Det matematisk-naturvitenskapelige fakultet

Institutt for Informatikk

IN1080 Mekatronikk

Mekanikk, programmering av innebygde systemer

Yngve Hafting
Universitetslektor
Robotikk og intelligente systemer

2023-03-29

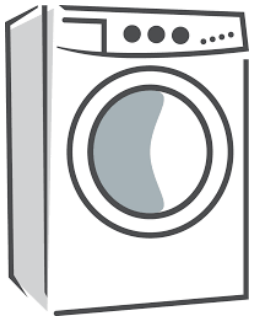
UNIVERSITETET
I OSLO

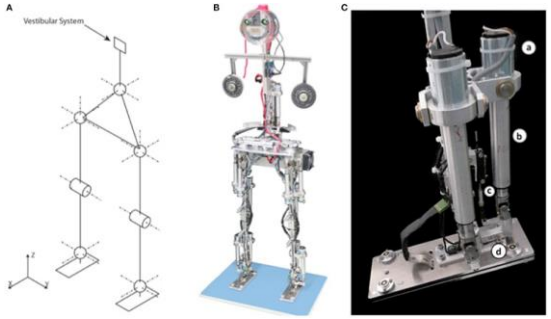
Beskjeder

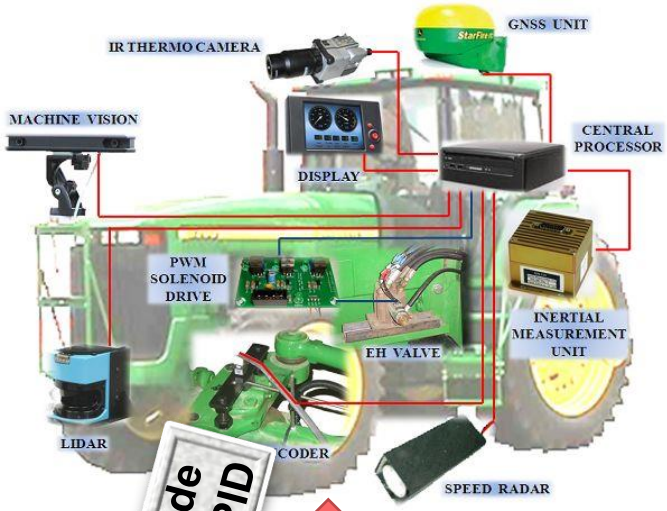
- Discourse plattform lite brukt
  - https://in1080-discourse.uio.no
  - Har du noe du ønsker for kurset?
    - Egen fane for tilbakemeldinger
  - Helt greit å ta spørsmål i andre fora (regneøvelse, grupper, forelesning)

Hvor står vi – hvor går vi…

Formål: Å lage og programmere mekatroniske systemer







AD konvertering

Sensorer

Instrumenterings-forsterkere

Halvledere

Opamper

Grunnleggende elektronikk …

Pulsbreddemodulering

Drivkretser

Elektromotorer…

Grunnleggende mekanikk

Grunnleggende kontrollteori, PID

Kommunikasjons-protokoller (busser)

Programmering av innebygde systemer

Arkitekturer for innebygde systemer

Grunnleggende Mekanikk

For å kunne bygge og programmere mekatroniske systemer trenger vi

- Grunnleggende lover for
  - Bevegelse
    - Bevegelsesligninger
    - Bevegelsesmengde
    - Rotasjon
  - Kraft
    - Newtons lover
    - Kraftmoment (Dreiemoment, Torque)
  - Energi
    - Mekanisk
    - *Elektrisk*

Skal vi gå lengre trenger vi bredere systemforståelse

- 2. ordens systemer/ svingesystemer, osv
- kybernetikk/reguleringsteknikk *ut over IN1080 -> IN3140-Introduksjon til Robotikk*

Grunnleggende Mekanikk

Innhold

- Repetisjon av relevant FYS 1-2 innhold:
  - Rettlinjet bevegelse
  - Bevegelsesmengde, kraft
  - Newtons lover
  - Tyngdekraft, potensiell energi
  - Arbeid og effekt
  - Sirkelbevegelse
- Nye Fysikktema:
  - Kraftmoment (Dreiemoment/ Torque)
- MERK: **Still spørsmål dersom noe er uklart!**
  - Mekanikken er ikke dekket i læreboka
  - Tidligere læreverk (FYS 1,2) og åpne nettressurser kan benyttes for selvstudium.
    - *Tips* for selvstudium:
      - Start med tidligere lærebøker, dernest wikipedia
      - Ytterligere forklaringer: *søk med utgangspunkt i ord fra wikipedia*
      - *Spør gjerne, men ikke stol på ChatGPT o.l..*

Mer utfyllende:
https://en.wikipedia.org/wiki/Linear_motion
https://en.wikipedia.org/wiki/Force
https://en.wikipedia.org/wiki/Newton%27s_laws_of_motion
https://en.wikipedia.org/wiki/Work_(physics)
https://en.wikipedia.org/wiki/Circular_motion
https://en.wikipedia.org/wiki/Potential_energy
https://en.wikipedia.org/wiki/Torque

Grunnleggende mekanikk

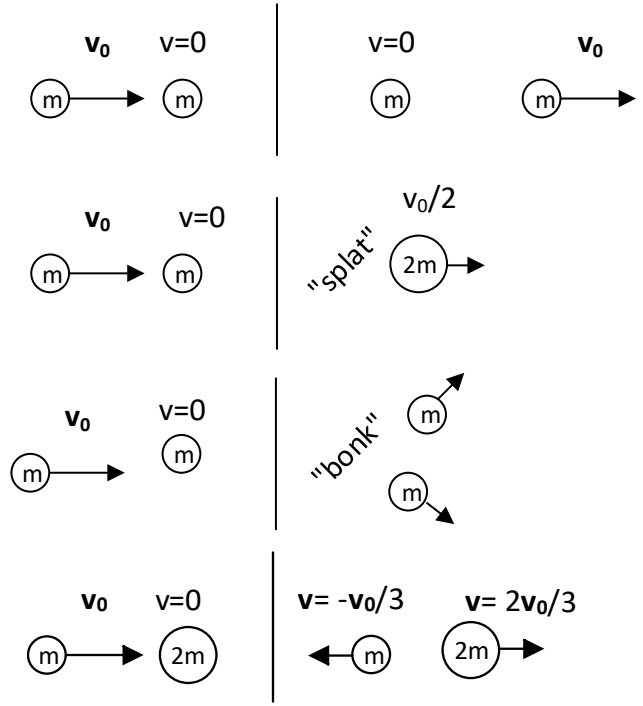
Bevegelsesligninger

- *Generelt*
- $\boldsymbol{s}(t) = \int \boldsymbol{v} \cdot dt$
- $\boldsymbol{v}(t) = \frac{d\boldsymbol{s}}{dt} \quad (alt. notasjon: \boldsymbol{v} = \dot{\boldsymbol{s}})$
- $\boldsymbol{a}(t) = \frac{d\boldsymbol{v}}{dt} \quad (\boldsymbol{a} = \ddot{\boldsymbol{s}})$
- *Konstant hastighet:*
  - $\boldsymbol{s} = \boldsymbol{s_0} + \boldsymbol{v} \cdot t$
- *Konstant akselerasjon:*
  - $\boldsymbol{a} = \boldsymbol{v}/t$
  - $\boldsymbol{s} = \boldsymbol{s_0} + \boldsymbol{v_0}t + \boldsymbol{a}t/2$

- Størrelser
  - Strekning (s) : m
  - Hastighet (v) : m/s
  - Akselerasjon (a) : m/s^2
- Merk:
  - Vektorer angis ofte med **fet** skrift
  - *Prøv å holde styr på når vilkårene for å bruke skalare størrelser er oppfylt*
    - Ie bevegelse i én retning, etc.

