

Det matematisk-naturvitenskapelige fakultet

Institutt for Informatikk

IN1080 Mekatronikk

Mekanikk, programmering av innebygde systemer

Yngve Hafting

Universitetslektor

Robotikk og intelligente systemer

2022-02-04

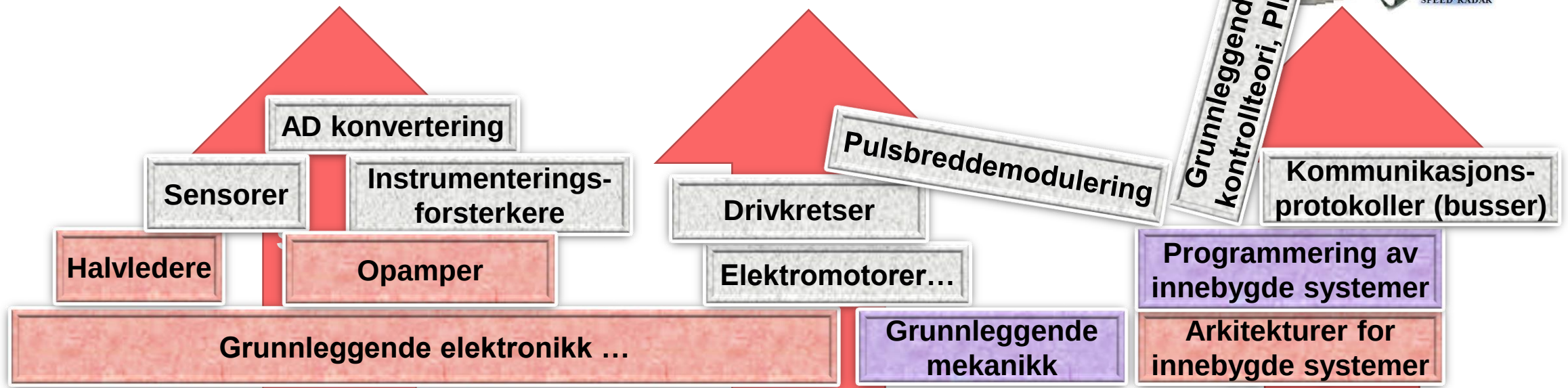
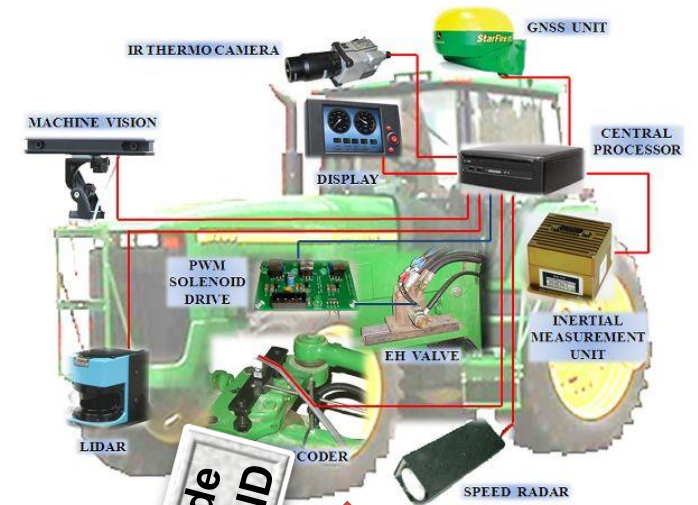
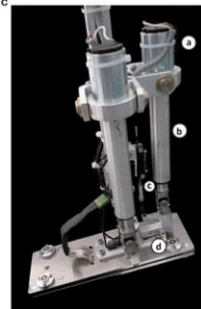
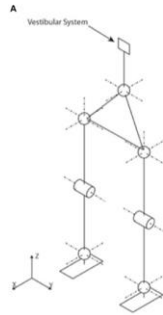
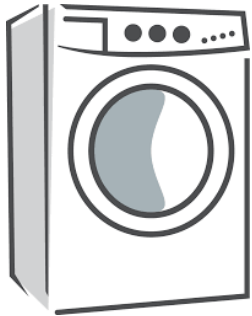


UNIVERSITETET
I OSLO



Hvor står vi – hvor går vi...

Formål: Å lage og programmere mekatroniske systemer



Grunnleggende Mekanikk

For å kunne bygge og programmere mekatroniske systemer trenger vi

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

Skal vi gå lengre trenger vi bredere systemforståelse

- 2. ordens systemer/ svingesystemer, osv

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*

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/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*

- $\mathbf{s}(t) = \int \mathbf{v} \cdot dt$

- $\mathbf{v}(t) = \frac{d\mathbf{s}}{dt}$ (alt. notasjon: $\mathbf{v} = \dot{\mathbf{s}}$)

- $\mathbf{a}(t) = \frac{d\mathbf{v}}{dt}$ ($\mathbf{a} = \ddot{\mathbf{s}}$)

