# Implementation og test

Implementeringen og test af hvert modul er beskrevet i hver deres afsnit.

Implementationen af kode er lavet i Particle Build (IDE) browser-portal. Nogle tests er lavet med Particle app’en.

Der er lavet individuelle tests til hver af de enkelte moduler, og til sidst en samlet test til at afprøve samspillet mellem alle parter.

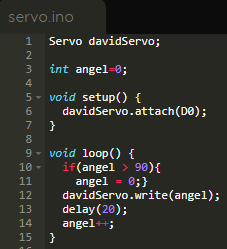
## Servomotor

https://www.electronicwings.com/particle/servo-motor-interfacing-with-particle-photon

Servomotoren styres ved at regulere dutycycle, dog er det ikke nødvendigt at forholde sig til dette pga. de indbyggede biblioteker, der sørger for at vinklen, der ønskes at servomotoren skal have, blot er et tal (integer, float mm.), som skrives ud til servomotoren.

Til at styre en servomotor skal der skal først laves en servoklasse med typen *Servo.* Derefter skal servoklassen knyttes til en pin med kommandoen *servoklasseNavn.attach(pin).* Når man vil skrive til servomotoren, altså ændre vinklen bruger man *servoklasseNavn.write(angel in degrees)* kommandoen.

Nedenfor er implementeringen af testen, som først sætter vinklen til 0 og derefter øger vinklen med 1 integer-værdi. Denne test viser det range servomotoren skal være i stand til at udføre.



Figur 1 - Kode til styring af servomotor

Test af dette script fungerer som forventet; øger vinklen gradvist fra vinklen 0 til 90.

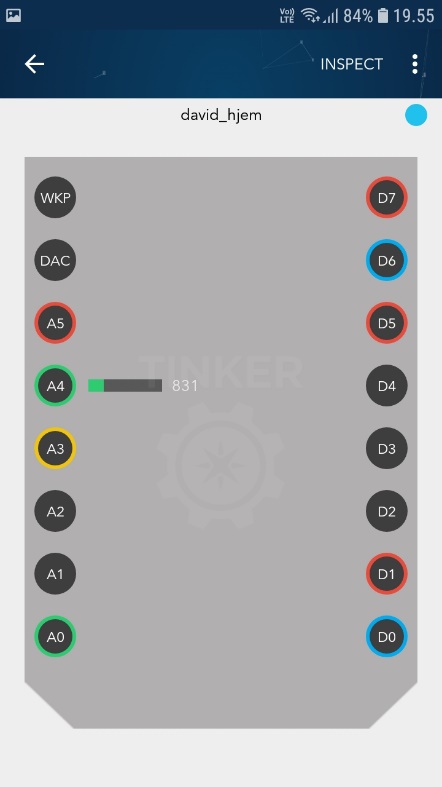
Til det endelige program skal servomotoren blot udføre to vinkler 0 og 90. 0 grader til at starte med og 90 grader når der er kommet post.

## Kontakt

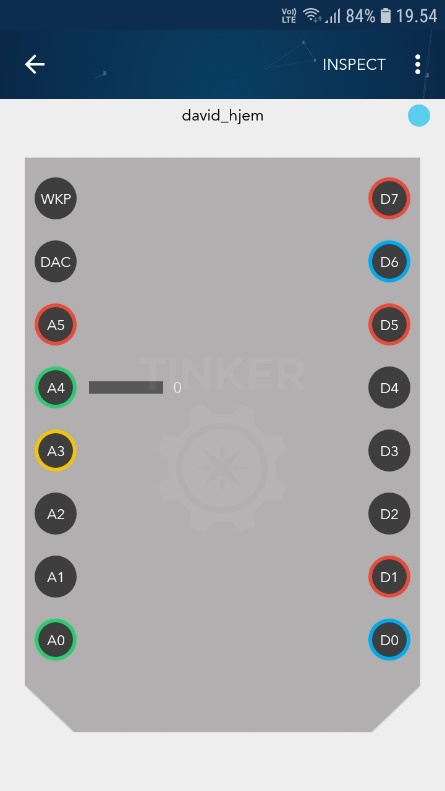
https://arduinotech.dk/shop/magnetisk-kontakt-aflang/

https://www.instructables.com/id/Magnetic-Door-Sensor-and-Arduino/

Kontakten er testet med Particle app’en, hvor der er læst fra pin A4, som er den læsbare pin fra kontakten (den anden ledning går i GND). Når de to dele til kontakten er forbundet, er outputtet 0, men når de fjernes fra hinanden, aflæses der en betydelig højere værdi fra den analoge pin.