Grunnleggende Mekanikk

Bevegelsesmengde, kraft



- Bevegelsesmengde (*linear* momentum)
  - $\boldsymbol{p} = m \cdot \boldsymbol{v}$ (enkelt partikkel)
  - $\boldsymbol{p} = \sum \boldsymbol{m_i v_i}$ (mange-partikkel)
  - I et lukket system er bevegelsesmengden bevart
    - (Lukket system = ingen ytre krefter)

- Kraft
  - Kraft er det som endrer bevegelsen til et objekt
  - Et legemes endring i bevegelsesmengde er proporsjonal med kraften det utsettes for
    - $\boldsymbol{F} = \frac{d\boldsymbol{p}}{dt} = \frac{m d\boldsymbol{v}}{dt} = m\boldsymbol{a}$ (også kjent som N2. lov).

$$\text{Enhet}: \text{Newton} \left(1\text{N} = \frac{1\ kgm}{s^2}\right)$$

Grunnleggende Mekanikk

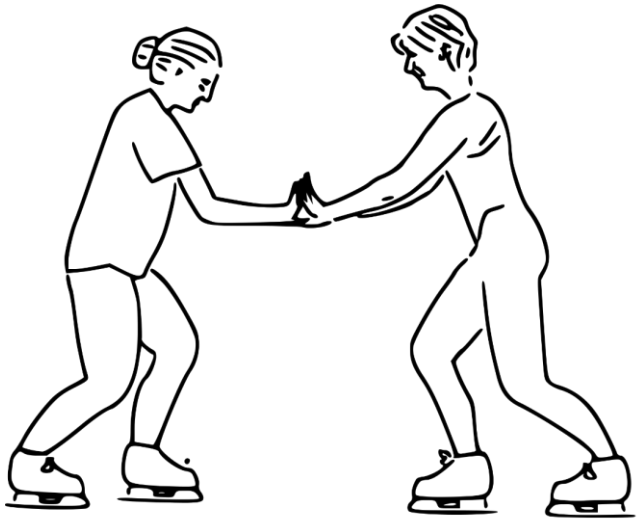
Newtons lover

N1.lov:

- Et legeme som ikke er utsatt for krefter vil stå stille eller bevege seg med konstant fart i rett linje

N2.lov

- Endringen av hastighet til et legeme i bevegelse er proporsjonal med og i samme retning som summen av krefter som virker på legemet.
- $\sum \boldsymbol{F} = \boldsymbol{m} \cdot \boldsymbol{dv}/\boldsymbol{dt}$
- $\boldsymbol{F} = \boldsymbol{m} \cdot \boldsymbol{a}$



N3.lov

- Hvis et legeme påvirker et annet legeme med en kraft F, så vil det andre legemet påvirke det første legemet med en kraft F’ som er like stor som F, men motsatt rettet.
- «Kraft er lik motkraft»

Grunnleggende Mekanikk

Tyngdekraft og potensiell energi

- Tyngdekraft ($\mathbf{G}$)
  - $\mathbf{G} = m\mathbf{g}$, (*m er masse oppgitt i antall kg*)
- Tyngdeakselerasjon ($\mathbf{g}$)
  - $g \approx 9{,}81 \frac{m}{s^2}$

IN1080: *Trenger ikke utlede g...*
Vi holder oss på jordoverflaten hvis ikke annet er oppgitt

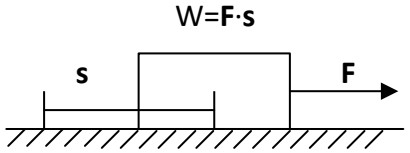
- Potensiell energi
  - $E_p = mgh$, (*h er høyde i meter over valgt nullpunkt*)

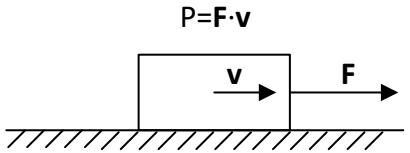
- Enhet for Energi er
  - Joule (1J = $1\ \mathrm{Nm}$ = $1\mathrm{kgm^2/s^2}$ = 1Ws)
  - (1kWh = 1000W*3600s = 3,6 MJ)

Grunnleggende Mekanikk

Arbeid og effekt

- *Mekanisk Arbeid*
  - $W = \boldsymbol{F} \cdot \boldsymbol{s}$ (Generelt $\mathrm{W} = \int \mathbf{F}\mathrm{d}\boldsymbol{s}$)
  - $W = P \cdot t$ $(konstant\ effekt)$
- *Effekt*
  - $P = \frac{dW}{dt}$ $\quad (\frac{d}{dt}\int \boldsymbol{F}d\boldsymbol{s} = \boldsymbol{F}\frac{d\boldsymbol{s}}{dt})$
  - $P = \boldsymbol{F} \cdot \boldsymbol{v}$ *(I øyeblikket)*
- Elektrisk
  - $P = V \cdot I$
  - $P = E/t$

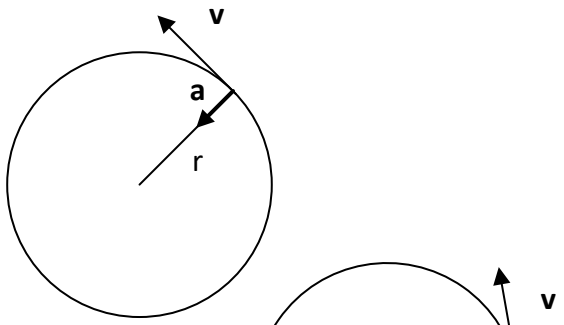


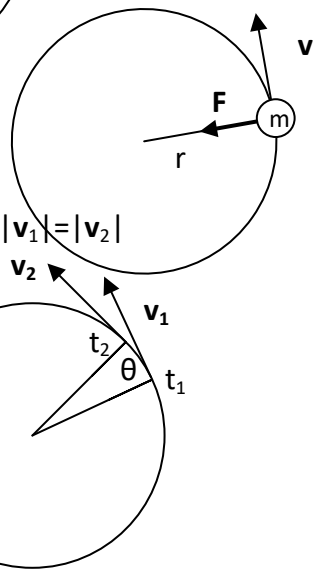


- Størrelser
  - Kraft (F) : N, Newton
  - Arbeid (W) : J, Joule
  - Effekt (P) : W, Watt

