Internt notat

Måling og kalibrering av sensorverdier

Forspenning av trykksensorene

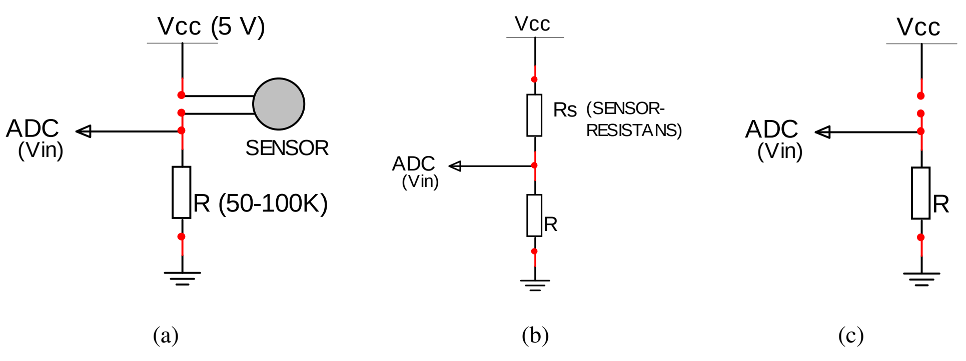
Figur **??** viser hvordan trykksensorene kan forspennes. Trykksensorens ene terminal er koblet til en tilførselsspenning V_{CC} på 5 V. Den andre terminalen går til jord gjennom en motstand R på 50–100 kΩ. Analog-til-digital-omformeren (ADC) måler spenningen mellom sensoren og motstanden.

Dette oppsettet gir en spenningsdeling mellom sensorresistansen R_S og motstanden R (fig. **??**). Når sensoren ikke brukes, er $R_S \approx \infty$ og kan regnes som et brudd (fig. **??**). Spenningen ADC-en måler, V_{IN}, er da 0 V. Ellers avhenger målingen av spenningsdelingen:

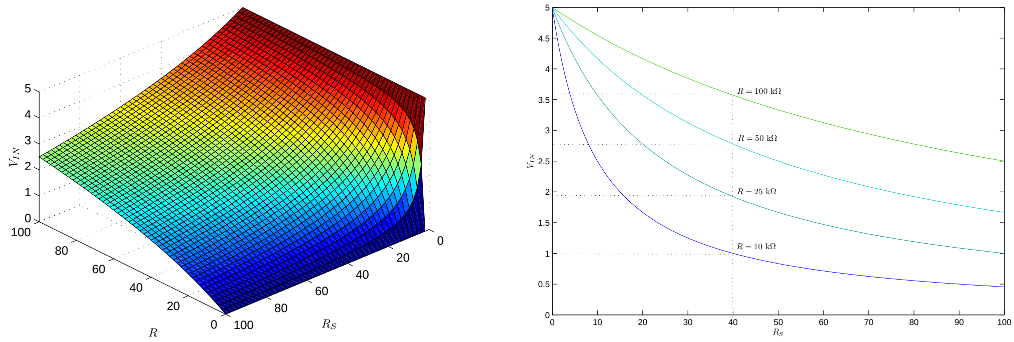
$$V_{IN} = V_{CC} \cdot \frac{R}{R_S + R} \tag{1}$$

Dersom R holdes konstant, sier vi at $V_{IN} = f(R_S)$ antar *hyperbolsk vekst*: V_{IN} synker når R_S øker, men den synker stadig mindre og mindre. Hvor markert denne stigningsendringen er, avhenger av verdien av R, som må velges i proporsjon med R_S slik at ADC-en får fornuftige voltverdier. Målinger på en sensor ga R_S = 10–100 kΩ, der 100 kΩ svarte til et forsiktig trykk og 10 kΩ er om lag den laveste oppnåelige resistansen ved bruk av tungen. Vi lar R variere og betrakter V_{IN} som en funksjon av R og R_S:

$$V_{IN} = f(x, y) = f(R, R_S)$$



Figur 1: Forspenning av trykksensor **??**, spenningsdeling **??** og brudd **??**



(a) (b)

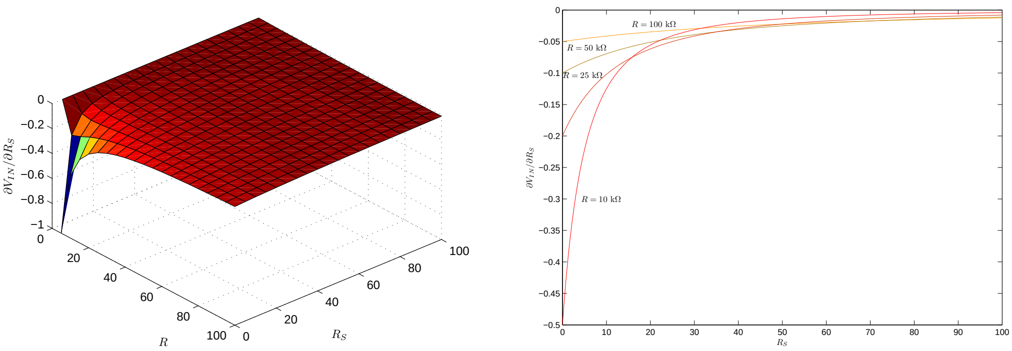
Figur 2: V_{IN} som funksjon av R og R_S, ligning (**??**)

