

# Swagway

Carl-Emil Grøn Christensen & Mathias Dannesbo  
Teknikfag A: El 11. maj 2012



Navn: \_\_\_\_\_

Klasse: \_\_\_\_\_

## Eksamten 2012 – Teknikfag A

**Eksamensbekendtgørelse - BEK nr. 1018 af 25.08.2010 § 20, stk. 5**

*Jeg bekræfter herved med min underskrift, at opgavebesvarelsen er udarbejdet af mig.*

*Jeg har ikke anvendt tidligere bedømt arbejde uden henvisning hertil.*

*Opgavebesvarelsen er udfærdiget uden anvendelse af uretmæssig hjælp og uden brug af hjælpemidler, der ikke har været tilladt under prøven.*

Dato: \_\_\_\_\_ Underskrift:\_\_\_\_\_



Navn: \_\_\_\_\_

Klasse: \_\_\_\_\_

## Eksamten 2012 – Teknikfag A

**Eksamensbekendtgørelse - BEK nr. 1018 af 25.08.2010 § 20, stk. 5**

*Jeg bekræfter herved med min underskrift, at opgavebesvarelsen er udarbejdet af mig.*

*Jeg har ikke anvendt tidligere bedømt arbejde uden henvisning hertil.*

*Opgavebesvarelsen er udfærdiget uden anvendelse af uretmæssig hjælp og uden brug af hjælpemidler, der ikke har været tilladt under prøven.*

Dato: \_\_\_\_\_ Underskrift:\_\_\_\_\_

Teknikfag A eksamen:

# SWAGWAY

af

Carl-Emil Grøn Christensen & Mathias Dannesbo

## Forord

Gennem teksten vil der være angivet link til “Issues” på GitHub siden for projektet.<sup>1</sup> Disse Issues har været brugt som projektstyring samt bugtracker igennem projektforløbet. Links er angivet som i enden af denne sætning.<sup>#1</sup> Det er ikke nødvendigt for forståelsen af denne rapport at følge linksne.

Vi vil gerne takke Steffen Linnerup Christiansen for hjælp med mekanikken, samt Kristian Holm Nielsen for hjælp med forståelsen af PID.

---

<sup>1</sup><https://github.com/neic/Swagway>

# Indhold

<b>Indhold</b>	<b>6</b>
<b>1 Indledning</b>	<b>7</b>
<b>2 Input</b>	<b>8</b>
2.1 Sensor . . . . .	8
2.2 Styring . . . . .	15
<b>3 Control</b>	<b>16</b>
3.1 Hoved loop . . . . .	16
3.2 Filter . . . . .	17
3.3 Regulering . . . . .	18
<b>4 Output</b>	<b>20</b>
4.1 H-broens virkemåde . . . . .	20
4.2 PWM . . . . .	21
4.3 Overvejelser . . . . .	21
4.4 Motorcontroller . . . . .	22
<b>5 Auxiliary</b>	<b>27</b>
5.1 Mainboard . . . . .	27
<b>6 Mekanik</b>	<b>31</b>
<b>7 Konklusion</b>	<b>31</b>
<b>8 Perspektivering</b>	<b>32</b>
<b>Tabeller</b>	<b>33</b>
<b>Figurer</b>	<b>33</b>
<b>Litteratur</b>	<b>33</b>
<b>A Arbejdsdeling</b>	<b>34</b>
<b>B Kildekode</b>	<b>34</b>
B.1 <code>swagway.ino</code> . . . . .	34
B.2 <code>ADXL345.h</code> . . . . .	40
B.3 <code>ADXL345.cpp</code> . . . . .	42

## 1 Indledning

Transport enheder bliver mere og mere avancerede, og der skabes ofte nye enheder. Specielt mobilitet og anvendelighed har været i højsædet i mange nye produkter. Det handler om hurtigt at kunne komme fra punkt a til punkt b, uden at det kræver større besvær. Derudover er løsninger som er intuitive at bruge også fremskreden, og et produkt som opfylder alle disse punkter er Segwayen. Den er hurtig, kræver næsten ingen erfaring at køre på og er lille. Man kan hurtigt suse fra en afdeling til anden.

Formålet med projektet er at bygge en motoriseret selvbalsancrende tohjulet transportenhed. Det er en klon af en Segway som er et kommersielt produkt. Kendtegnet ved Segwayen er, at de to hjul sidder på samme aksel, hvor elektronik og motorene sørger for at, selv med en fører, holde enheden i balance. Når en fører



Figur 1: Den mekaniske udformning af Swagway

læner sig frem eller tilbage, kompenserer Segwayen ved køre i samme retning, for at balancere den. For at dreje enheden vipper føreren håndtaget til den ene eller anden side, som man ønsker at dreje til. Segwayen kan købes for cirka halvfjerdstudsinde kroner, hvis man ønsker at eje sådan en.

Projektoplæget lyder:

Formålet er at bygge en balance robot på to hjul, v.z en Segway klon.

Minimums målet er at få robotten til at balancere. Derefter få den til at køre og kunne styre den hvis tid og evner rækker til det.

Hovedeufordringen er at holde enheden lodret. For at gøre dette skal man kende enhedens vinkel i forhold til lodret og omsætte denne vinkel til et signal til motoren. Når vinklen stiger driver motorene hjulene som så flytter enheden og brugerenes tygdepunkt ind over akslen igen. Det kan se som tre isolerede problemstillinger som passer ind i I-C-O-modellen:

Input Måle Swagwayens hældning i forhold til lodret

Control Omsætte vinklen til et signal til motoren.

Output Drive to motoren baseret på signalet.

Projektet er udarbejdet ud fra mikrokontrolleren Arduino Uno, i programmet arduino. Mikrokontrolleren bliver programmeret via. programmet Arduino, som nærmest er i sproget C++ med et Arduino bibliotek. Da vi har modtaget undervisning i denne platform, har dette været det oplagte redskab at benytte til løsningen.

## 2 Input

### 2.1 Sensor

Efter nogen research af andre køretøjer og mindre robotter der fungerer på samme måde, var det klart at gyroskoper og accelerometere kunne måle vinklen så hurtigt og præcist som det var krævet.

#### Sensor hardware

Gyroskoper og accelerometere har indtaget forbrugerens produkter, og kortlægger allerede nu deres bevægelser. De faldende priser og faldene energiforbrug har gjort, at gyroskoper og accelerometere er langt mere tilgængelige for alle. Mobiltelefoner, spilkonsoller og kameraer kan nu måle hvordan du bevæger dig, og flere og flere enheder importerer også disse bevægelses sensorer, for at adskille sig fra de andre.

Disse bevægelses sensorer er avancerede MEMS (mikroelektromekaniske systemer) teknologier, som inddører gyroskoper, accelerometere og kompas sensorer. I dette projekt har vi arbejdet med gyroskoper og accelerometere, som vi også kender fra vores smartphones. Swagwayen skal, lige som en smartphone kan måle om den står eller ligger ned, måle i hvilken vinkel den står i.

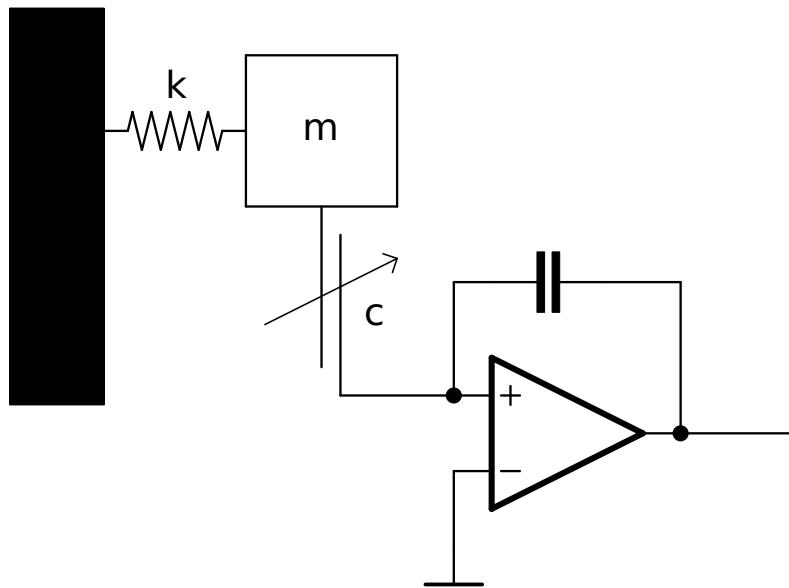
#### Gyroskop

Et gyroskop mäter vinkelhastigheder, hvor hurtigt noget drejer om en akse. Hvis denne vinkelhastighed integreres over tid finder man vinklen som gyroskopet har flyttet sig. Problemet med at integrere gyroskopdata er, at pga. måleusikkerheder vil

det målte nulpunkt drive væk fra det fysiske nulpunkt. Moderne gyroskoper er en MEMS. Det betyder, at det både er en mekanisk og elektrisk funktion i et system.

