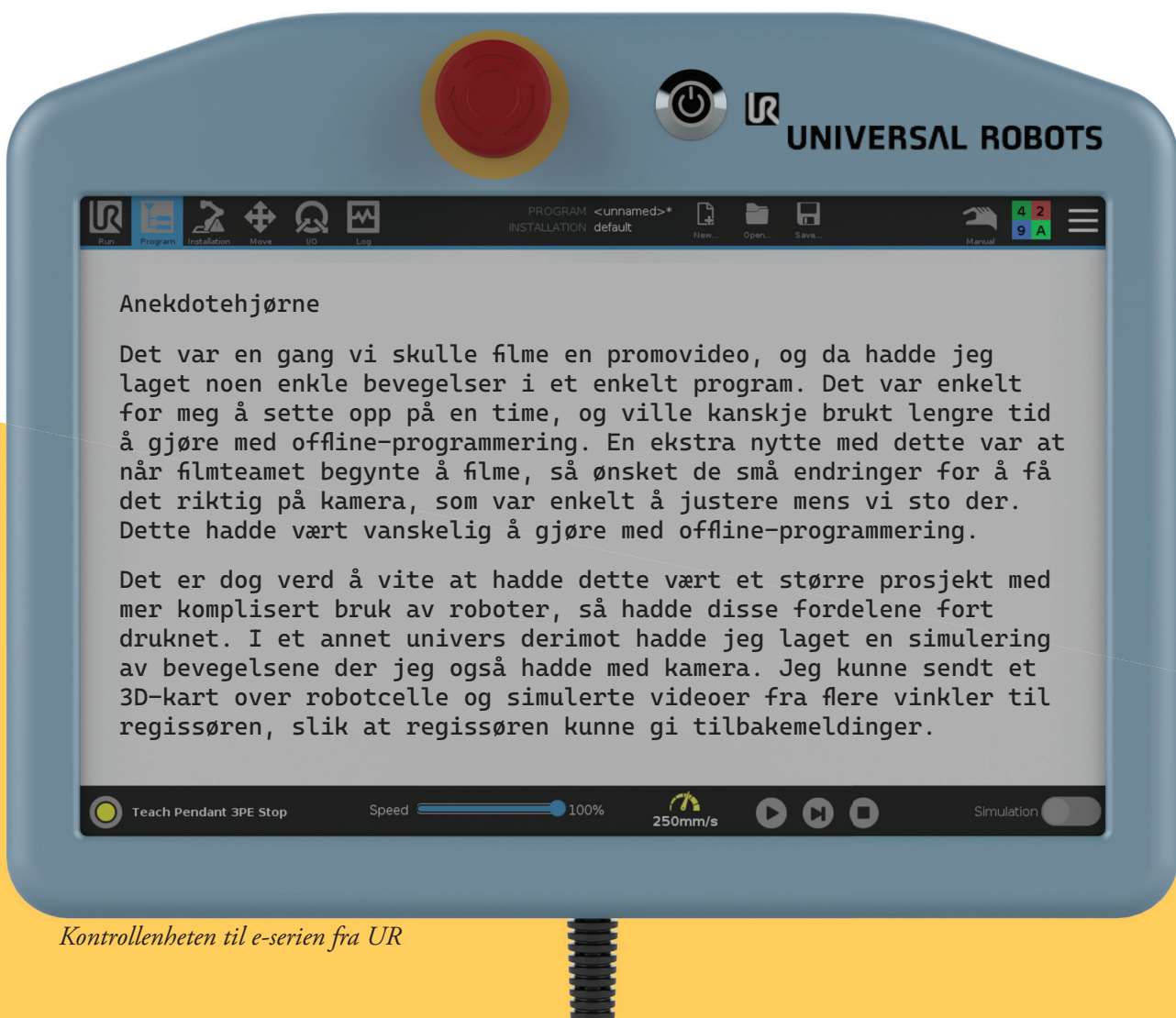
# OFFLINE- VS ONLINE-PROGRAMMERING

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.

