Obligoppgåve, TMA4101

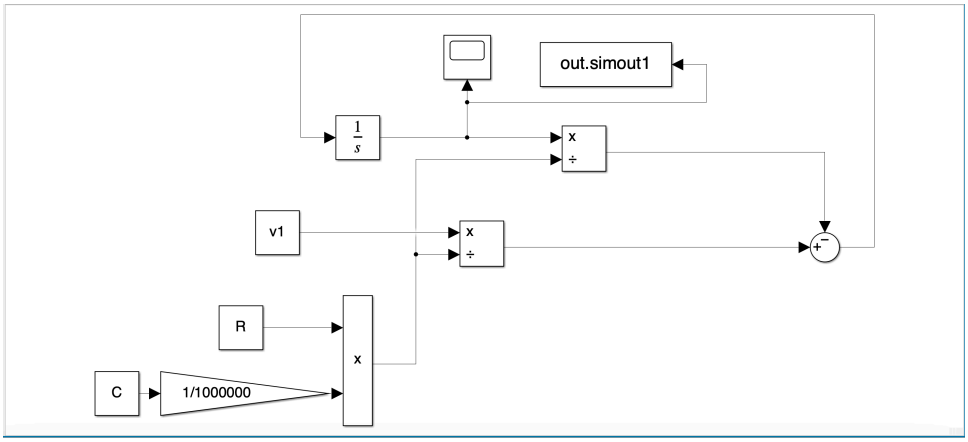
Eg begynte med å finne komponentane eg ville bruke, og ende opp med ein kondensator på 100 mikrofarad, og ein 10k ohms motstand. Desse var vald for å gi ein tidskonstant som var stor nok til gjere målingar med.

$$T = RC = 10000 * \frac{100}{1000000} = 1s$$

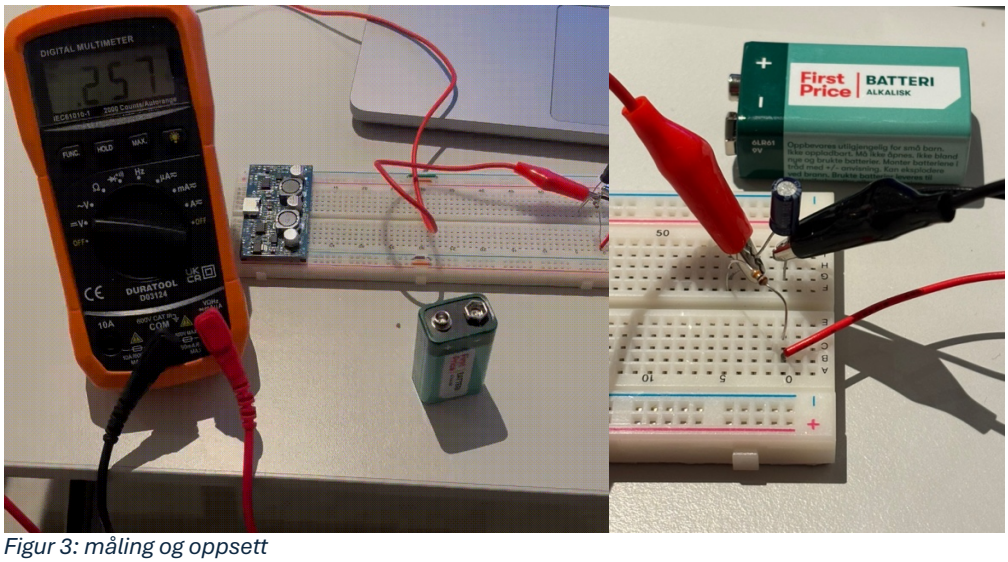
Målte så den faktiske spenninga til batteriet, og den faktiske motstanden til motstanden. Dette var for at modellen skulle bli meir nøyaktig enn dersom eg hadde brukt dei oppgitte verdiane, ettersom desse ikkje er heilt nøyaktige.
Definere så dette i matlab, og simulerte modellen i simulink. Eg brukte ein «out» blokk i simulink for å eksportere data til matlab, og brukte dette i plottinga seinare

```
1   v1=9.71; %volt
2   C=100; %mikrofarad
3   R=9770; %ohm
```

Figur 1: målte verdiar, definert i matlab



Figur 2: Simulink modell



Figur 3: måling og oppsett

Etter å ha utført målinga, brukte eg litt (mykje) tid på å gå gjennom videoen i figur 3 og stoppa den for kvar gong skjermen på multimeteret viste ny verdi. Skreiv så ned verdiane samt tida det hadde gått frå batteriet var kopla på. Resultatet visast i tabell 1 under.