- *Konstant hastighet:*

- $\mathbf{s} = \mathbf{s}_0 + \mathbf{v} \cdot t$

- *Konstant akselerasjon:*

- $\mathbf{a} = \mathbf{v}/t$

- $\mathbf{s} = \mathbf{s}_0 + \mathbf{v}_0 t + \mathbf{a} t^2 / 2$

- Størrelser

- Strekning (s) : m

- Hastighet (v) : m/s

- Akselerasjon (a) : m/s²

- Merk:

- Vektorer angis ofte med **fet** skrift

- *Prøv å holde styr på når vilkårene for å bruke skalare størrelser er oppfylt*

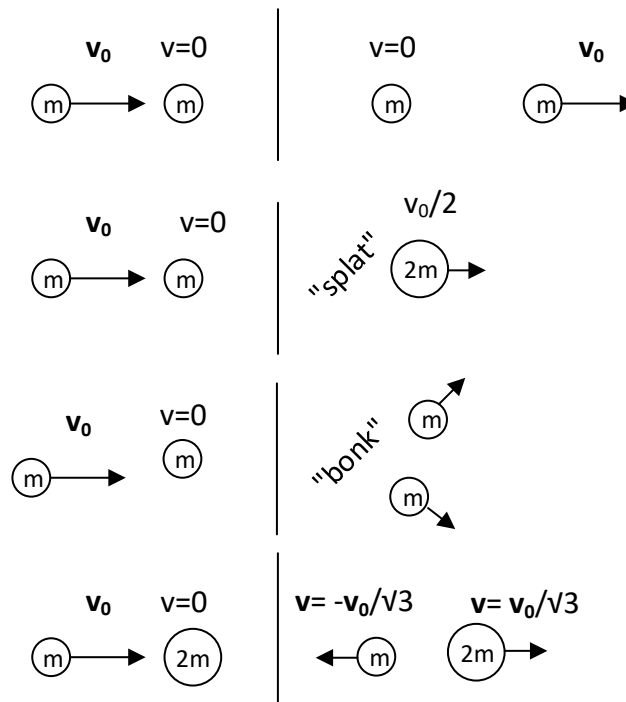
- 1D bevegelse i én retning, etc.

Grunnleggende Mekanikk

Bevegelsesmengde, kraft

- Bevegelsesmengde (*linear momentum*)

- $\mathbf{p} = m \cdot \mathbf{v}$ (enkelt partikkel)
- $\mathbf{p} = \sum \mathbf{m}_i \mathbf{v}_i$ (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
 - $\mathbf{F} = \frac{d\mathbf{p}}{dt} = m\mathbf{a}$ (også kjent som N2. lov).

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

Grunnleggende Mekanikk

Newtons lover

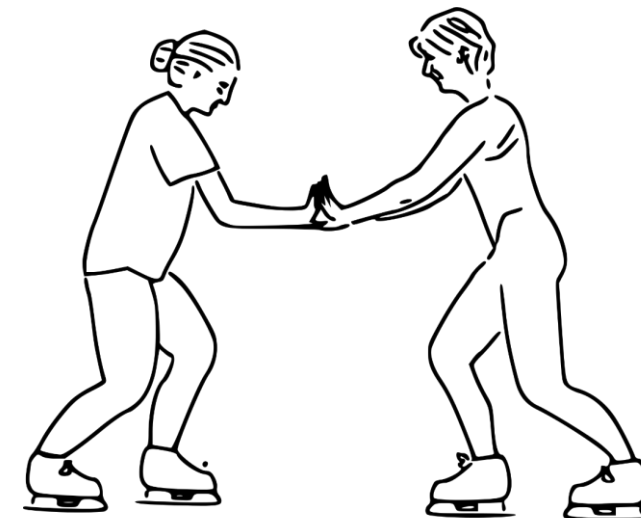
N1.lov:

- 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.
- $\sum F = m \cdot dv/dt$
- $F = m \cdot a$



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 (**G**)

- $G = mg$, (*m er masse oppgitt i antall kg*)

- Tyngdeakselerasjon (**g**)

- $g \approx 9,81 \frac{m}{s^2}$

- Potensiell energi

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

For alle praktiske formål holder vi oss på jordoverflaten hvis ikke annet er oppgitt

- Enhet for Energi er

- Joule ($1J = 1 Nm = 1kgm^2/s^2 = 1Ws$)
 - ($1kWh = 1000W \cdot 3600s = 3,6 MJ$)

Grunnleggende Mekanikk

Arbeid og effekt

- *Mekanisk Arbeid*

- $W = \mathbf{F} \cdot \mathbf{s}$

- $W = P \cdot t$ (*konstant effekt*)

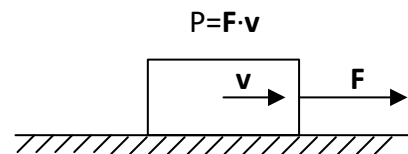
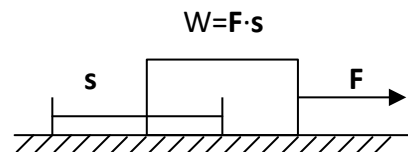
- $P = \frac{dW}{dt}$