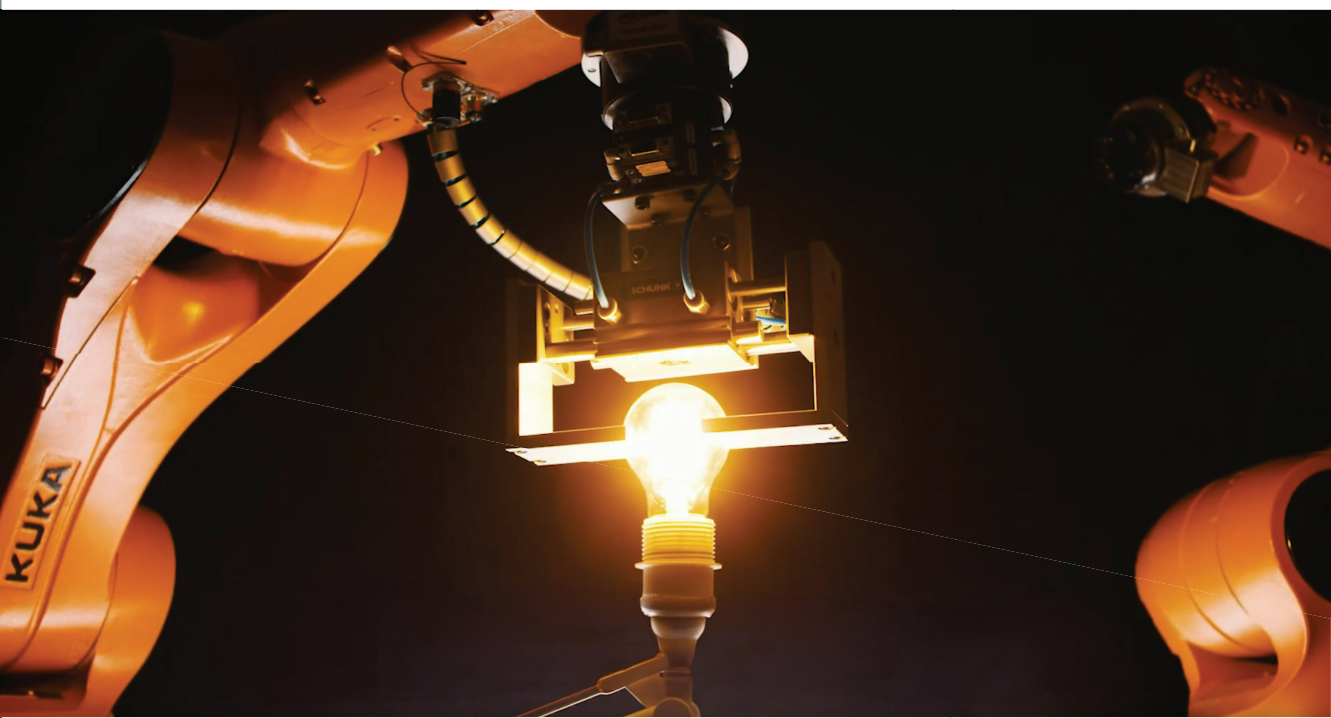
Den største ulempen med online-programmering er sikkerhet. Du må stå nært, og av og til inni robotcellen for å programmere, og man får heller ingen tilbakemelding om hva roboten planlegger å gjøre når den beveger seg fra A til B. Det er sjeldent at roboten får problemer med selve bevegelse, men det største 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 simulatorprogram ikke kan generere, fordi de som lagde simulatorprogrammet ikke visste eller tenkte over at den kommandoen eksisterte. Mange simulatorprogram 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  
skruer i en lyspære.*



## Offline-programmering

Når man skal programmere offline, så trenger man ikke en fysisk robot. Man er bokstavelig 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.

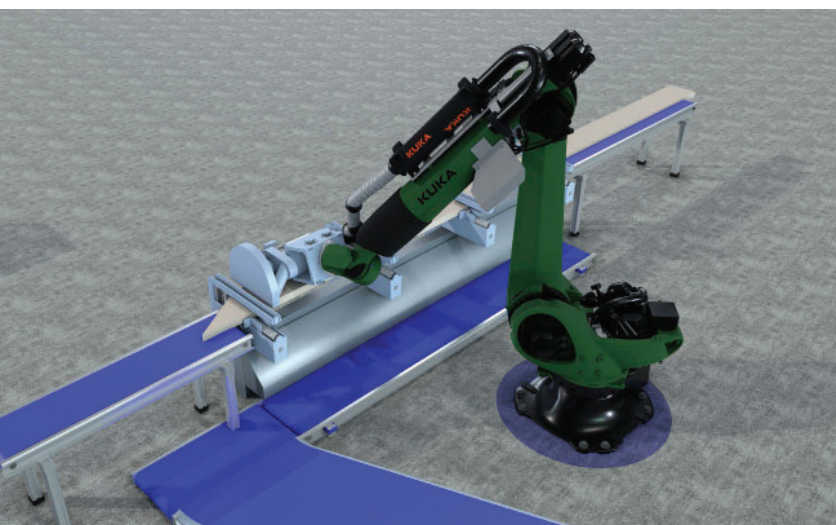
Selv om G-kode tidligere ble programmert manuelt, så er det mest vanlig nå å generere den ut ifra det du planlegger å maskinere. Alle store 3D-modelleringsprogrammer har en mulighet til å eksportere modellen til G-kode, eller til en filtype som et annet program kan lage G-kode utav. Uansett hvordan du 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 simuleringsverktøy 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 hardkoder 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 såkalt *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. Regulerings-teknikk er en form for dynamisk programmering.





♥ 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 et fantastisk verktøy i programmering.

Når man innser at programmering ikke er å *skrive kode*, men å *redigere kode*, ser man fyllden 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.





# 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  $\mu$ s

**Jitter:** < 1  $\mu$ s

**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:** Publisjer/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 publisjer-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.