### Accelerometer

Et accelerometer kan måle accelerationer. Man kan med hjælp fra tangens og data fra to akser udregne den vinklen accelerometeret står i forhold til jordens tyngdekraft. Problemet med dette er at accelerometer måler alle accelerationer, ikke kun jordens tyngdekræft. Når Swagwayen fx accelererer, bremser eller køre over en sten bliver den udregnede vinkel forkert, da denne også frembringer en spike i accelerationsmålingen. Et accelerometer måler en bevægelse, som enten er forårsaget af



Figur 2: Et accelerometers interne MEMS-struktur

tyngdekraften eller en bevægelse. Som set på figur 2 måles accelerationen i et moderne MEMS accelerometer ved en ændring i kapacitansen,  $c$ , som skyldes en ændring i massen,  $m$ , position. Kapasitansen fører til en ændring i spændingen, som herefter digitaliseres via ADC og kan benyttes af en mikrokontroller som Arduinoen. Dette gøres for de tre akser x, y og z, og accelerationen kan nu måles i alle retninger.

### IMU

For at løse problemet med at gyroskopet driver og problemet med at accelerometret ikke er nøjagtig når det bliver påvirket af andre accelerationer end tygdekraften kan

man bruge begge sensorer sammen. Man bruger begge sensoreres data og samler dem i et filter. Se mere herom i sektion 3.2.

De fleste moderne accelerometere og gyroskoper kommer i små SMD pakker som er vanskelig håndlodde. Heldigvis er det forholdsvis let at få disse sensorer monteret på et breakoutboard. Efter at have kigget på andre Segway kloner og balancerobotter faldt valget på en IMU fra Sparkfun. Se figur 3. En inertial measurement unit, IMU, er en enhed som kombinerer flere sensorer. IMUen fra Sparkfun et



Figur 3: 6DOF IMU breakoutboard fra Sparkfun. (CC BY-NC-SA 3.0, Sparkfun)

ADXL345 accelerometer og et ITG-3200 gyroskop. Accelerometret og gyroskopet er begge tre-akset. Det giver IMUen seks frihedsgrader (DOF). Swagwayen skal dog kun bruge tre frihedsgrader; to akser fra accelerometret og en fra gyroskopet, men IMUen med seks frihedsgrader var det mindste med digital interface.

## I<sup>2</sup>C

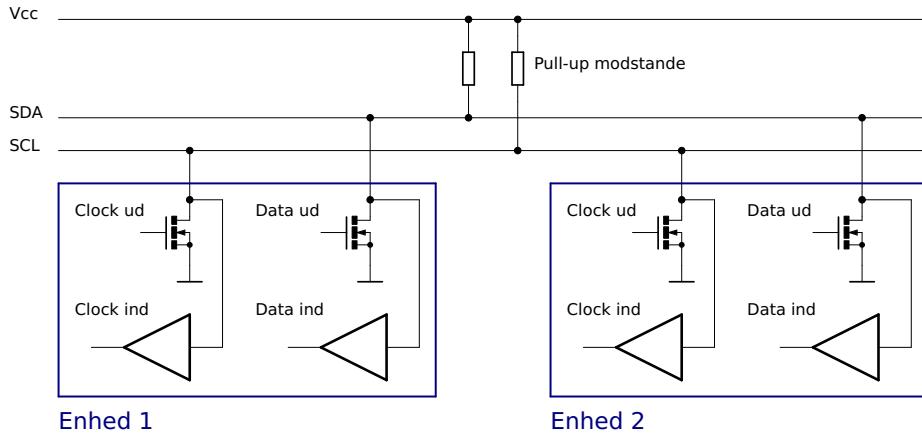
Kommunikation med IMUen sker digital over en I<sup>2</sup>C-bus. På bussen er der en master<sup>2</sup> og en eller flere slaves. I Swagwayen er Arduinoen master og accelerometret og gyroskopet er begge slaves. I<sup>2</sup>C bussen består af to fysiske forbindelser, en serial data linje (SDA) og en serial clock linje (SCL). Alle enheder på bussen forbinder til disse to forbindelser. Linjerne er open-drain med eksterne pull-up modstande. Det vil sige at i hver enhed sidder der MOSFETter der trækker spændingen på linjen mod GND efter behov. De eksterne pull-up modstande trækker linjerne op igen. Se figur 4.

På Arduinoen er I<sup>2</sup>C SDA på A4 og SCL på pin A5 med indbygget pull-up modstande på 20 kΩ–50 kΩ. [ATMega, 2011, s. 317] De er aktiveret som standard når man bruger Wire biblioteket.<sup>3</sup>

Arduinoens spænding er 5 V og IMUens er 3.3 V. IMUen bliver forsynet med 3.3 V fra Arduinoen, men SDA og SCL er koblet direkte til Arduinoens pins som har 5 V som reference. Den korrekte løsning ville være at lave en level-shifter imellem en 5 V- og en 3.3 V-del af I<sup>2</sup>C-bussen, men når pull-up modstandene er så høje som

<sup>2</sup>Det er mulig at lave et multi-master opstilling hvis alle masterne understøtter det.

<sup>3</sup>For reference se linje 75 i `utility/twi.c` i Wire biblioteket.

Figur 4: Eksempel på I<sup>2</sup>C-netværk

de er det stadig forsvarligt at kører IMUen direkte med 5 V som  $V_{IO}$ -reference. Der er flere eksempler på det samme blandt andre på Internettet der bruger samme chips.

Den letteste måde at kommunikere over I<sup>2</sup>C fra en Arduino er at bruge Wire-biblioteket som kommer sammen Arduino-softwaren: `#include <Wire.h>` For at få Arduinoen til at tilslutte sig bussen som master kaldes `Wire.begin()`; uden parametre. Slaverne på I<sup>2</sup>C-bussen har alle en adresse som fremgår af deres respektive datablade.

ADXL345 og ITG-3200 indeholder begge flere register med forskellige funktioner. Nogle registre indeholder opsætningen for chippen, andre indeholder det data som chippen mäter. Hvis man, fra Arduinoen, vil skrive til et af registrene køre man følgene kode:

```

1  Wire.beginTransmission(deviceAddress); // begynder en
   session med enheden med deviceAddress
2  Wire.write(regAddress); // sender adressen på den register
   man vil ændre
3  Wire.write(value); // sender værdien registeren skal
   indtage
4  Wire.endTransmission(); // afslutter sessionen

```

At læse et register er lidt mere omfattende:

```

1  Wire.beginTransmission(deviceAddress); // begynder en
   session med enheden med deviceAddress
2  Wire.write(regAddress); // sender adressen på den register
   man vil ændre
3  Wire.endTransmission(); // afslutter sessionen

```

```

4
5   Wire.beginTransmission(deviceAddress); // begynder en ny
     session
6   Wire.requestFrom(deviceAddress, numberofBytes); // sender
     anmodning på at modtage numberofBytes
7   int i = 0;
8   while (Wire.available()) { // så længe der er data tilgæ
     ngeligt
9     buffer[i] = Wire.read(); // modtag data og fyld det i
     buffer
10    i++;
11  }
12  Wire.endTransmission(); // afslutter sessionen

```

Disse to funktioner bliver senere refereret til som hhv. `writemem()` og `readmem()`.