Grunnleggende Mekanikk

Sirkelbevegelse





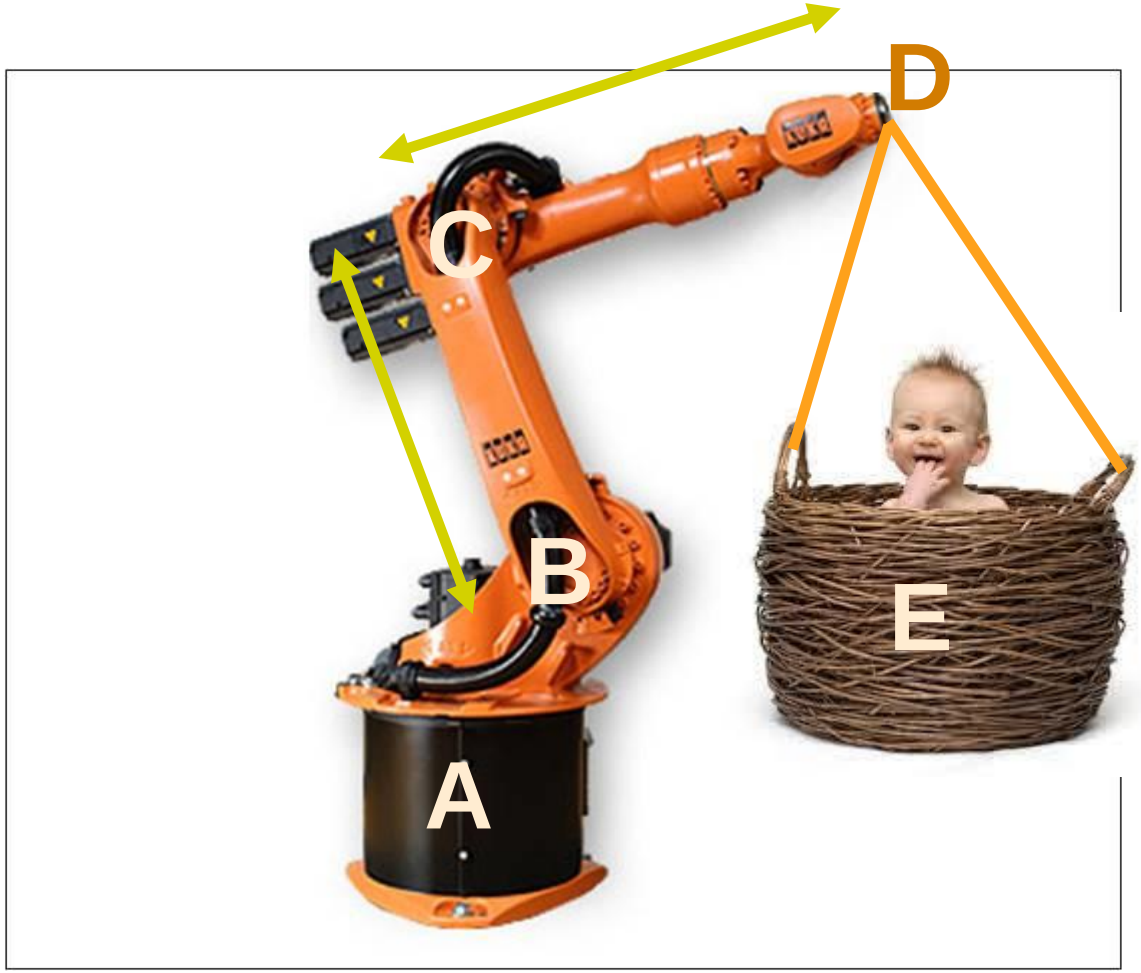
- *Konstant vinkelhastighet (ω):*
  - $a = \frac{v^2}{r}, F = \frac{mv^2}{r}$
- $\omega = \frac{d\theta}{dt}$
- $\omega = \frac{v}{r}$
- $\alpha = \frac{\omega}{t}$, α er vinkelakselerasjonen
- $\omega = \frac{2\pi}{T}$, *T er tiden for en hel runde*
- $f = \frac{\omega}{2\pi}$, *f er frekvensen*

- Størrelser
  - Vinkel (θ): radianer (rad)
  - Vinkelhastighet (ω) : rad/s
  - Vinkelakselerasjon (α) : rad/s^2
- Merk radianer er i prinsippet dimensjonsløs
- Omregning RPM til rad/s
- En runde $= 2\pi$, $ett\ minutt = 60s$
  - $1\text{RPM} = 2\pi/60s$
  - $= \frac{\pi}{30} s^{-1} \approx 0{,}105 rad/s$

Regneeksempel 1 (Vanskelig)

Hva er hastigheten (v) til kurven?

- Premisser:
  - konstant hastighet, ω_A= 60 RPM
    - Ingen gynging
  - D står i forlengelsen av aksen til A (sentrert, rett opp)
  - l_{DE} = 1m