- $P = \mathbf{F} \cdot \mathbf{v}$ Konstant kraft og fart

- 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):*

- $a = \frac{v^2}{r}, F = \frac{mv^2}{r}$

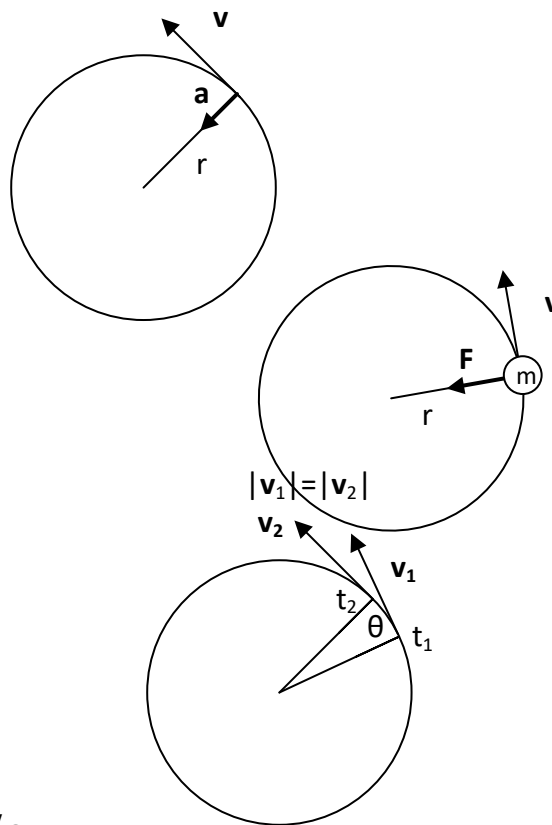
- $\omega = \frac{d\theta}{dt}$

- $\omega = \frac{v}{r}$

- $\alpha = \frac{\omega}{t}$, 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/s²

- Merk radianer er i prinsippet dimensjonsløs

- Omregning RPM til rad/s

- En runde = 2π , ett minutt = 60s

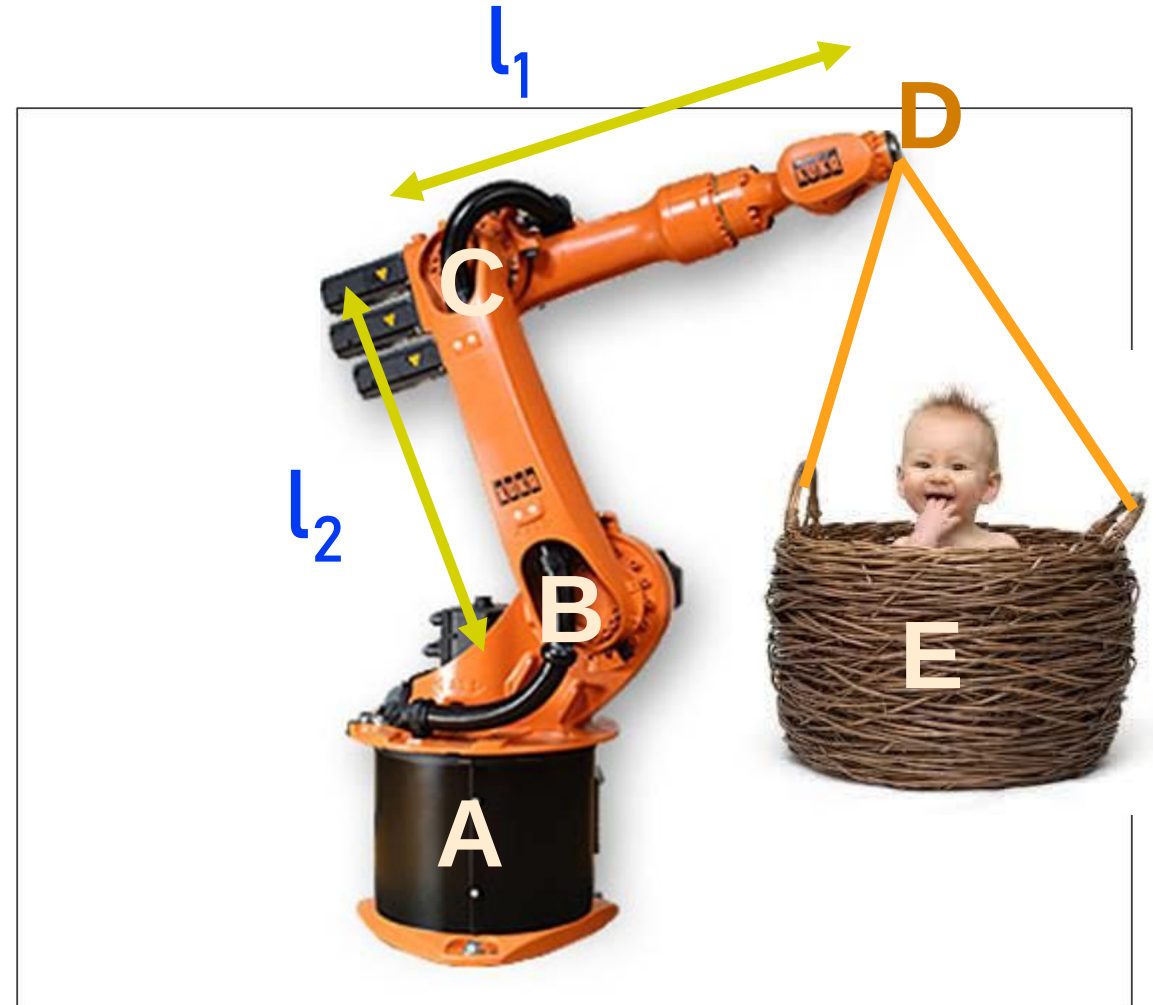
- $1\text{RPM} = 2\pi/60\text{s}$
 - $= \frac{\pi}{30}\text{s}^{-1} \approx 0,105\text{rad/s}$