## IMU biblioteker

I begyndelsen af projektet var alt kommunikationen med ADXL345 og ITG-3200 programmeret i `swagway.ino`. Det virkede og Arduinoen modtog data fra IMUen, men koden var svær at vedligeholde og gik tit i stykker. Der var desuden en fejl igennem længere tid der resulterede i at den vinkel der blev udregnet fra gyroskopet, `gyroAngle`, var ca. dobbelt så stor som den burde være.<sup>#20</sup> Efter mange ufuldendte forsøg på at rette fejlen blev alt gammel kode der var releateret til I<sup>2</sup>C slettet. Der blev istedet brugt et bibliotek passende til gyroskopet hvor alle funktioner var implementeret. Biblioteket hedder ITG-3200.

Det var ikke til at finde et ligende bibliotek til accelerometret, så med inspiration fra `itg3200`-biblioteket blev der skrevet et. Biblioteket består af to filer. En headerfil, `ADXL345.h`, og selve funktionaliteten, `ADXL345.cpp`.

Headerfilen indeholder først definitioner på konstanter som enhedsadressen og registrere:

```

23 #define ADXL345_ADDR_SDO_HIGH 0x1D
24 #define ADXL345_ADDR_SDO_LOW 0x53
25
26 // Registers
27 #define BW_RATE          0x2C // RW SETUP: Output rate and low
     power mode
28 #define POWER_CTL        0x2D // RW SETUP: Power control
29 #define INT_SOURCE       0x30 // R  INTERRUPT: Status
30 #define DATA_FORMAT      0x31 // RW SETUP: Self-test and data
     format
31 #define DATA_X0          0x32 // R  SENSOR: Data
32
33 // Bitmaps
34 #define STANDBY_MODE     0x00 // 0000 0000
35 #define MEASURE_MODE      0x08 // 0000 1000

```

Værdierne og navnene er hentet fra [ADXL345, 2011, s. 23] At have definitioner på alle konstanter gør koden lettere læselig, så man ikke skal søge referencer i databladet.

Efterfølgene er der defineret en klasse med alle funktioner relateret til accelerometeret:

```

37 class ADXL345
38 {
39   public:
40     float scaleFactor[3];
41     float voltage;
42
43   ADXL345();
44
45   void init(unsigned int address);
46
47   // SETUP: Mode
48   void setStandbyMode();
49   void setMeasureMode();

```

I ADXL345.cpp er funktionalitet skrevet. Som eksempel er her tre af de første funktioner er begge til opsætning af accelerometret.

### **setMeasureMode()**

```

33 void ADXL345::setMeasureMode()
34 {
35   writemem(POWER_CTL, MEASURE_MODE);
36 }

```

setMeasureMode() skriver til power control registret, POWER\_CTL. Funktionaliteten af POWER\_CTL er beskrevet i [ADXL345, 2011, s. 25--26]. POWER\_CTLs adresse er, som tidligere beskrevet, defineret i ADXL345.h. MEASUREMODE er en sekvens af otte bits (en byte) somm hvis de skrives i POWER\_CTL, vil sætte accelerometret i måle tilstand. Sekvensen er ligledes udledt af [ADXL345, 2011]. Den er i dette tilfælde 0000 1000.

### **getOutputRate()**

```

38 byte ADXL345::getOutputRate()
39 {
40   readmem(BW_RATE, 1, &_buff[0]);
41   return (_buff[0]);
42 }

```

getOutputRate() læser registret BW\_RATE, ligger dataene i en buffer og retunerer aflæsningen.

### **setFullRes()**

```

56 void ADXL345::setFullRes(bool _fullRes)
57 {

```

```

58     readmem(DATA_FORMAT, 1, &_buff[0]);
59     writemem(DATA_FORMAT, (_buff[0] & ~(1 << 3)) | (_fullRes
60         << 3));

```

`setFullRes()` skal kun ændre *en* bit i registret DATA\_FORMAT. For at bibeholde de andre bits læser den først det som står i registret og gemmer det i en buffer.  $1 << 3$  betyder en bit shiftet tre pladser til venstre: 0000 1000.  $\sim(1 << 3)$  er Bitwise NOT værdien af dette: 1111 0111. Det bliver AND med bufferen,  $(\_buff[0] \& \sim(1 << 3))$ . Det er lig med alle de bit som er sat i bufferen på nær den fjerde sidste. Altså: den sætter den fjerde sidste bit til 0 og bibeholder alle andre.  $(\_fullRes << 3)$  er værdien der skal sætte shiftet tre pladser til venstre: 0000 x000. Disse to værdier bliver til slut OR med hinanden og den fjerde sidste bit er nu ændret til værdien af `_fullRes` uden at nogle andre bits bliver ændret. Denne værdi bliver nu skrevet tilbage i samme buffer.

To andre eksempler på funktionaliteten af ADXL345.cpp. Disse funktioner har med modtagelse af data at gøre.

#### isRawDataReady()

```

75
76 bool ADXL345::isRawDataReady()
77 {
78     readmem(INT_SOURCE, 1, &_buff[0]);
79     return (_buff[0] >> 7);

```

Accelerometeret har et register med interrupt flag, INT\_SOURCE. `isRawDataReady()` læser dette register og gemmer det i en buffer. Bufferen bliver shiftet syv pladser til højre og `isRawDataReady()` returnerer en bool værdi, alt efter, om interrupt flaget er sat.

#### readAccRaw()

```

99 void ADXL345::readAccRaw(int *_AccX, int *_AccY, int *_AccZ)
100 {
101     readmem(DATAX0, 6, &_buff[0]);
102     *_AccX = _buff[1] << 8 | _buff[0];
103     *_AccY = _buff[3] << 8 | _buff[2];
104     *_AccZ = _buff[5] << 8 | _buff[4];
105 }

```

Dataen i accelerometeret fylder seks registre af 8 bit. Hver akse fylder to registre hvoraf det første er den mindst betydene. `readAccRaw()` læser DATAX0 og seks registre frem. Dataen bliver lagt i et buffer-array. Den mest betydene af de to bytes som hører sammen bliver shiftet 8 pladser til venstre og derefter AND med den mindst betydene. De to bytes bliver sat sammen direkte efter hindanden. Værdierne bliver

retuneret til tre integer-pointers.

## 2.2 Styring

Styringen af Swagwayen skulle være en længerevarende og holdbar løsning. Det ønskes ikke, at enheden efter forholdsvis kort tid skulle udskiftes eller repareres. Vi overvejede fem forskellige løsninger: En lineær potentiometer, et drejepotentiometer, gaffelsensor, strain-gate og stepper motor.

### Lineær potentiometer

Et lieært potentiometer fungerer som et normalt potentiometer, dog trækkes "armen" i en sliske. Potentiometret skal placeres i bunden af stangen, hvorefter potentiometrets arm trækkes med stangens bevægelse. Herefter vil man kunne ændre på PWM signalet til motorene, og på den måde dreje. Løsningen er simpel at udføre, og vil virke let i praksis.

Ulempen ved denne løsning er dog, at potentiometret hurtigt slides op ved brug, og det er derfor ikke en holdbar løsning.

### Dreje potentiometer

Samme princip fra det lineære potentiometer gælder her. Derimod drejes dette potentiometer i stedet for om sin egen akse. Ideen med dette var, at man let kan implementere den i toppen af Swagwayens styr, og så styre PWM signalet til motorerne sådan.

Lige som det linære potentiometer, slides denne løsning hurtigt, og er derfor ikke holdbar. Derudover er denne løsning ikke lige så intuitiv og føles klodset, hvilket ikke er det vi ønsker med produktet.