Figur 2 - Kontakten er afbrudt



Figur 3 - Kontakten er forbundet

Ud fra dette kan der konkluderes at der kan skelnes mellem forbundet og afbrudt forbindelse. Denne kontakt skal til det endelige program vække Photon’en fra sleepmode, enten ved at detektere at denne analoge værdi øges eller falder.

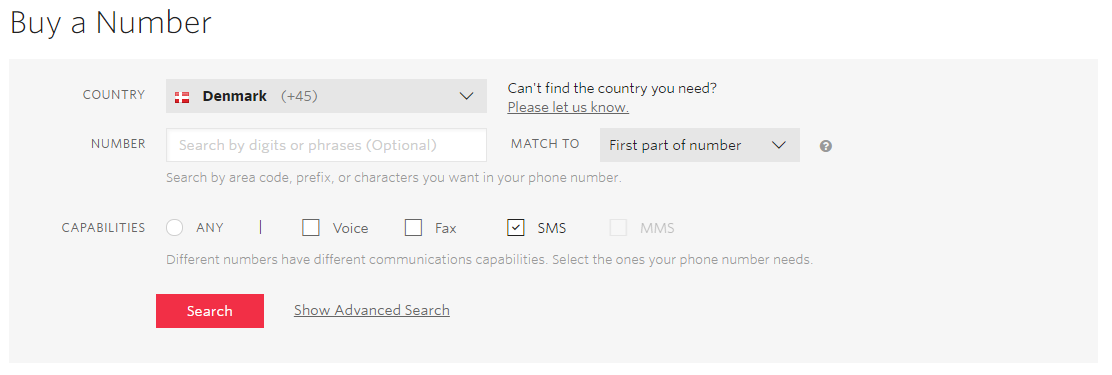
## SMS-service

https://www.twilio.com/docs/sms/tutorials/how-to-send-sms-messages-particle-photon

Implementation af SMS-servicen er lavet i samspil mellem forskellige services. Følgende steps er udført for denne implementation.

1. Køb af telefonnummer, som skal bruges som afsender.
   1. Købt hos Twilio.com
2. Opsæt Webhook, som mellemled mellem Twilio og Photon
3. Kode der aktiverer Webhook’en og sender en SMS

### Twilio service



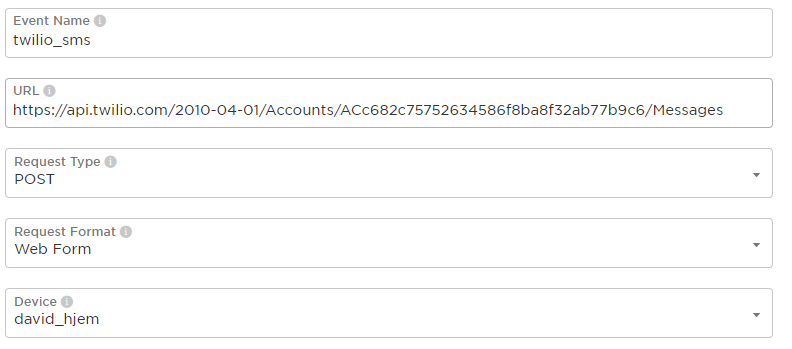
Figur 4 - Køb af telefonnummer

Efter at have oprettet en Twilio-konto, kan man med en trial konto oprette et telefonnummer. Der er lidt begrænsninger ved en trial konto, men det er muligt at anskaffe et nummer der kan klare sms’er. Dog følger der i sms’en en lille trail reklame med (kan ses i et senere afsnit).