Regneeksempel 1 (Vanskelig)

Hva er hastigheten (v) til kurven?

- Premisser:

- konstant hastighet, $\omega_A = 60$ RPM
 - Ingen gynging
- D står i forlengelsen av akse til A
- $l_{DE} = 1\text{m}$

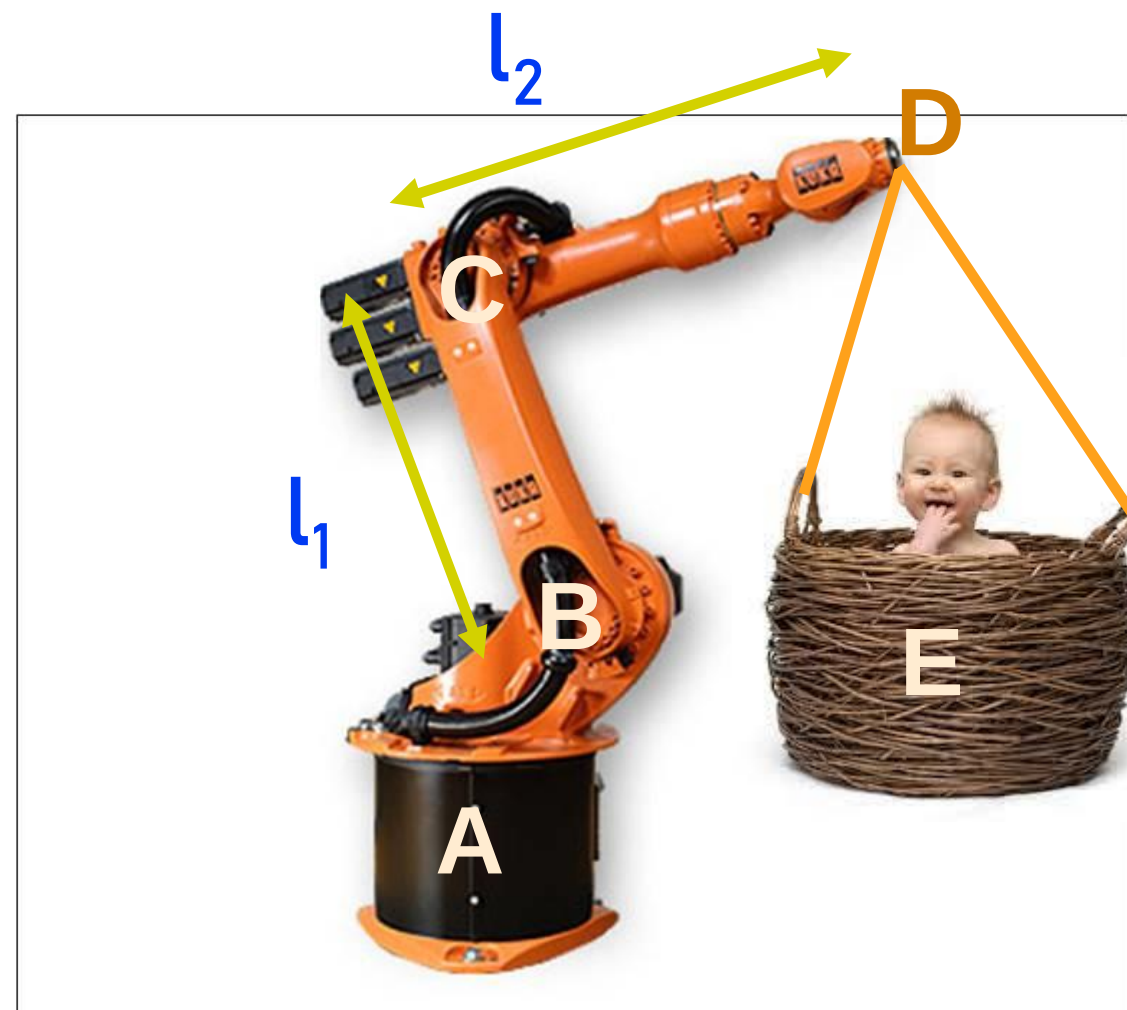


Regneeksempel 2

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

• La

- $l_1 = l_2 = 1\text{m}$
- Arm 2 har masse 1kg, jevnt fordelt
- Arm 1 har masse 1kg, jevnt fordelt
- $m_E = 10\text{kg}$



Grunnleggende Mekanikk

Kraftmoment (dreiemoment, torque)

- Dreiemoment (torque) er en krafts evne til å forandre et legemes rotasjon.
- Dreiemoment τ (tau) beregnes med kryssproduktet av avstanden (r) til legemets rotasjonspunkt og kraftvektoren (F).