### Gaffelsensor

Gaffelsensorer benyttes i gamle kuglemuse, som vi alle nok kender. Gaffelsensorene mäter hvor meget noget har flyttet sig ved at analysere bevægelsen i forhold til nogle bånd buede bånd, i to krysende retninger. Ud fra bevægelsen kan gaffelsensoren bestemme, hvordan kuglen har bevæget sig i dens socket. En lignende anordning kunne placeres i bunden af stangen lige som det lineære potentiometer, og en kugle kunne skubbes rundt lige som i en mus, for at bestemme stangens hældningen. Løsningen er meget præcis hvis den er lavet mekanisk godt.

Gaffelsensoren er derimod ekstremt upræcis hvis mekanikken ikke er rentsåret næsten perfekt. Yderligere bliver dens præcision påvirket af støv og skidt, hvilket hurtigt vil ophobe sig i bunden af vores Swagway.

### Strain-gate

En strain-gate løsning er en anordning, som placeres på siden af stangen. Strain-gaten kan mærke tryk i metallet, og kan ud fra dette tryk bestemme en værdi, som

man kan bruge til at styre Swagwayen med. Denne løsning er ekstrem præcis, og benyttes ofte i store maskiner.

Problemet er dog, at den ikke virker på vores stang, da den er for stiv, og løsningen hverken er intuitiv eller responsiv for forbruger. Den er ikke lige til at finde ud af fra første øjekast, og den er sværere at lære.

## Stepper motor

Det sidste løsningsforslag: Stepper motor. Planen med denne var at foretage den modsatte funktion af hvad man normalt gør med en stepper motor: I stedet for at få motoren til at bevæge noget, skal styret bevæge motoren, hvorefter det måles hvilket step den er på i Arduinoen. Løsningen er ekstremt robust og præcis, som vi kender stepper motorer for. Derudover vil den ikke være modtagelig for omgivelsernes påvirkninger i form af støv eller andet snavs. Yderligere kan steppene geares, så den bliver endnu mere præcis hvis nødve +ndigt.

Det sværreste ved denne løsning er, at implementere den.

Hvis tiden havde rækket til det, havde vi derfor valgt denne som den umiddelbare løsning, da den ikke påvirkes af udefrakomne krafter, og det eneste umiddelbare problem er implementeringen.

## 3 Control

### 3.1 Hoved loop

Hoved loopet i `swagway.ino` er forholds kort fordi de fleste funktioner er flyttet til biblioteker eller metoder. Hele loopet består af tre `if()`-statements. I det første bliver accelerometeret spurgt om der er nye data klar. Hvis det er tilfældet bliver dataende hentet og `atan2()` bliver brugt til at udregne vinklen ud fra *xy*-planet. Et flag der indekerer at der er ankommet nye data fra accelerometeret sættes til `true`.

Derefter bliver gyroskopet spurgt om der er nye data klar. Hvis de er det bliver de ligeledes hentet. Data fra gyroskopets *z*-akse integreres.<sup>4</sup> Et tilsvarende flag der indekerer at der er ankommet nye data fra gyroskopet bliver sat til `true`.

```

102 void loop()
103 {
104     if (acc.isRawDataReady())
105     {
106         acc.readAccRaw(&xa,&ya,&za);
107         accAngle = atan2(xa,ya)*180/3.1415; // calcutalte the X-Y
108         -angle
109         newAccData = true;
110     }

```

---

<sup>4</sup>Integralet af gyroskobets *z*-akse bruges ikke direkte til at balancere. Det sendes kun over serielparten til debugging og tuning af Kalman-filteret.

```

111  if (gyro.isRawDataReady())
112  {
113      gyro.readGyro(&xg,&yg,&zg);
114      gyroAngle += zg/gyroSampleRate;
115      newGyroData = true;
116  }
117
118  if (newAccData && newGyroData)
119  {
120      estAngle = kalman(accAngle, zg, micros()-sinceLastSend);
121      // sendToGraph();
122      newAccData = newGyroData = false;
123      float pwm = pid(estAngle);
124      motorControl(pwm,pwm);
125      Serial.print(estAngle);
126      Serial.println(pwm);
127
128      sinceLastSend = micros();
129  }
130 }
```

Når flagende indikerer at der er nye data fra både accelerometeret og gyroskopet sendes dataende til Kalman-filteret som retunerer en estimeret vinkel. (Se afsnit 3.2 for forklaring hvordan det virker.) Vinklen sendes videre til regulering (se afsnit 3.3) som retunerer en PWM-værdi som sendes til motoren.

## 3.2 Filter

Målet med filtret er at samle dataen fra gyroskopet, accelerometeret og ud fra disse beregne en vinkel, som vi kan benytte til at regulere PWM efter. Ud fra dette mål, udvalgte vi tre filtre, som har de egenskaber vi leder efter.

### Komplementær filter

Det komplementære filter er kendtegnet ved, at det er let at sætte op, fordi det er meget intuitivt. Filtret hjælper også kraftigt på støj og drifting. Derudover er det hurtigt, hvilket betyder at Arduinoen kan beregne vinklen i nærmest real-time.

Filtret består af et lav pas filter på accelerometeret, som kun lader længerevarende ændringer igennem, hvilket fjerner de problemer accelerometeret har med bevægelses-acceleration. Gyroskopets data passerer igennem et høj pas filter, som fjerner længerevarende ændringer. De længerevarende ændringer som gyroen er udsat for er drifting. Når de to svagheder er reduceret kraftigt vha. det komplementære filter, ender man med en vinkel der ligger meget nær virkeligheden.

## Kalman filter

Kalman filtret er et professionelt filter, som benyttes i de fleste køretøjer nutildags, da dette filter er yderst præcis. Kalman filtret er desværre også ekstremt svært at forstå, men fungerer i principippet lige som det komplementære: Det fjerne støj fra de to enheder, og reducerer kraftigt deres ulighed. Kalman filtret er klart det bedste da den selv korrigerer dens gain med tiden - den lærer jo længere tid den kører. Desværre kræver Kalman filtret en ekstraordinær viden inden for matematik.

## 3.3 Regulering

For at kontrollere hvorledes Swagwayens motorer bevæger sig i forhold til vinklen, skal vi bestemme et forhold mellem vinklens hældning og PWM.

### Lineær

Den simpleste og mest basale reguleringsteknik til at holde Swagwayen i balance er et lineært filter. Her mappes PWM mellem et og treddive, hvorefter den forøges med en, hver gang vinklen stiger med en i den ene eller anden retning. Swagwayen vil derfor stige lineært i hastighed med den vinkel som den hælder med.

Den lineære regulering fungerer fint uden en fører eller med andet formål end blot at balancere.

### PID

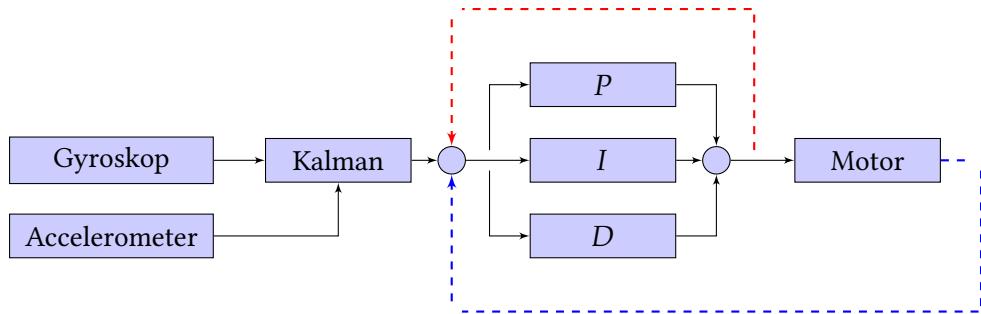
Proportional-Integral-Differential kontroller er en loop feedback mekanisme. PID kontrollerer en stor del af industrielle kontrolsystemer, da det er en feedback controller: Den beregne en "fejl" som er forskellen mellem den ønskede handling og den udførte. PID kontrolleren forsøger herefter at minimere fejlen ved at justere på kontrol input. PID forsøger derfor at opretholde eks. vinklen konstant ved det ønskede, ved hele tiden at justere sig selv.