### Webhook

https://www.twilio.com/docs/sms/tutorials/how-to-send-sms-messages-particle-photon

Der er blevet oprettet en Webhook integration på console.particle med følende data, som gør det muligt at sende data fra Particle enheder til en anden web service.



Figur 5 - Opsætning af Webhook

Event navn: Der er valgt et navn til denne Webhook integration.

URL: En URL på den API-service hvor data skal sendes til når den her Webhook bliver udløst. URL’en indeholder Twilio Account SID, så den korrekte konto findes.

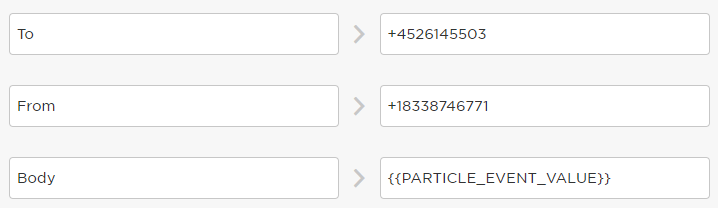
Request type: Valgt til at være POST, som sender data ud til Twilio servicen.

Request format: Valgt til at være Web Form, som er det format Webhook’en vil videregive data.

Device: Navn på den enhed, der kan benytte denne integration.

**Dynamisk Data**

Disse data bliver sendt afsted sammen med Webhooken



Figur 6 - Opsætning af dynamiske data

To - Til hvilket nummer – Brugers nummer

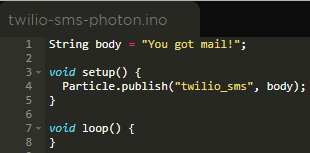
From – Fra hvilket nummer - Twilio-nummeret

Body – Indholdet som skal sendes i SMS’en, som er den

Dette er data som Twilio-servicen kan forstå.

## Photon – Kode

Koden er meget simpel, den skal blot trigger den Webhook, som går under navnet ”twilio\_sms”. Den tekst der skal sendes som sms, er argument to, som i dette tilfælde er en string, der har følgende data: ”You got mail!”.



Figur 7 - Kode der sender en sms

<https://docs.particle.io/reference/device-os/firmware/photon/>

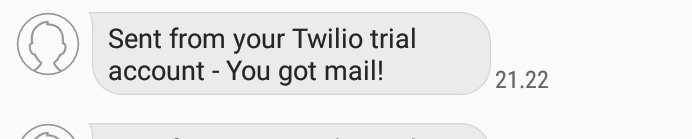
Funktionen *Particle.publish* bruges til at publish (offentliggøre) en event til lyttere og i dette tilfælde Webhooken. Der er muligt også at sende nogle parametre med. Første parameter er navnet på den event der skal der skal udløses (Webhooken: *twilio\_sms*), den anden parameter er data der kan sendes med. Den fulde syntax er her:   
*Particle.publish(String eventName, String data)*

Det er muligt at undlade denne dataparameter, og blot specificere den ønskede sms-tekst i Webhooken i feltet *Body*. Men body er en particle\_event\_value, som netop er denne parameter data, som sendes med.

Det er vigtigt at funktionen til at udløse sms’en ikke forekommer for ofte og slet ikke i en evig loop, da det kan crashe Twilio-services.

### Test

Testen udføres blot ved at flashe testkoden til ens Particle Photon. SMS’en bliver modtaget efter få sekunder og ser således ud:



Figur 8 - Sms modtaget fra Twilio

Som det kan ses, er den body der blev sendt med Publish funktionen, sendt med i SMS’en. Og som nævnt tidligere følger der også en automatisk starttekst med, pga. trail kontoen.

# Verification

Efter at have testet og alle dele hver for sig er næste step at samle alle funktionerne til et samlet system. Systemet skal kunne styre en servomotor, modtage data fra en kontakt, udløse Webhooken.

Til det samlede system er der dog tilføjelser. Systemet sættes i sleepmode, og reagere på en faldende værdi af kontakten. Det vil mindske strømforbruget af systemet.



Figur 9 - Kode til systemet