- $\tau = r \times F$ (rekkefølgen avgjør retningen)

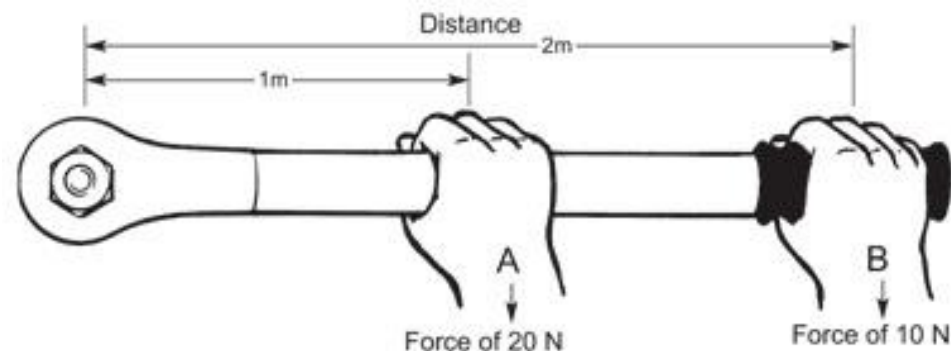
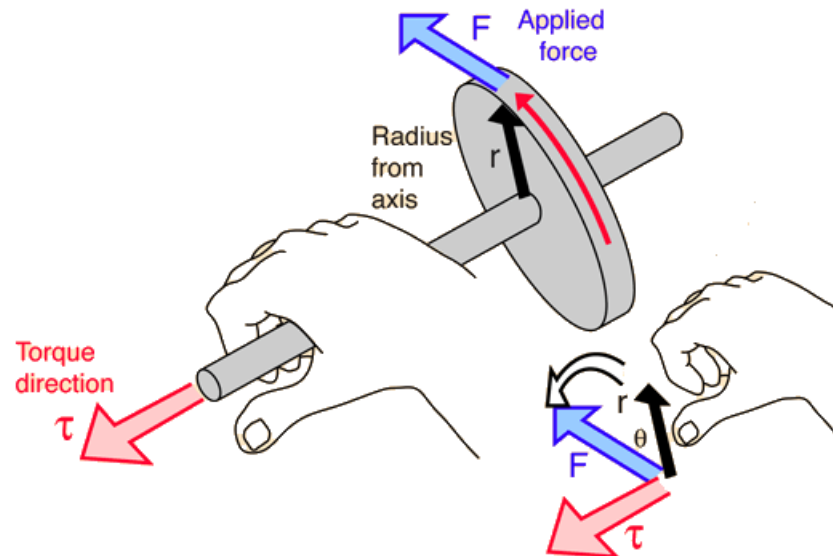
- Retningen til kryssproduktet er gitt ved høyrehåndsregelen, dvs. 90 grader på både radius og kraftvektor. (τ -tommel, r -håndflate, F -fingre).

- Størrelsen på kraftmomentet blir

$$\tau = r \cdot F \sin(\theta) \text{ eller } \tau = r \cdot F$$

- I svært mange praktiske sammenhenger så er F normalt på r , og da får vi $\tau = r \cdot F$ ("kraft · arm")

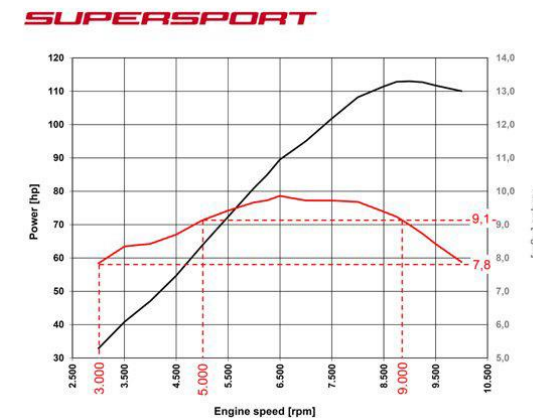
- Eks: Både A og B på figuren gir et kraftmoment på 20Nm (retning inn i skjermen) på pipenøkkelen, i sum: 40Nm



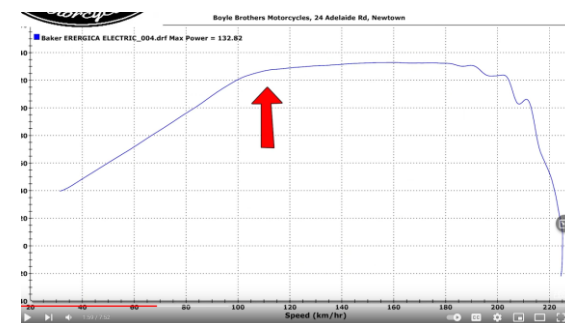
Grunnleggende Mekanikk

Kraftmoment og motorer, eksempel på tall.