PID er delt op i tre dele: Et proportionalt gain  $K_P$ , som ganges med fejlen i systemet. Integralet af fejlen over tid, som ganges med  $K_I$ , for at fjerne fejlen omkring nul punktet, også kaldet steady-state. Differentialet af hvor hurtig fejlen ændrer sig, ganges med  $K_D$ . Hvis der sker en hurtig ændring, bliver  $K_D$  meget stor for at kompensere.

Disse tre dele gør op for PID, som sørger for, at fejlen justeres lige meget hvilken måde den ændres.

Vi startede med at implementere PID til at regulere den, men erfarede hurtigt at det ikke virkede. For at PID virker, er det vigtigt at den fejl man beregner, stammer fra kilden. Efter som vi ikke havde encodere på motorene kunne vi ikke bestemme hvordan motorene reagerede. Så i stedet for at beregne forskellen mellem den ønskede handling og den udførte, beregnede vi forskellen mellem den ønskede handling og den handling PWM signalet sendte. Se figur 5 for en grafisk forklaring. For at PID

kontrollen skal fungere, skal der foretages målinger efter motoren, vist med den blå linje, hvorimod vi foretog målingerne før motoren, vist med den røde linje. Da tiden var ved at løbe ud, bestemte vi os for at lade PID og encoderne ligge, og gå videre til en ny løsning.



Figur 5: Motor regulering

## Eksponentiel

I stedet for PID introducerede vi en eksponentiel funktion. Den eksponentielle funktion får PWM signalet til at stige som en normal eksponentiel funktion stiger. Variablerne som funktionen er opstillet efter er fundet ud fra trial and error principippet. Nedenfor ses implementationen af reguleringen:

```

54 // PID
55
56 const int targetAngle = 0;
57 const float Ex = 2.6; //Exponential value
58 const float Kp = 2; //Proportional value

168 float pid(float input)
169 {
170     float output;
171     if (input>0)
172     {
173         output = pow(input,Ex) + Kp*input;
174     }
175     else
176     {
177         output = -pow(-input,Ex) + Kp*input;
178     }
179     return constrain(output, -255, 255);
180 }

```

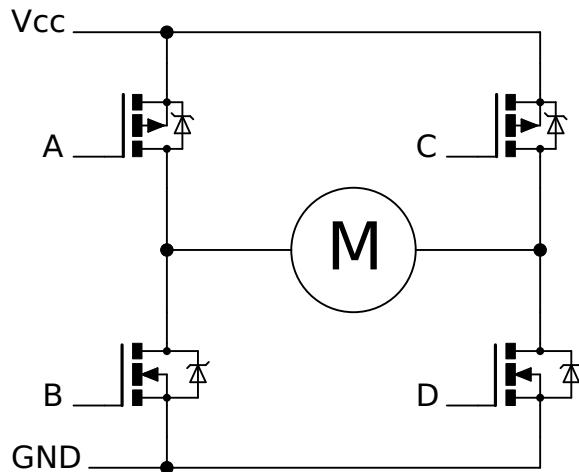
Den eksponentielle funktion er den som sidder i det nuværende produkt. Vi kan med denne funktion køre på Swagwayen uden at falde, og den kan holde sin position uden at stå og "rykke" hårdt fra side til side.

## 4 Output

For at holde Swagwayen i balance, ønsker vi, ligesom Segwayen, at motorene skal kunne køre i begge retninger med en variabel hastighed.

### 4.1 H-broens virkemåde

H-broen bruges til kontrol af motoren. Se figur 6 En H-bro er fire kontakter sat op som et H, med en motor i midten. A og B kontakterne på en side må ikke være trykkede nede samtidigt, så kører strømmen uden om motoren. Derudover må man ikke tænde for C og D for så kortslutter den. Hvis man derimod tænder for enten A og D, eller C og B, bevæger motoren sig i den retning strømmen løber. Man tænder altså derfor H-broen i par af to modsatte: A og D samtidigt, og C og B samtidigt.



Figur 6: Diagram over simpel H-bro

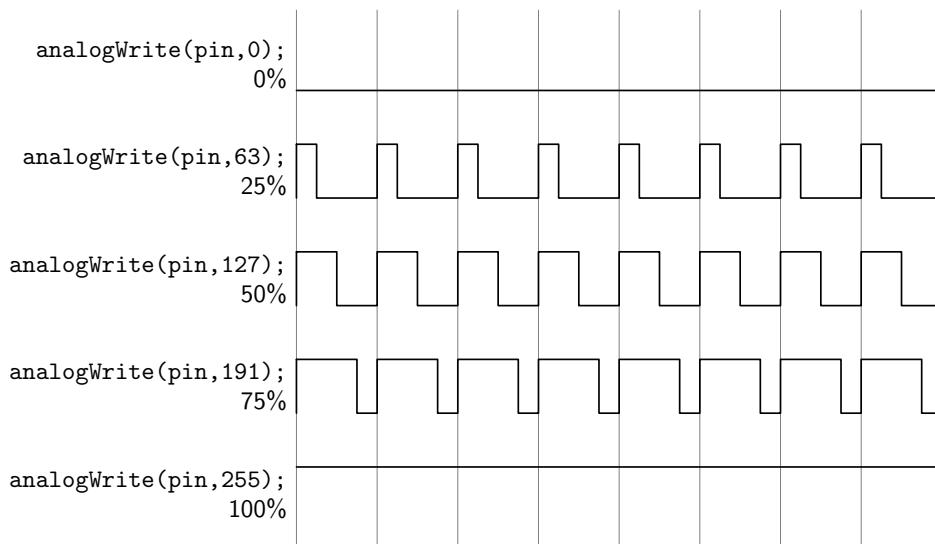
H-broen på Swagwayen er lidt mere kompleks. Hele logikken kan findes på tabel 1.

## 4.2 PWM

Pulsbreddemodulation (Pulse-Width-Modulation) er en metode til at kontrollere mængden af strøm i elektriske apparater.

PWM leverer strøm ved at sende en impuls, med variabel tid mellem 0 og 255. Jo mere tallet nærmer sig 255, jo længere tid er det elektriske apparat som er tilsluttet tændt. Se figur 7

Måden vi styre dette på i en Arduino er via `analogWrite(pin, \dots);`. Her har vi mulighed for at give en værdi fra 0--255. Dette betyder at `analogWrite(pin, 255);` er 100% og `analogWrite(pin, 127);` er 50%.



Figur 7: Eksempel på PWM med varierende dutycycle

## 4.3 Overvejelser

Der er to muligheder under fremstilling af en H-bro. Enten kan der benyttes en simpel H-bro chip, eller man kan fremstille en selv. Chippen er den nemme løsning, som gør altting super let at arbejde med. Dog kan chippen ikke håndtere store strømme, og det var derfor urealistisk at piggy-back dem.

Vi fremstillede derfor selv en H-bro, da den kan klare de store mængder af strøm vi arbejder med, selvom at det er langt mere komplekst og det forøger kraftigt mængden af ting der kan gå galt.

Efter som at motorene skal kunne køres med variable hastigheder, skal der køres PWM på disse. Man kan vælge, om dette skal ske i toppen eller bunden af H-broen som diskuteret før 4.1. Vi startede med at gøre dette i toppen, men dette blev senere ændret til bunden, hvilket er dybere beskrevet i motorcontroller v2.1. 4.4

## 4.4 Motorcontroller

Tabel 1: Motorcontroller v5.2 sandhedstabel

Arduino pin			HEXFET spænding				HEXFET on/off			
P7	P6	P5	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
P8	P9	P10	Q5	Q6	Q7	Q8	Q5	Q6	Q7	Q8
0	0	0	0	1	0	0	0	0	0	1
1	0	0	0	0	0	1	0	1	0	0
0	1	0	1	1	0	0	1	0	0	1
1	1	0	1	0	0	1	1	1	0	0
0	0	1	0	1	1	0	0	0	1	1
1	0	1	0	0	1	1	0	1	1	0
0	1	1	1	1	1	0	1	0	1	1
1	1	1	1	0	1	1	1	1	0	Short

Tabellen viser hvordan den seneste motorcontroller, v5.2, opfører sig hvis den får inputtet angivet under "Arduino Pin"

H-bro, PWM, PWM-kondensator, beskyttelses dioder, 4000 serie, optocopler

### Samlet board

Det var upraktisk at have alle funktioner på samme board. H-broerne og optocuplerne blev flyttet på sit eget board "Motorcontroller v1.0".