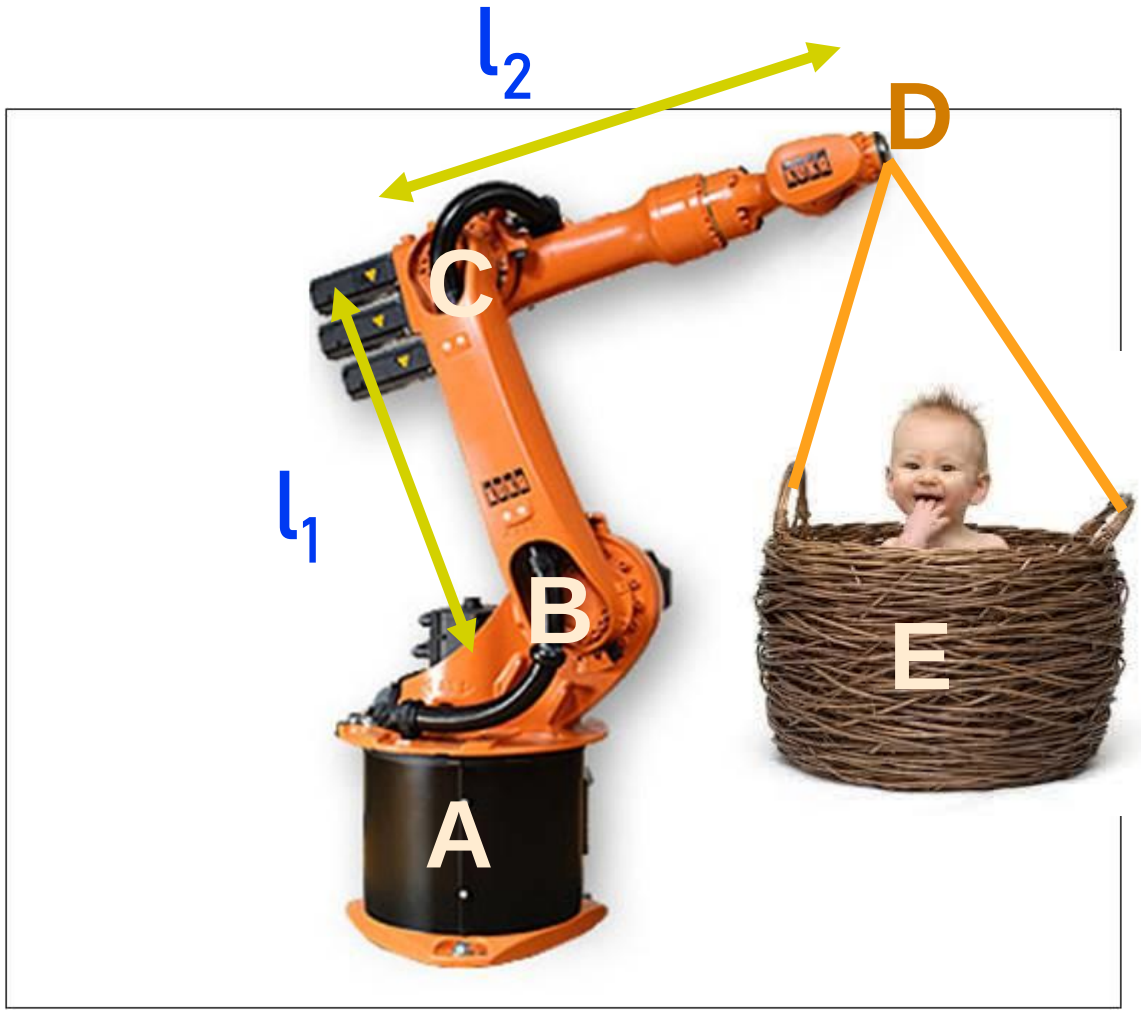


Regneeksempel 2

Hva er minste kraftmoment motorene i B og C kan ha for å kunne løfte kurven (i alle stillinger, fra stillstand)?

- La
  - $l_1 = l_2 = 1m$
  - Arm 2 har masse 1kg, jevnt fordelt
  - Arm 1 har masse 1kg, jevnt fordelt
  - $m_E = 10kg$

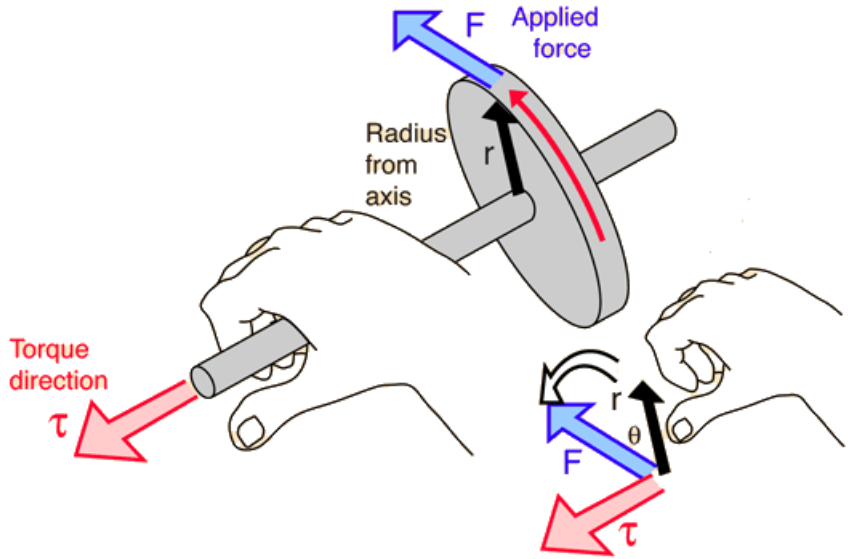
  Hint:
  - I hvilken retning står roboten når det er tyngst?

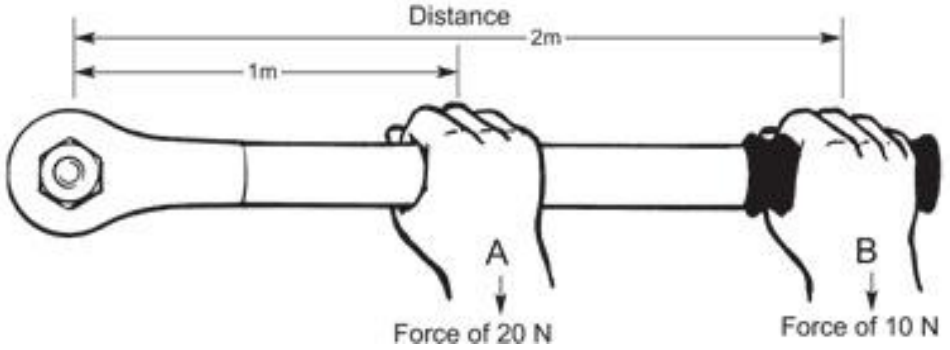


Grunnleggende Mekanikk

Kraftmoment (dreiemoment, torque)

- Dreiemoment (torque) er en krafts evne til å forandre et legemes rotasjon.
- Dreiemoment $\boldsymbol{\tau}$ (tau) beregnes med kryssproduktet av avstanden (r) til legemets rotasjonspunkt og kraftvektoren (F).
  - $\boldsymbol{\tau} = \boldsymbol{r} \times \boldsymbol{F}$ (*rekkefølgen avgjør retningen*)
- Retningen til kryssproduktet er gitt ved høyrehåndsregelen, dvs. 90 grader på både radius og kraftvektor. ($\boldsymbol{\tau}$-tommel, $\boldsymbol{r}$-håndflate, $\boldsymbol{F}$-fingre).
- Størrelsen på kraftmomentet blir $\tau = r \cdot Fsin(\theta)$ eller $\tau = r \cdot F$
  - I svært mange praktiske sammenhenger så er F normalt på r, og da får vi $\tau = r \cdot F$ ("kraft · arm")
- Eks: Både A og B på figuren gir et kraftmoment på 20Nm (*retning inn i skjermen*) på pipenøkkelen, i sum: 40Nm



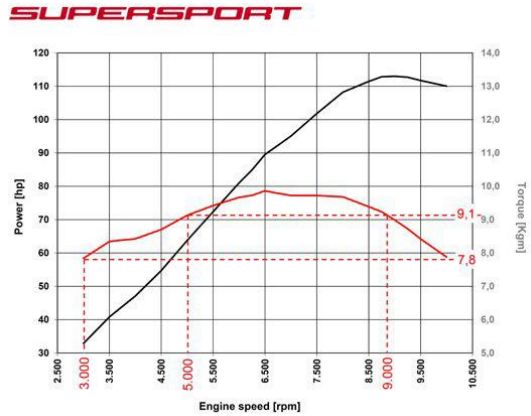


Grunnleggende Mekanikk

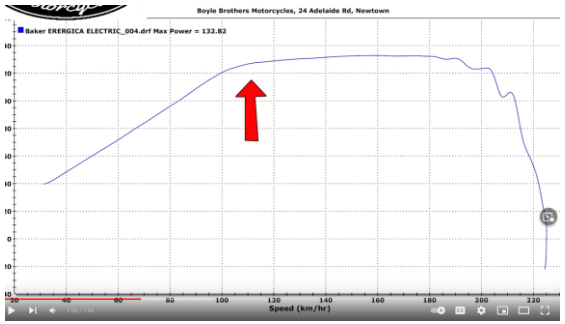
Kraftmoment og motorer, eksempel på tall.