Tabell 1: Målt spenning over tid

TID	Spenning
0,000	0,257
0,110	0,374
0,560	3,600
0,890	4,390
1,280	5,580
1,480	6,480
1,860	7,160
2,060	7,690
2,400	8,100
2,710	8,420
3,040	8,670
3,320	8,860
3,750	8,900
3,830	8,970
4,200	9,060
4,470	9,180
4,800	9,270
5,140	9,340
5,750	9,440
6,000	9,480
6,540	9,530
7,340	9,570
7,870	9,600
8,680	9,620
9,560	9,640
10,000	9,640

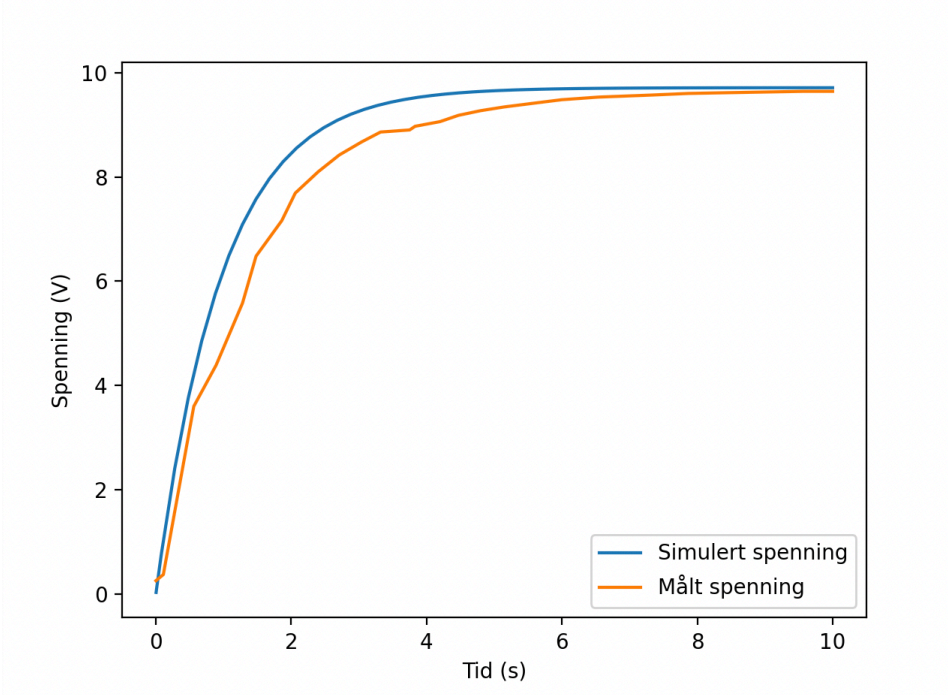
Neste steg var å plotte dataa i Python. Ettersom simulink og videoopptaket mitt ikkje brukte same tidssteg, måtte eg definere to tidsaksar, som du kan sjå i programmet under.