Figur **??** viser plott av V_{IN} for $R, R_S \in [0, 100]$. Vi ser at jo større R-verdi, jo høyere er den laveste V_{IN}-verdien, som vi kan kalle $V_{IN\,\mathrm{min}}$. Det må imidlertid understrekes at vi *ønsker* at $V_{IN\,\mathrm{min}} > 0$ V, for 0 V er jo typisk spenningen som måles når sensoren *ikke* brukes ($R_S \approx \infty$). Når sensoren tas i bruk, vil vi få et *sprang* på minst minimumsverdien, $V_{IN\,\mathrm{min}}$, som selvfølgelig ikke bør være så lav at spranget ikke registreres.

Vi ser også at sammenhengen mellom R_S og V_{IN} blir *mer ulineær* (hyperbolsk) for lavere verdier av R. Dette fremgår enda tydeligere dersom vi betrakter den retningsderiverte av V_{IN} i R_S-retning, dvs. den partiellderiverte av V_{IN} med hensyn på R_S (fig. **??**):

$$\frac{\partial V_{IN}}{\partial R_S} = \frac{\partial}{\partial R_S}\left(V_{CC} \cdot \frac{R}{R_S + R}\right) = -\frac{V_{CC}R}{(R_S + R)^2} \tag{2}$$

For virkelig lave R-verdier, $R < 40$ kΩ, observerer vi *kraftig* stigning når vi beveger oss fra null i R_S-retning. (Stigningstallet til de lavere R_S–V_{IN}-kurvene i figur **??** endres altså kraftig i dette



(a) (b)

Figur 3: Plott av den partiellderiverte av V_{IN} med hensyn på R_S, ligning (**??**)

området.) Denne endringen følges av et jevnt nivå som er om lag det samme for alle R-verdier. For høye R-verdier, derimot, er det nesten ingen begynnende endring i R_S-retning, og nivået er «jevnt» hele veien. Selv om dette er en konsekvens av zoom-nivået, viser det tydelig forskjellen i linearitet for lave og høye R-verdier.

Ulinearitet er ikke nødvendigvis negativt. Si at vi ønsker at pekerfarten skal være konstant for $R_S = 40$–100 kΩ, men at den skal øke når $R_S < 40$ kΩ. Da er det gunstig med *lav oppløsning* for «lette trykk» og *høy oppløsning* for «harde trykk». Utlagt vil det si at vi bruker en stor del av intervallet av V_{IN}-verdier til å differensiere mellom ulike «harde trykk» ($R_S < 40$ kΩ), mens en mye mindre del delegeres til likestilte «lette trykk» ($R_S = 40$–100 kΩ). Fig. **??** viser at en R-verdi på 10–25 kΩ kan være egnet.

Digitale verdier

Gjennom ADC-en får vi en overgang fra den *fysiske* størrelsen V_{IN} til den *digitale* verdien `ADC_VARIABEL`,[1] som har en oppløsning på 10 bit. Fra databladet har vi følgende formel:

$$ADC = \frac{V_{IN} \cdot 1023}{V_{REF}} \tag{3}$$

der V_{REF} er *referansespenningen*, som settes lik V_{CC} (5 V). Uttrykket for V_{IN} blir

$$V_{IN} = \frac{ADC \cdot V_{REF}}{1023} \tag{4}$$

Den digitale verdien `0x000` representerer altså jord, og `0x3FF` (= `0b1111111111`, desimalt 1023) representerer referansespenningen *minus én LSB* (least significant bit). Av (**??**) er voltverdien til én bit 0,005 V, så den digitale maksverdien svarer til 4,995 V. (I praksis har dette lite å si, ettersom vi aldri kommer ned på en sensorresistans på 0 kΩ.)

I den følgende teksten tillater vi oss å bruke V_{IN} og `ADC_VARIABEL` om hverandre. Det er lettere for oss å betrakte en voltverdi enn et heltall i intervallet [0, 1023], men *programkoden* må selvfølgelig forholde seg til det sistnevnte. Uansett kan faktiske verdier bare oppnås gjennom forsøk.

Kalibrering av nullnivå

ADC-verdien når sensoren ikke brukes, kaller vi *nullnivået*, og er ideelt $V_{IN} = 0$ V. Når sensoren tas i bruk, får vi et sprang fra nullnivået, og sensorens musefunksjon kalles. Når sensoren ikke lenger brukes, får vi et sprang *tilbake til* nullnivået, og musefunksjonen opphører.

Det knytter seg flere problemstillinger til dette, først og fremst til nullnivået som sådan. Bastian har påpekt at det ikke er garantert at nullnivået er 0 V; feil med bøylen eller forskjell i hodestørrelser kan gi en høyere spenning selv om sensoren ikke brukes. Dette oppfordrer til en *selvkalibrerende* rutine, som fastsetter nullnivået på nytt hver gang sensoren brukes.