- Når vi karakteriserer motorer, får vi gjerne oppgitt
  - Dreiemoment (Nm «Newtonmeter», US: ft. lb. (foot-pound) 1Nm ≈ 0,74 ft.lb.)
  - (Maksimal) effekt («Watt» W, «hestekrefter» hk evt. hp 1kW ≈ 1,36 hp).

  - Forbrenningsmotor:
    - Dreiemoment øker opp til maksimalt turtall
      - Begrenses av inntak av drivstoff+luft, kompresjon, etc.
  - Elmotor
    - Konstant dreiemoment til vi har maks effekt
      - Begrenses av strøm, spenning, selvinduksjon
  - Eks:
    - Dronemotor, T-motor Antigravity MN4004
      - Power: 216W
      - Torque 0,27 Nm
    - Tung Motorsykkel Bensin
      - 110 hp – 81 kW,
      - 93 Nm - 69 lb-f
    - Tung Motorsykkel Elektrisk
      - 80kW - 109 hp
      - 200 Nm - 147 ft.lb.



Ducati Supersport 950 dyno run,
Power (Black) Torque (Red)



Energica Ego Dyno run (Power curve)

Programmering av innebygde systemer

Innhold

- Hva skiller innebygde (mekatronikk-) systemer fra annen programmering?
  - Periferienheter
  - Sanntidskrav
  - Operativsystem?

- Hva har dette å si for hvordan vi programmerer?
- = *dagens tema*

Programmering av innebygde systemer

Innhold

- Hendelsesdrevet programmering (Event driven programming)
    - Hendelser/ Events
        - Blokkerende kode
        - Ikke blokkerende kode
            - Bruk av globale variabler
    - Service rutiner
    - Avbrudd, avbruddsrutiner (Interrupts, ISR)
    - *Tilstandsmaskiner*
    - *Quadrature enkoder (eksempel på tilstandsmaskin)*

Programmering av innebygde systemer

Hendelsesstyrt programmering «Event driven programming»

- **Hendelse** (*event*)
  - *Noe som skjer og skal føre til aksjon*
    - Museklikk, bryter som er skrudd på, AD-verdier osv.
- En **hendelsessjekk** (*event checker*) sjekker om en bestemt hendelse har skjedd.
- **Hendelseshåndtereren-** (*event handler*)
  - Operativsystem, Vindustjener, «Arduino-loop» / while(1) {…}, …
  - kjører i en evig løkke som starter hver rutine for å sjekke hendelser…
  - …starter en **tjeneste** (*service*) som utfører det som skal skje ved en hendelse

- *Hva kan vi gjøre for å hindre at én hendelse starter samme tjeneste mange ganger?*
  - Eksempel: neste side:

Programmering av innebygde systemer

Eksempel på Hendelsesstyrt programmering «Event driven programming»

- **blokkerende** kode:
  - Forhindrer at noe annet skjer mens vi venter på noe.
  - Eks Arduino:
    - `delay(ms)`
    - `while(true){}`
    - ...
- **Ikke-blokkerende** kode
  - Eks: Vi sjekker både trykk og slipp av knapp.

```
void knappehendelse(){
  if(knappetrykk == true){
    knapperutine();
    while(knappetrykk == true){// gjør ingenting}
  }
}

boolean knappehendelse(){
  static boolean forrige_status = false;
  if(knappetrykk != forrigestatus){
      knapperutine();
    return true;
  }
}
```

Programmering av innebygde systemer

Strukturere kode

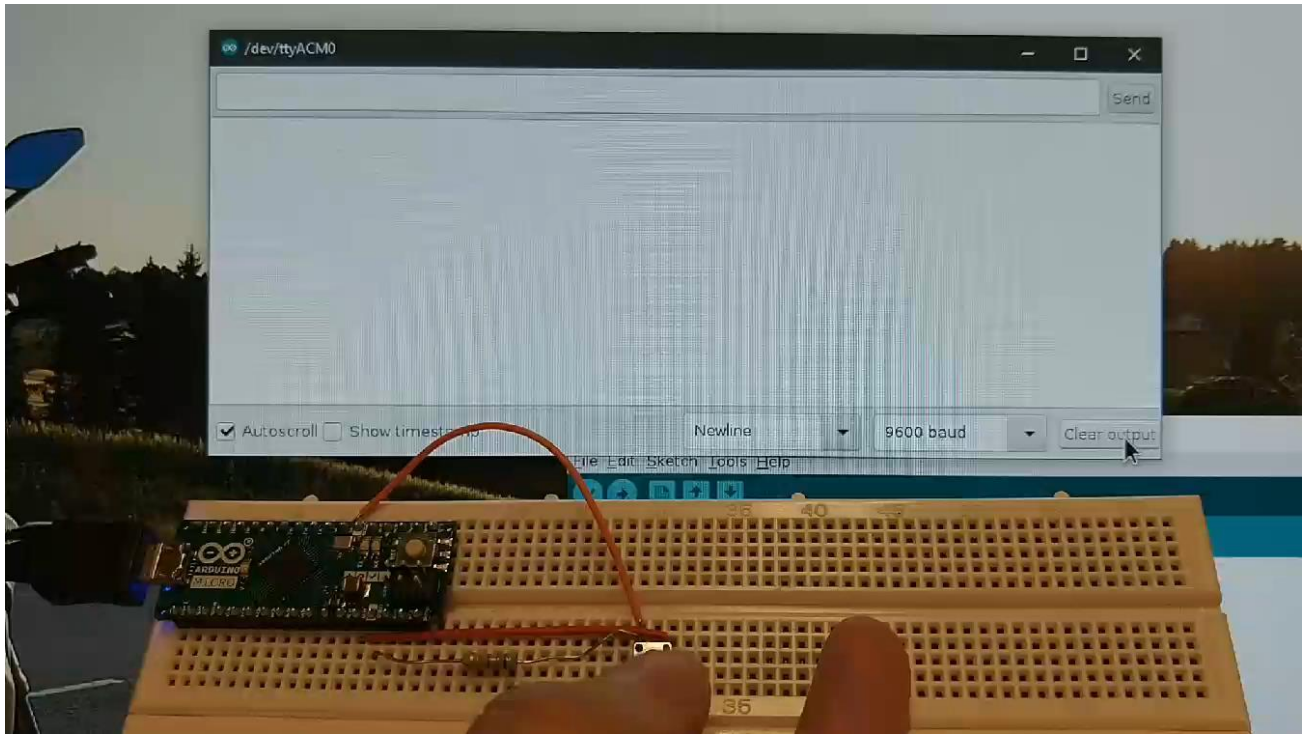
- Unngå duplisering av kode
  - Bruk funksjoner/ prosedyrer
- Bruk lokale variable såfremt mulig.
  - *Globale variable benyttes kun når verdien skal benyttes av flere rutiner.*
  - «**statiske variabler**»
    - holder verdi mellom funksjonskall
    - *fungerer som global variabel, kan bare leses lokalt*
- Skille
  - Oppsett (*Arduino «setup»*)
  - Hendelsessjekker (event checkers)
  - Tjenesterutiner (services)
  - Hendelseshåndterer (*Arduino: «loop»*)

```
const int buttonPin = 2;     // pushbutton pin number
const int ledPin =  13;      // LED pin number (Arduino standard)

void setup() {
  pinMode(ledPin, OUTPUT);    // LED pin is output:
  pinMode(buttonPin, INPUT);  // pushbutton pin is input:
  Serial.begin(9600);         // initialize serial communication
}

// event checker, non-blocking
boolean buttonEvent(){
  static int lastState = 0;
  int buttonValue = digitalRead(buttonPin);
  if (buttonValue != lastState){
    lastState = buttonValue;
    return true;
  }
  return false;
}

// service routine can do anything.., non-blocking
void buttonService(){
  if (digitalRead(buttonPin) == HIGH){
     digitalWrite(ledPin, HIGH);
     Serial.println("on");
  } else {
    digitalWrite(ledPin, LOW);
    Serial.println("off");
  }
}

// Run event-checkers and service accordingly
void loop() {
if ( buttonEvent() ) {
    buttonService();
  }
  // Insert other event checks as needed...
}
```