- Når vi karakteriserer motorer, får vi gjerne oppgitt
 - Dreiemoment (Nm «Newtonmeter», US: ft. lb. (foot-pound) $1\text{Nm} \approx 0,74\text{ ft.lb.}$)
 - (Maksimal) effekt («Watt» W, «hestekrefter» hk evt. hp $1\text{kW} \approx 1,36\text{ hp.}$)
- 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 Motorsykel Bensin
 - 110 hp – 81 kW,
 - 93 Nm - 69 lb-f
 - Tung Motorsykel Elektrisk
 - 200 Nm - 147 ft.lb.
 - 80kW - 109 hp



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



Energica Ego Dyno run (Power curve)

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;
}
```

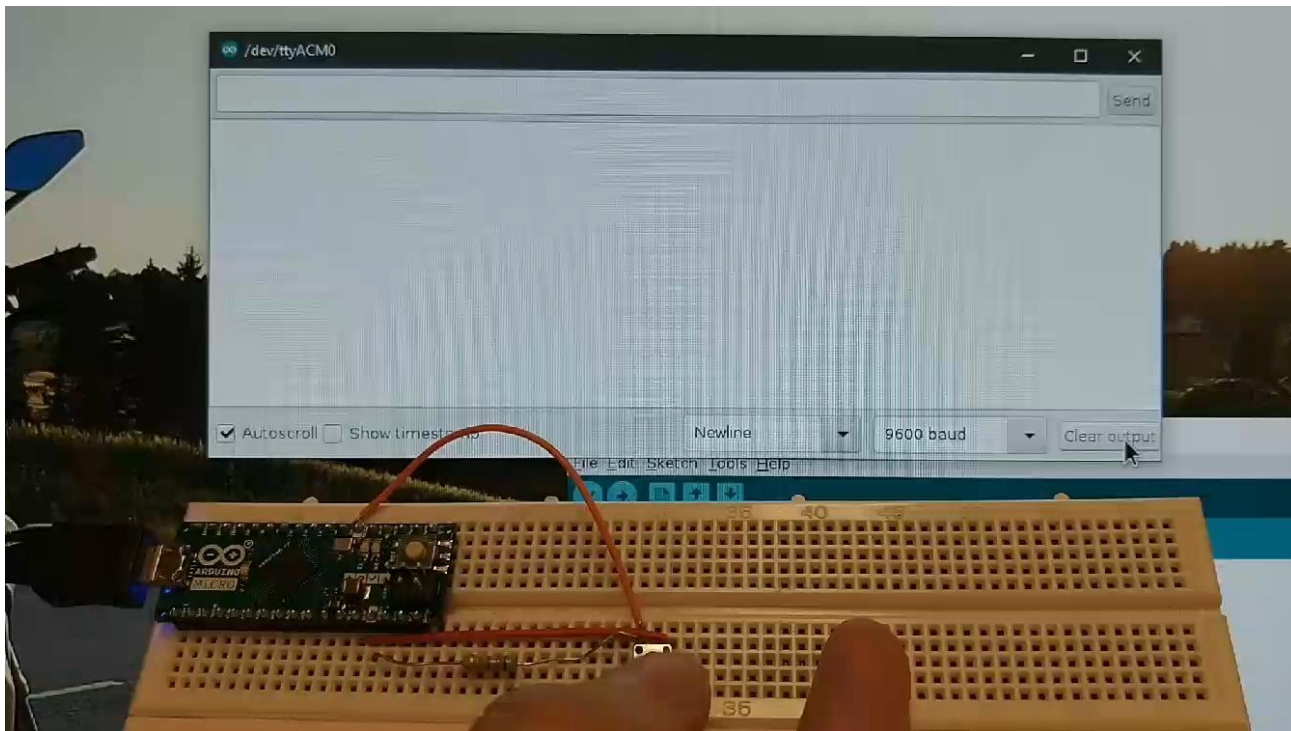
Vi har en hendelse kun når tilstanden endres
Verdien den endres til er ikke så viktig,
men vi må huske verdi fra gang til gang

```
// 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...
}
```

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;
}
```

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

```
// 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...
```

```
}
```

Programmering av innebygde systemer

Interrupter «Avbruddshåndtering»

- Mikrokontrollere og prosessorer *kan* være laget for avbruddshåndtering (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
 - Triggres 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 avbruddet, 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 enkjernerne-systemer.
 - Alt skjer i sekvens - Det er ingen «bakgrunnsprosesser»