[1] «`ADC_VARIABEL`» er et pseudo-navn på C-variabelen som inneholder verdien til en vilkårlig ADC. De faktiske variabelnavnene er ikke viktige her.

Kodeutdrag 1: Pseudo-C-kode for selvkalibrering

```c
const int SPRANG = 100; // Global, hardkodet verdi

int main() {
  int nullverdi = ADC_VARIABEL; // Setter nullnivået
                                // første gang
  // Pollingsløkke
  do {
    sensorverdi = ADC_VARIABEL;

    if(nullverdi - sensorverdi > SPRANG) {
      musefunksjon(nullverdi); // Kaller musefunksjon
                               // med nullnivå
                               // som referanse

      // Eventuell tidsforsinkelse

      nullverdi = ADC_VARIABEL; // Setter nullnivået
    }                           // på nytt

    // Eventuell tidsforsinkelse

  } while(1);
}
```

Kodeutdrag **??** skisserer en slik rutine. Når programmet starter, leser det ADC-verdien og setter nullnivået til dette (`nullverdi`). Deretter går det inn i en løkke hvor ADC-verdien kontinuerlig avleses og sammenlignes med den fastsatte `nullverdi`. Dersom differansen er større enn den globale konstanten `SPRANG`, tolkes det som et sprang, og musefunksjonen kalles. (Merk at `musefunksjon()`, som ikke er beskrevet ytterligere, kalles med `nullverdi` som argument. Dette er for at funksjonen skal kunne oppdage et sprang *tilbake* til `nullverdi`, for så å opphøre.) Når musefunksjonen er ferdig, har vi en eventuell tidsforsinkelse før `nullverdi` fastsettes på nytt. På dette viset kalibreres nullnivået hver gang sensoren brukes.

I eksemplet over «polles» kun én sensor. Dersom koden for alle sensorene bygges opp rundt prinsippet om at *bare én sensor kan brukes av gangen*, vil selvkalibreringen enkelt kunne stanse sensorer som «løper løpsk». La oss si at nullnivået til sensor A, som flytter pekeren til venstre, gjennomgår en endring og blir gradvis høyere. Endringen blir til slutt så stor (>`SPRANG`) at musefunksjonen kalles, og pekeren begynner å bevege seg «av seg selv». Ettersom det *gamle* nullnivået brukes som referanse, vil brukeren aldri kunne oppnå et sprang *tilbake* (stoppe pekeren) ved å trykke på sensor A. Men dersom en *annen* sensor aktiveres, sensor B, vil musefunksjonen til sensor A avbrytes ut fra prinsippet om én sensor av gangen. Dermed vil nullnivået til sensor A kalibreres på nytt. Slik vil endringen i nullnivå ikke utgjøre noe varig problem.

Midlingsfilter

Sensorresistansen har vist seg å variere hyppig ved målinger. Det kan derfor være gunstig å anvende et midlingsfilter på strømmen av innleste verdier for å «jevne» den ut. AVR-programmet kan implementere et filter av orden N ved å lagre de siste N ADC-verdiene i et array og beregne gjennomsnittet. Kodeutdrag **??**, som skisserer en «pollingsløkke» for én enkelt sensor, implementerer filteret med differanseligningen

$$y[n] = \frac{1}{4}(x[n] + x[n-1] + x[n-2] + x[n-3]) \tag{5}$$

der $y[n]$ svarer til `gjennomsnitt`, $x[n]$ til `sensorverdi` og $x[n], \ldots, x[n-3]$ lagres i arrayet `verdier` slik at *etter linje 10* svarer `verdier[0]` til $x[n-3]$, `verdier[1]` til $x[n-2]$, `verdier[2]` til $x[n-1]$ og `verdier[3]` til $x[n]$.

Hvis dette skal kombineres med deteksjon av sprang, bør filteret slås av og på slik at sprangene ikke glattes ut. Ellers vil vi få en forsinkelse før musepekeren f.eks. beveger seg og før den slutter å bevege seg; særlig det siste vil være et hinder for presisjonen.

Filteret vil imidlertid ikke ha noen erindring om hva som fant sted før det ble slått på; det må basere seg på «standardverdier» for verdiene det mangler. I det 4-punkts filteret i kodeutdrag **??** er disse verdiene $\{0, 0, 0, 0\}$, men dette medfører en utjevning «fra null». Det er i stedet om å bruke den sist målte V_{IN}-verdien som filterets «referansenivå»: $\{V_{IN}, V_{IN}, V_{IN}, V_{IN}\}$.

Kodeutdrag 2: Pseudo-C-kode for 4-punkts midlingsfilter, ligning (**??**)

```
1  int main() {
2    int verdier[4] = {0, 0, 0, 0};
3    int sensorverdi = 0;
4    int gjennomsnitt = 0;
5
6    // Pollingsløkke
7    do {
8      sensorverdi = ADC_VARIABEL;
9
10     verdier = { verdier[1], verdier[2], verdier[3], sensorverdi };
11
12     gjennomsnitt = (verdier[0] + verdier[1]
13                     + verdier[2] + verdier[3]) / 4;
14
15     // Eventuell tidsforsinkelse
16
17   } while(1);
18 }
```