Vi har en hendelse kun når tilstanden endres
Verdien den endres til er ikke så viktig,
men vi må huske verdi fra gang til gang

Programmering av innebygde systemer

Eksempel



Merk: I videoen kan vi se at vi får *prell* når jeg slipper knappen 2. gang. (Gir off-on-off i sekvens)

```
const int buttonPin = 2;     // pushbutton pin number
const int ledPin =  13;      // LED pin number (Arduino standard)

void setup() {
  pinMode(ledPin, OUTPUT);    // LED pin is output:
  pinMode(buttonPin, INPUT);  // pushbutton pin is input:
  Serial.begin(9600);         // initialize serial communication
}

// event checker, non-blocking
boolean buttonEvent(){
  static int lastState = 0;
  int buttonValue = digitalRead(buttonPin);
  if (buttonValue != lastState){
    lastState = buttonValue;
    return true;
  }
  return false;
}

// service routine can do anything.., non-blocking
void buttonService(){
  if (digitalRead(buttonPin) == HIGH){
     digitalWrite(ledPin, HIGH);
     Serial.println("on");
  } else {
    digitalWrite(ledPin, LOW);
    Serial.println("off");
  }
}

// event handler- run event-checkers and service accordingly
void loop() {
if ( buttonEvent() ) {
    buttonService();
  }
  // Insert other event checks as needed...
}
```

Vi har en hendelse kun når tilstanden endres
Verdien den endres til er ikke så viktig,
men vi må huske verdi fra gang til gang

Programmering av innebygde systemer

Interrupter «Avbruddshåndtering»

- Mikrokontrollere og prosessorer *kan* være laget for avbruddhåndtering (interrupter)
- To typer avbrudd
  - Maskinvarebaserte avbrudd (Hardware interrupts)
  - Programvareavbrudd (Software interrupts)
- Programvareavbrudd
  - håndteres gjerne av OS
  - Er i prinsippet et hendelseshåndteringssystem
    - event handler+ event checker + service routines
    - Hendelsessjekkene *polles* jevnlig
  - Kan ha varierende bruk av prioritet eller køordninger
- Maskinvareavbrudd
  - Trigges gjerne av fysiske hendelser
    - Software *kan* også trigge maskinvareavbrudd
  - Håndteres i prioritert rekkefølge
    - *Prioriterte avbrudd håndteres før andre avbrudd, uavhengig av hvem som kom først.*

Programmering av innebygde systemer

Flagging av avbrudd, «interrupt flag»

- Interruptstyringen har et antall «avbrudds-flagg»
  - Når et avbrudd er flagget, lagrer prosessoren status og starter avbruddshåndteringen (interrupt service routine “ISR”)
    - Kjøring av avbruddsrutinen (må/) tar ned flagget for avbrudded, slik at det ikke startes på ny.
  - Flaggene *kan* maskeres slik at vi velger hvilke avbrudd som benyttes.
  - Hvilke flagg som er tilgjengelige avhenger av prosessorarkitekturen
    - Tellere
    - Timere
    - Inngangspinner
    - Hardware porter
      - USB, m.fl.
    - Watchdog-timere :
      - Sjekker om systemet har hengt seg
      - F.eks hvis ISR-rutinene ikke rydder opp etter seg, eller en evig kø av interrupter.
        - For mange avbruddsmuligheter og for lange interruptrutiner kan forårsake dette
          - Mikrokontrollere er enkjerne-systemer.
            - Alt skjer i sekvens - Det er ingen «bakgrunnsprosesser»
        - *=> Vi kan beregne hvor lang tid håndtering av et avbrudd tar i en mikrokontroller..*
        - *=> Vi kan også beregne hvor ofte et interrupt burde forekomme..*

Board	**Digital Pins Usable For Interrupts**
Uno, Nano, Mini, other 328-based	2, 3
Uno WiFi Rev.2, Nano Every	all digital pins
Mega, Mega2560, MegaADK	2, 3, 18, 19, 20, 21 (pins 20 & 21 are not available to use for interrupts while they are used for I2C communication)
Micro, Leonardo, other 32u4-based	0, 1, 2, 3, 7