### Motorcontroller v1.0

24. januar 2012

Se figur 12 øverst til venstre.

### Det var der galt

Boardet virkede ikke. Det opførte sig som om det var kortsluttet. Det viste sig, at efter boardet var skilt helt af igen, at det plus tegn der skulle vise polariteten, var sat ved den forkerte pol. Printet havde taget skade af at blive loddet på flere gange.

Der var desuden nogle af ledningene for tætsiddene og loddeøerne var lidt underdimensionerede. Der manglede også en mulighed for at se, hvilken vej strømmen løber i H-broerne.

### Det blev der rettet

Plus tegnet blev flyttet over til den rigtige pol. Ledningerne blev flyttet længere væk fra hinanden, og loddeernes størrelser forøget.

**Motorcontroller v2.0**[8. marts 2012](#)**Det var der galt**

Dette board blev aldrig lavet færdigt; Ledningerne omkring pinheaderen var for tæt efter at loddeøeren blev forstørret.

**Det blev der rettet**

Loddeøerne blev gjort en smule mindre for at tillade ledningerners forbipasserend.

**Motorcontroller v2.1**[8 marts 2012](#)

Se figur 12 øverst i midten og figur 8.

**Det var der galt**

Boardet fungerede umiddelbart. Motoren kunne køre i begge retninger og farten kunne styres med PWM. Dog startede motoren på ca. 30% fart uintentionelt i den ene retning. Ved at måle på PWM signalet fra mainboardet og signalet til motoren via. et oscilloskop, kunne problemet indskrænkes til at være på Motorcontrolleren.

Det viste sig efter megen debugging, at spændingen på gaten på P-kanal HEXFETerne (IRF4905) ikke gik HIGH lige så hurtigt som ventet. Der blev opstillet et forsøg på et breadboard med en P-kanal HEXFET, en optocoupler og en Arduino, for at gennemskue problemet.

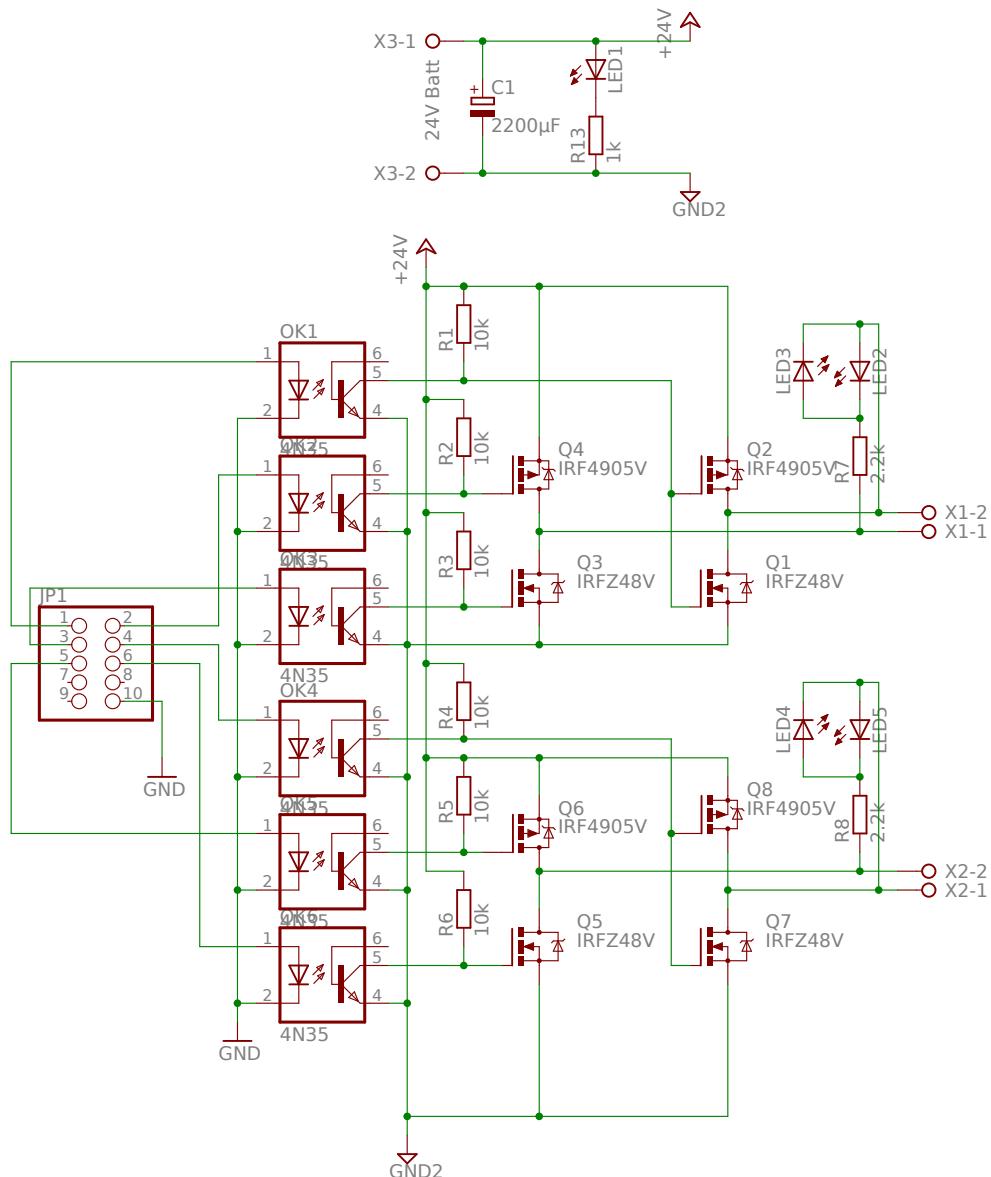
Forsøget viste, at når optocoupleren sad mellem HEXFETen og Arduinoen var der en ubekendt kapacitet mellem HEXFETens gate og source. Figur 9 viser nederst PWM signalet fra Arduinoen og øverst signalet på P-kanal HTXFETens gate, hvor man tydeligt kan se, at det tager en ubelejlig tid før signalet på gaten stiger.

Ved at sætte en mindre pull-up modstand på, kunne gaten aflades hurtigere, men det var ikke muligt at få den tilpas langt ned til at kunne styre motoren godt. Ved at fjerne optocoupleren og køre HEXFETens gate direkte fra Arduinoen eller ved fjerne HEXFETen og måle direkte på optocoupleren, var stigningstiden  $\approx 0$ . Det var kun i kombination mellem HEXFETen og transistoren i optocoupleren, at stigningstiden ikke var  $\approx 0$ .

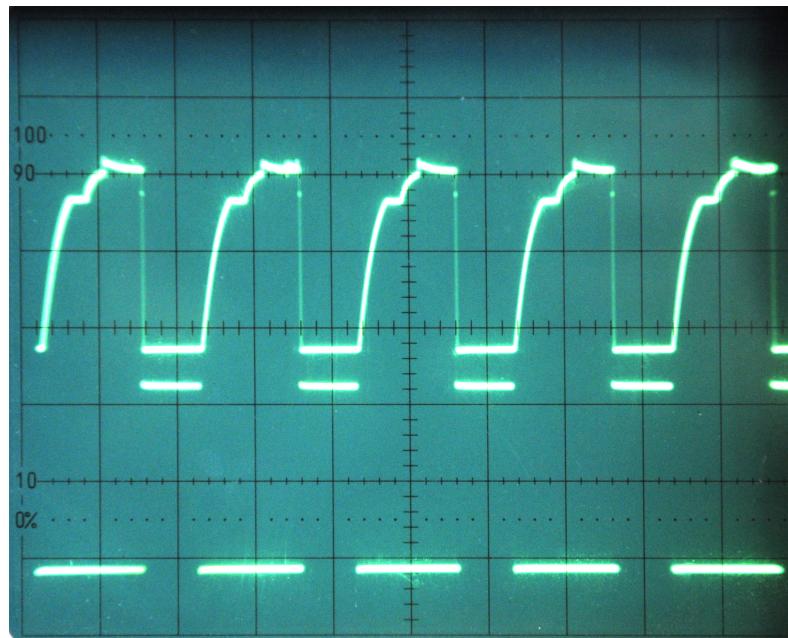
Der blev forsøgt med en 4N25 optocoupler istedet for 4N35, og en BC547 istedet for optocoupleren; der var samme stigningstid.