- => *Vi 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

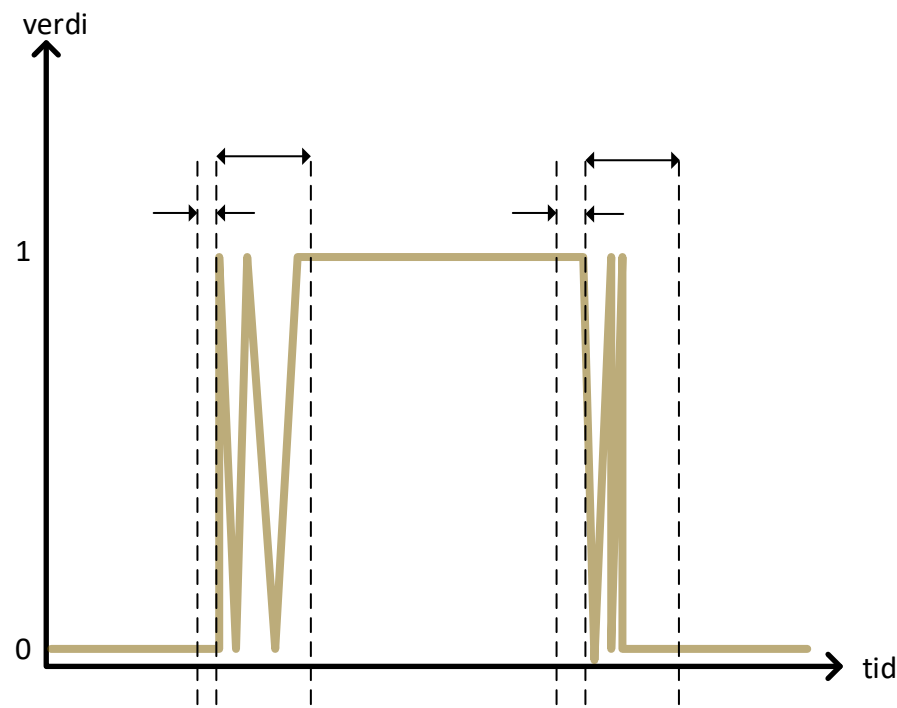
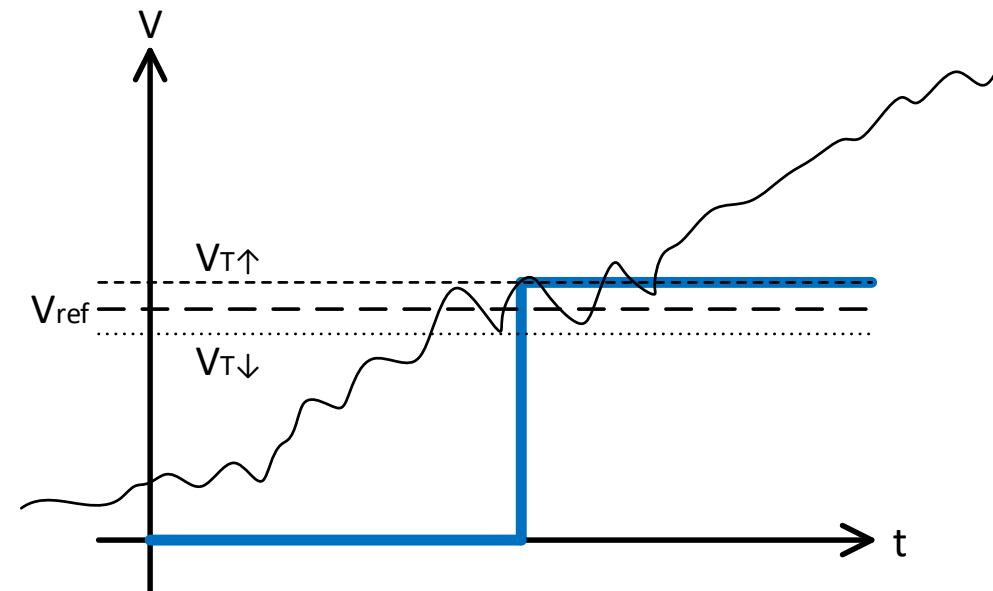
Avbruddsrutiner, ISR Interrupt service routines

- 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 trenger 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

Terskelverdier og hysteres

- 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 hysteres, 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