Programmering av innebygde systemer

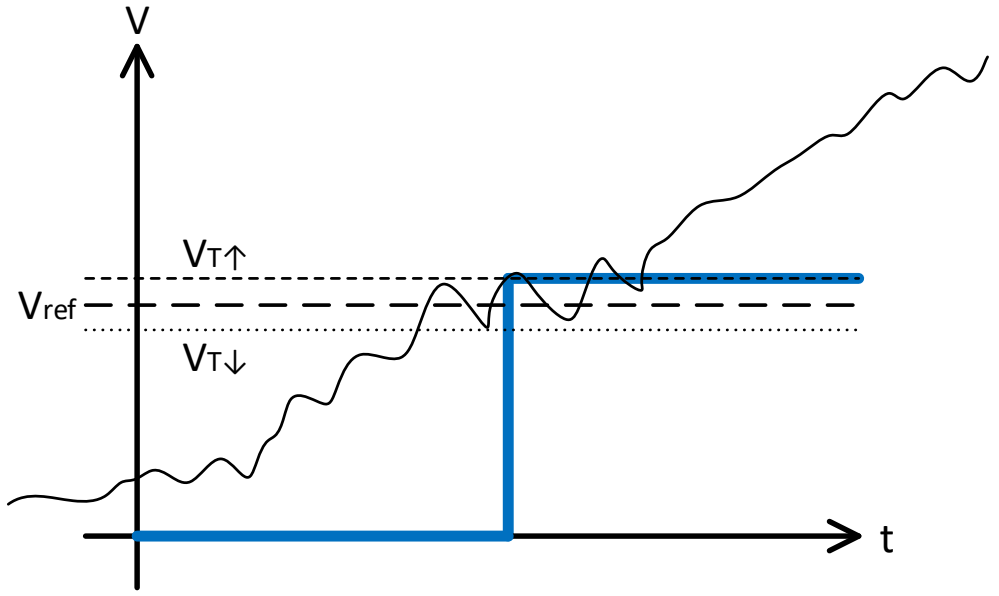
Avbruddsrutiner, ISR Interrupt service routines

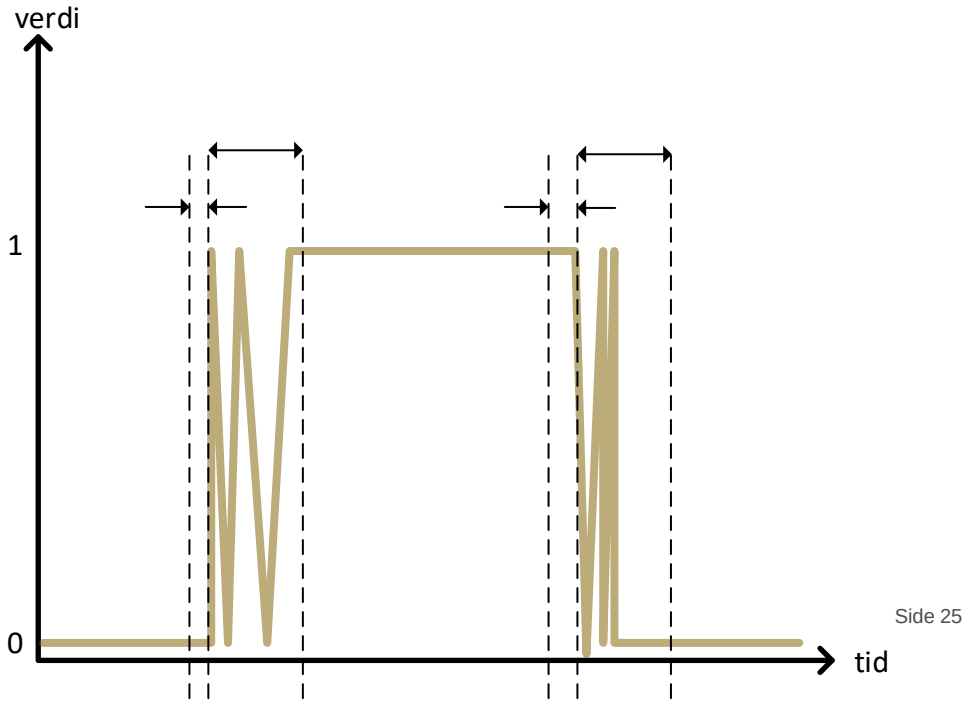
- Avbruddsrutiner er servicerutiner på lik linje med event service rutiner
  - En avbruddsrutine bør være kort, så den ikke blokkerer andre funksjoner lenge.
    - Blokkerende kode (venteløkker ol) bør ikke forekomme.
  - I mikrokontrollere er det lite behov for spesiell lagring av statusregistre ved håndtering av avbrudd
    - Man må holde styr på at man ikke endrer verdier som ikke tåler å endres i andre rutiner.
      - *=> Bruk lokale variable, evt lokale «static» variable fremfor globale i mest mulig grad*
  - I mikroprosessorer, så må man lagre registre ol. på stakken ved avbrudd
    - = mer komplisert og flere muligheter for feil (*IN2060 går i dybden på prosessorarkitektur*)
- *I IN1080 bruker vi ikke den innebygde avbruddstyringen til Arduino direkte, men rammeverket bruker det i enkelte rutiner*
  - Tonegeneratorer, `tone()`
  - Pulsbreddemodulering, `analogWrite()`, osv.
  - Venterutiner `delay()`
- *Overstyrer vi arduinorammeverkets interruptstyring, så risikerer vi at arduinokommandoene ikke virker*
  - Eks: `millis()`, `micros()` `delay()`, `delaymicros()`, `tone()`, `analogWrite()` …
  - *Mer om interrupter for arduino her:*
    - *https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/*

Programmering av innebygde systemer

Terkselverdier og hysterese

- Analoge verdier har støy (kommer tilbake til dette jf sensorer)
- En stigende/synkende verdi kan trigge mange events om vi kun ser på én referanseverdi
  - Hvis vi sjekker på forskjellige verdier på opp og nedtur, kan vi unngå unødvendige skift
    - Dette gir en form for hysterese, dvs ulik output avhengig av historikken (om vi er på vei opp eller ned)
- Brytere har prell
  - *Når vi skrur av eller på en bryter, kan spenningen på polene skrus av/på mange ganger i ett trykk.*
  - Hvis vi krever at en bryter må ha stabilisert seg en viss tid før vi aksepterer at den er trykket eller sluppet, kan vi begrense virkningen av prell.
  - Alternativt kan vi akseptere én endring umiddelbart, men ikke to tett fulgte hendelser.
    - NB: Konflikt mellom hurtig respons og stabilitet.
      - *Vi kan lage tjenesterutiner som håndterer alle tenkelige kombinasjoner…*





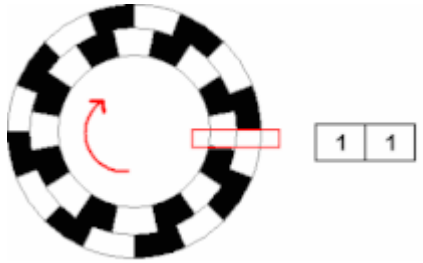
Programmering av innebygde systemer

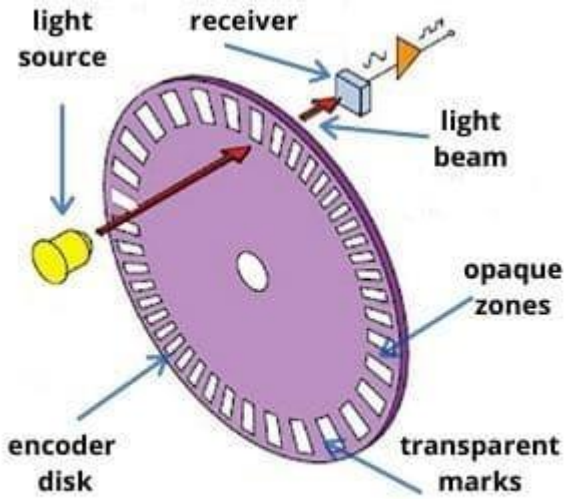
Tilstandsmaskiner (Finite state machines - FSM, Finite state automata FSA)