Det har ikke været muligt, selv med hjælp fra vejleder, at forklare hvorfor denne kapacitet er der.

Problemet blev ikke løst, det blev bare gjort ubetydeligt: Istedet for at bruge en N- og en P-kanal HEXFET til at bestemme retning og køre PWM på de andre to N- og P-kanal HEXFETter, blev det lavet om til at begge P-kanal HEXFETter blev brugt til at bestemme retning og at N-kanal HEXFETerne bliver brugt at køre PWM. Det er ikke et problem at stigetiden på P-kanal HEXFETerne er langsom da de kun ændre sig når der skiftes retning og ikke med høj frekvens som ved PWM.



Figur 8: Diagram over Motorcontroller v2.1



Figur 9: Oscilloskop billede af stigetid på en P-kanal HEXFET gate. Nederst ses PWM-signalet fra Arduinoen, øverst ses signalet på HEXFETens gate.

For ikke at tilføje flere optocouplere og bruge flere pins på Arduinoen blev der, på motorcontrolleren tilføjet to invertere.

#### **Det blev der rettet**

Vi byttede om på to P- og N-kanal HEXFETer, så der nu køres PWM på N-kanalen, og P-kanalen bestemmer retning. Derudover blev det tilføjet to invertere på motorcontrolleren.

#### **Motorcontroller v3.0**

[27. marts 2012](#)

Se figur 12 øverst til højre.

#### **Det var der galt**

Efter at der blev tilføjet en inverter på to af gatesne til P-kanal HEXFETerne er denne LOW når der ikke er spænding på optocouplerne (for eksempel når den ikke er koblet til mainboardet). Det tænder HEXFETen sammen med N-kanal HEXFETerne, som også er tændt uden spænding på optocouplerne, hvilket kortslutter H-broen. Motorcontroller v3.0 var fungerede, men det var upraktisk, at den var kortsluttet uden at være koblet sammen med mainboardet.

**Det blev der rettet**

Pull-up modstandende blev derfor erstattet med pull-down modstande.

**Motorcontroller v4.0**

[12. april 2012](#)

**Det var der galt**

Dette board blev aldrig lavet færdigt. En stor del af boardet blev re-routed da der var blevet rodet efter mange versioner.

**Det blev der rettet**

Boardet blev rerouted pga. rod.

**Motorcontroller v4.1**

[13. april 2012](#)

Se figur 12 Nederst til venstre.

**Det var der galt**

Der var en ledning der ikke var routed og der var noget mindre re-routing.

**Det blev der rettet**

Rettet op på fejl i routing

**Motorcontroller v4.2**

[17. april 2012](#)

**Det var der galt**

Dette board blev aldrig lavet færdigt. For-modstandene til optocouplerene sad på mainboardet af en ukendt årsag. Det virkede mere logisk og praktisk at have disse på motorboardet.

**Det blev der rettet**

For-modstandene til optocouplerne blev flyttet fra mainboardet til motorcontrolleren.

**Motorcontroller v5.0**

[24. april 2012](#)

**Det var der galt**

Dette board blev aldrig lavet færdigt. Det er svært at debugge motorcontrolleren når man ikke kan se hvad der sker. Optocouplerenes manglende tegn på hvad der sker skabte problemer.

### **Det blev der rettet**

Der blev tilføjet LEDer til optocouplerne.

### **Motorcontroller v5.1**

[24. april 2012](#)

#### **Det var der galt**

Dette board blev aldrig lavet færdigt. Logikken blev ændret pga. en fejl.

#### **Det blev der rettet**

Nogle ledninger i fladkablet blev byttet om.

### **Motorcontroller v5.2**

[24. april 2012](#)

Se figur 12 nederst i midten.

#### **Det var der galt**

Boardet v5.2 sidder i Swagwayen og fungerer. Den nuværende version er en erstatning af et magen til, da dette brændte af under en længerevarende, ellers succesfuld, prøvekørsel. Se figur 10.

Motorene danner strøm når de går i frigang, og hvis man derfor kører hurtigt fremad, og ændrer vinklen til et modsatgående fortegn, sendes strømmen tilbage igennem boardet. Der blev derfor tilføjet beskyttelses dioder for at tage strømmen.

#### **Det blev der rettet**

Der blev til v6.0 tilføjet beskyttelses dioder til kredsløbet, for at tage strømmen fra motorene.

### **Motorcontroller v6.0**

[24. april 2012](#)

Se figur 12 nederst til højre.

## **5 Auxiliary**

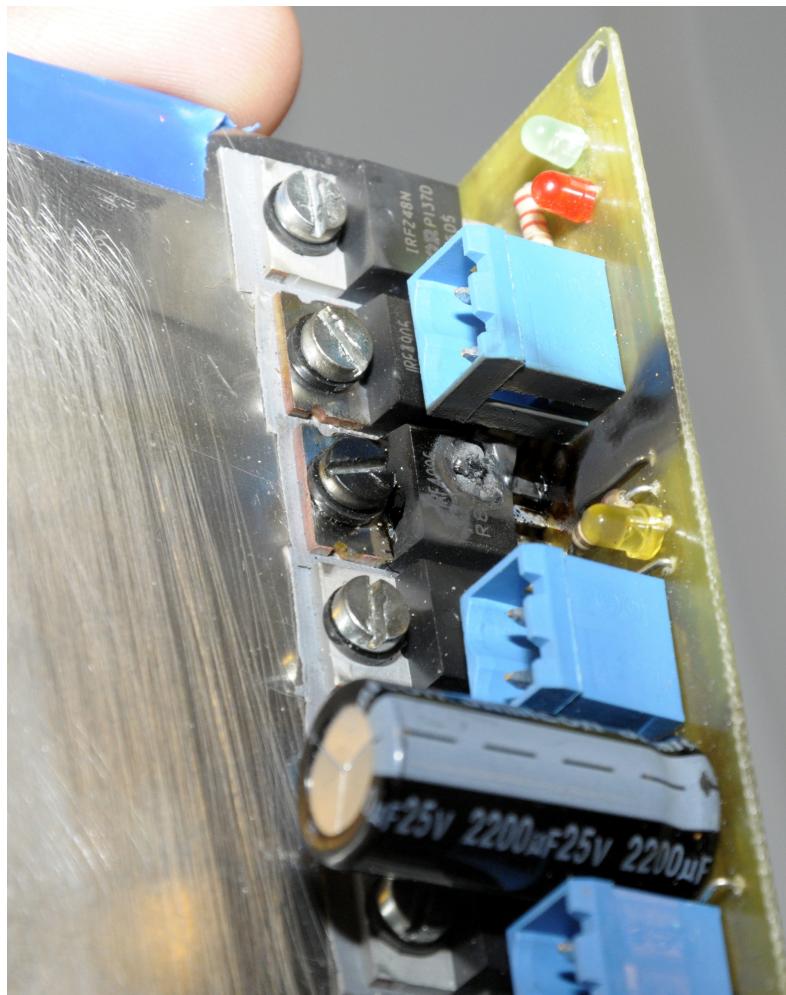
### **5.1 Mainboard**

#### **Mainboard v1.0**

[24 jan 2012](#)