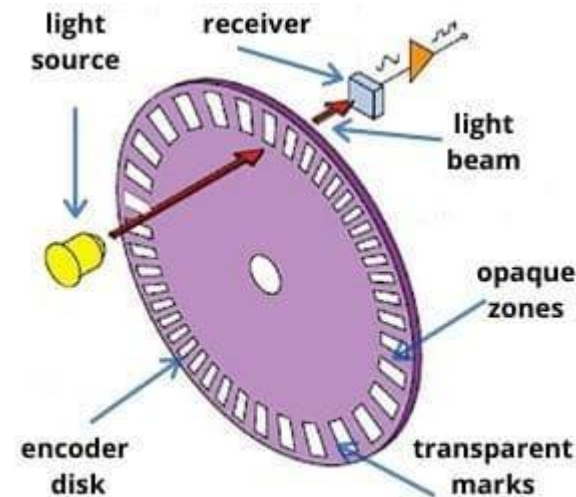
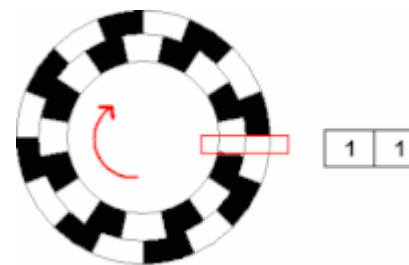
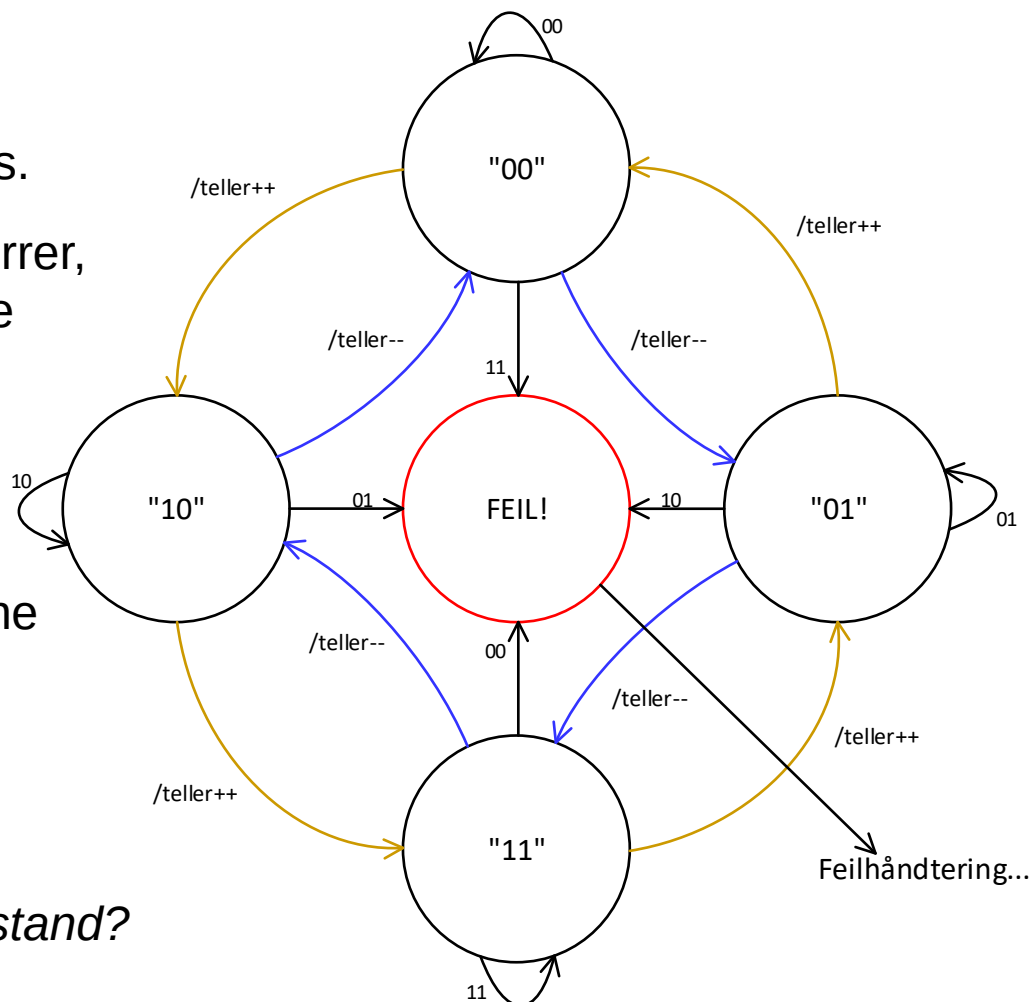
Tilstandsmaskiner (Finite state machines - FSM, Finite state automata FSA)

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

Programmering av innebygde systemer

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

- 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

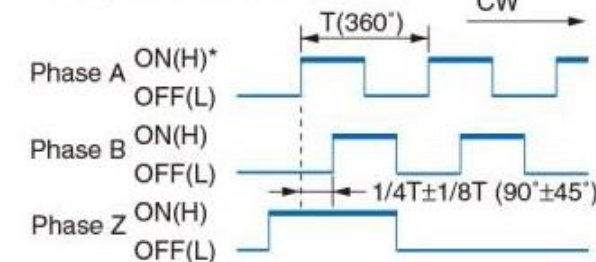


Optical encoder working principle



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

Output transistor



Mekanikk og programmering av innebygde systemer

- Lese:
 - Forelesningsfoiler, evt wikipedia el. om mekanikk.
 - COK: 5.0-5.10 (Viktigst: 5.0-5.5)
- Oppgaver:
 - Regneeksempel 1 og 2 (Til i morgen..?)
 - COK: 5.4, 5.6.
 - Lag en hendelsesstyrt håndtering av tilstander for quadrature-enkoderen
 - Trenger ikke ta hensyn til prell
 - Bruk pinne 4 og 5 på Arduino
 - Feil kan rapporteres serielt.