plotting.py > ...

```python
import matplotlib.pyplot as plt

#har to grafar med ulike tidssteg

#definerer den simulerte grafen:
#har data frå "out" blokk i simulink
tid_sim=[0.00315420502409259,0.0157912443834379,0.0789764411801644,0.278976441180164,0.478976441180164,0.678976441180164,0.878976441180164,
spenning_sim=[0.0312977936495277,0.155681138453341,0.754027564759297,2.41191227098214,3.76289767408607,4.86379543407454,5.76090052134128,6.

#definerer den målte dataen
tid_målt=[0.000,0.110,0.560,0.890,1.280,1.480,1.860,2.060,2.400,2.710,3.040,3.320,3.750,3.830,4.200,4.470,4.800,5.140,5.750,6.000,6.540,7.3
spenning_målt=[0.257,0.374,3.600,4.390,5.580,6.480,7.160,7.690,8.100,8.420,8.670,8.860,8.900,8.970,9.060,9.180,9.270,9.340,9.440,9.480,9.53


#plotter dei to grafane over kvarande
fig, ax = plt.subplots()

ax.plot(tid_sim, spenning_sim, '-')
ax.plot(tid_målt, spenning_målt, '-')
plt.legend(["Simulert spenning", "Målt spenning"], loc="lower right")
plt.xlabel("Tid (s)")
plt.ylabel("Spenning (V)")

plt.show()
```

Figur 4: Program for plotting av data i Python

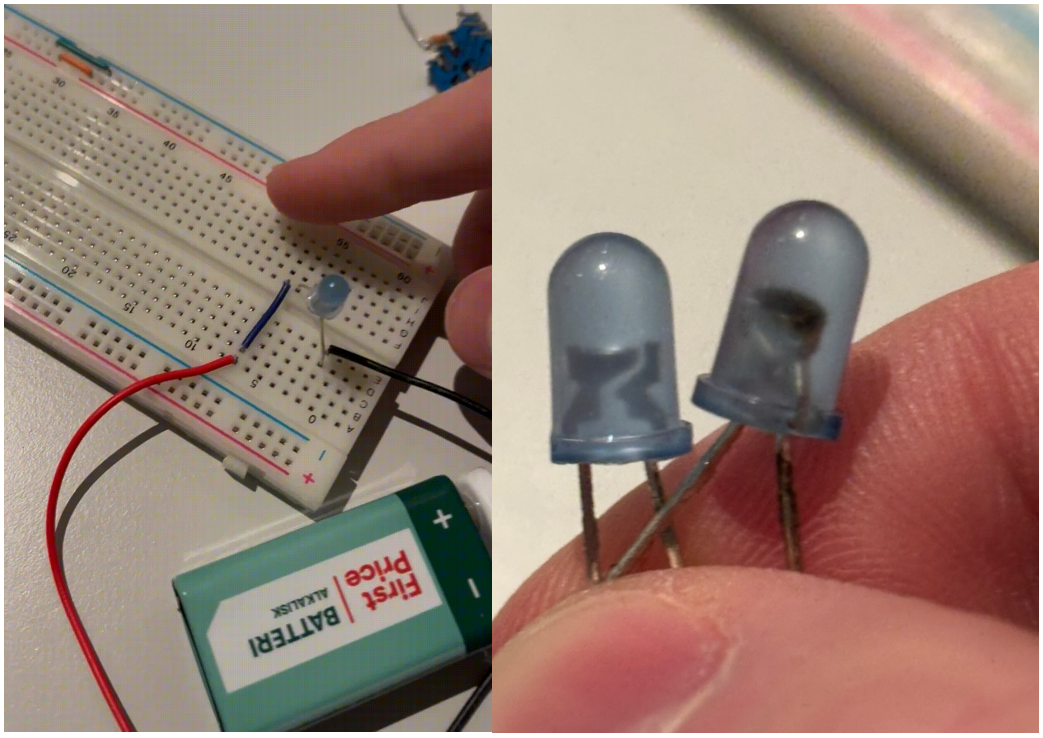
Resultatet av dette var denne plotten, som viser at den simulerte og den målte spenninga var ganske nære kvarandre.



Figur 5: Målt og simulert spenning plotta over kvarande

For å gjere oppgåva meir morosam, ville eg eigentleg snu kondensatoren den andre vegen og sprenge den. Dette advarte storebroren min meg om, ettersom han hadde gjort det same på Gløshaugen og utløyst brannalarmen og måtte betale for utrykkinga.

I staden velde eg å sprenge ein lysdiode. Eg må innrømme at dette er litt mindre spennande, men også litt tryggare (og billegare for meg).



Figur 6: brenning av lysdiode. Dette lukta ikkje godt, men brannalarmen gikk ikkje