- Er et program eller en konstruksjon der utgangsverdier settes som følge av historikk og inngangsverdier
- Brukes i ulike typer kontrollsystemer og digital elektronikk
- kan lages med
    - maskinvare (IN3160- digital systemkonstruksjon)
    - programvare
        - Hendelsesstyrt programmering egner seg godt til å beskrive tilstandsmaskiner
- et hvilket som helst program *kan* kalles en tilstandsmaskin, *men*
    - Begrepet brukes gjerne om tilfeller der antallet tilstander er svært begrenset. (ie Finite)
        - Eks:
            - kombinasjonslås, trafikklysstyring, robot-gressklipper, etc.

Programmering av innebygde systemer

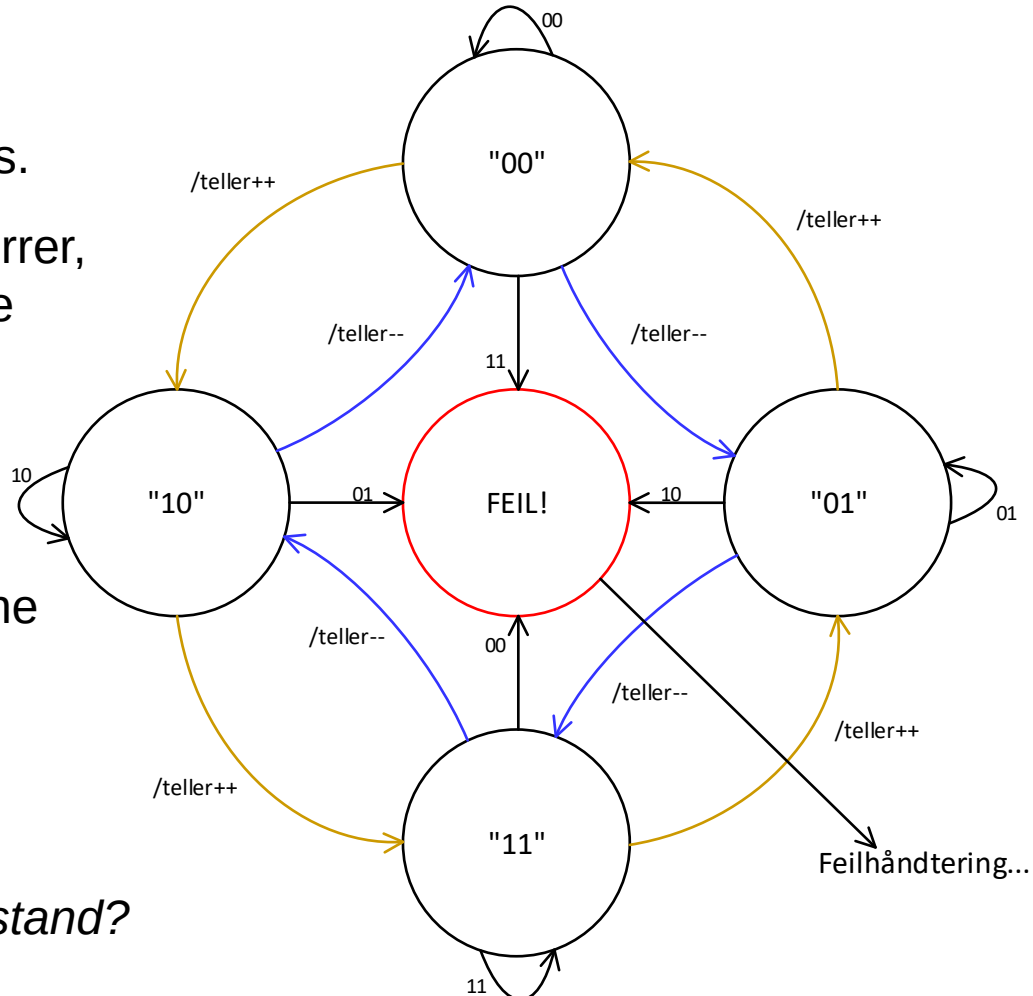
Tilstandsmaskin eksempel: Quadrature enkoder
Q.E. brukes til å gi informasjon om rotasjon…



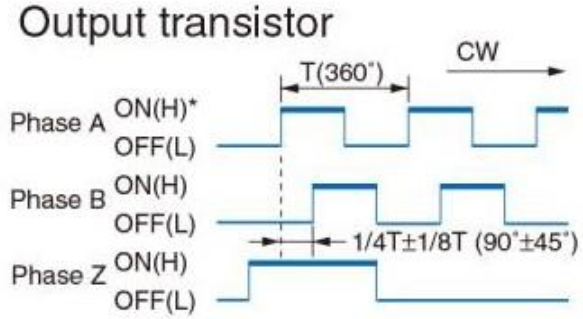


Optical encoder working principle

- Quadrature enkoder bruker
  - En eller to lyskilder
  - Et hjul med vinduer
  - To sensorer som registrerer lys.
- Sensorene står slik at om hjulet snurrer, så vil en sensor ha lys før den andre
  - Sensorene er
    - på når det er lyst
    - av når det er mørkt
- Leser vi av sensorene kan vi beregne retning og hvor fort hjulet snurrer-
  - Men det krever at vi husker tilstanden vi står i



Direction of rotation: CW
(as viewed from end of shaft)



- *Hvorfor har vi en feiltilstand?*

Mekanikk og programmering av innebygde systemer

- Lese:
  - Forelesningsfoiler, evt wikipedia el. om mekanikk.
  - COK: 5.0-5.10
- Oppgaver:
  - Regneeksempel 1 og 2 (Neste regneøvelse er etter påske)
  - Test deg selv (Kraftmoment/statikk):
    - https://phys.libretexts.org/Bookshelves/University_Physics/Exercises_(University_Physics)/Exercises%3A_College_Physics_(OpenStax)/09%3A_Statics_and_Torque
  - COK: 5.4, 5.6, 5.8. (Programmering av innebygde systemer)
  - Lag en hendelsesstyrt teller for quadrature-enkoderen (tilstandsdiagram forrige slide)
    - Trenger ikke ta hensyn til prell
      - Bruk pinne 4 og 5 på Arduino som innganger fra quadrature enkoderen.
    - (Feil kan rapporteres serielt.)