#### **Det var der galt**

Mainboardets loddeører var underdimensioneret<sup>#1</sup>, hvilket betyder, at lodninger blev besværlige. Yderligere var det ikke til at komme til at trykke på resetknappen på Arduinoen, da shieldet dækkede over knappen.<sup>#2</sup>



Figur 10: En HEXFET på Motorcontroller v5.2 der er brændt af

### Det blev der rettet

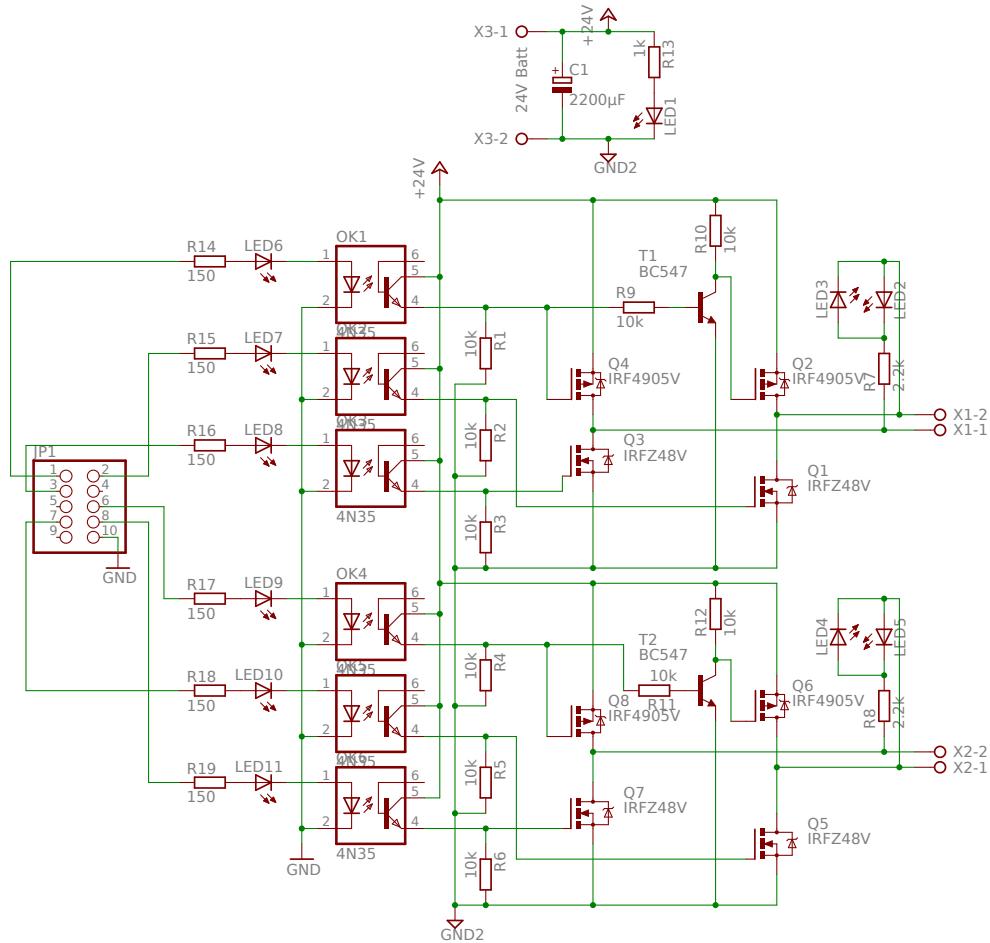
Der blev tilføjet en reset knap på mainboardet, i den følgende version. Der blev også tilføjet et stik til datamodtagning fra styringen på mainboardet.<sup>#12</sup> Derudover blev loddeørerne udvidet.

### Mainboard v2.0

1 marts 2012

### Det var der galt

Logik kredsløbet blev kasseret. Efterfølgende blev der brugt tre Arduino pins per motor.<sup>#21</sup> Display-boardet blev ligeledes kasseret.<sup>#9</sup> Pinheaderne til 9V og IMUen var desuden for tæt sammen.<sup>#13</sup>



Figur 11: Diagram over Motorcontroller v5.2

**Det blev der rettet**

Pinheaderen til 9V batteriet og IMUen blev flttet længere væk fra hinanden. Derudover blev der implementeret nyt logik.

**Mainboard v3.0**

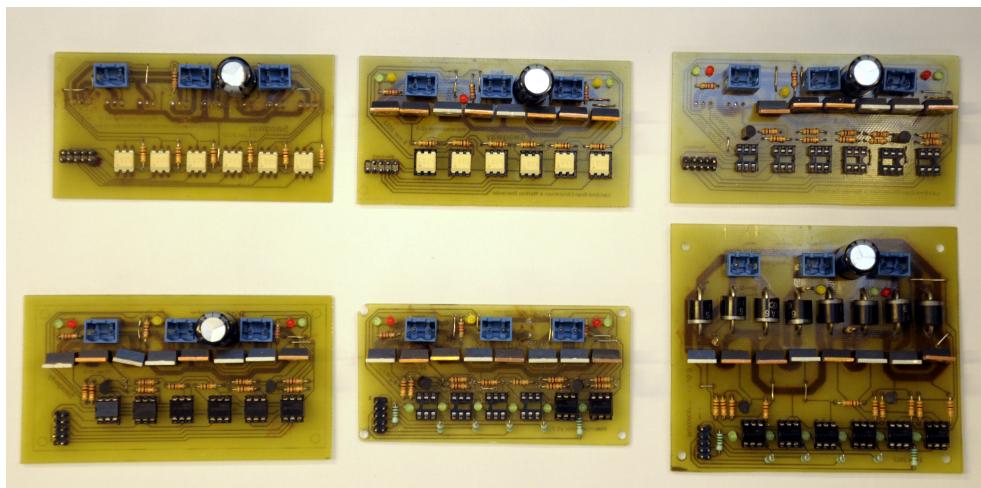
26 marts 2012

**Det var der galt**

Dette board blev aldrig lavet færdig. På boardet blev der dog tilføjet pins til en radio, hvis formål er at kunne ændre reguleringsværdierne trådløst.<sup>#29</sup>

**Det blev der rettet**

-



Figur 12: Tidlige versioner af Motorcontrolleren. Øverste række fra venstre: v1.0, v2.1, v3.0. Nederst række fra venstre: v4.1, v5.2, v6.0

Tabel 2: Pin forbindelser på Arduino

Pin	Forbindelse	Egenskaber
0	USB Rx	
1	USB Tx	
2	Radio Rx	Interrupt
3		Interrupt, PWM
4	Radio Tx	
5	Motorcontroller R2	PWM <sup>a</sup>
6	Motorcontroller L2	PWM <sup>a</sup>
7	Motorcontroller L1	
8	Motorcontroller R1	
9	Motorcontroller L3	PWM
10	Motorcontroller R3	PWM
11		PWM
12		
13		LED
A0		
A1		
A2	Steering	
A3	Steering	
A4	IMU I <sup>2</sup> C SDA	SDA
A5	IMU I <sup>2</sup> C SCL	SCL

<sup>a</sup> PWM outputtet fra disse er lidt højere end forventet. De drives af en anden timer.<sup>#42</sup> Se under Mainboard 4.0 i sektion 5.1.

**Mainboard v3.1**

29 marts 2012

**Det var der galt**

For-modstandene var placeret på mainboardet, hvilket var ulogisk.<sup>#37</sup> Pinheader til IMU passer med en  $5 \times 2$  socket, men der er placeret en  $7 \times 2$  på boardet.

**Det blev der rettet**

For-modstandene blev flyttet fra mainboardet til motorcontrolleren. Pinheaderen til IMUen blev lavet om til en  $5 \times 2$  socket.

**Mainboard v4.0**

24 april 2012

**Det var der galt**

Swagwayen kører med dette board, men den ene motor kørte hurtigere end den anden.<sup>#42</sup> Det lignede umiddelbart en mekanisk fejl, men ved at bytte om på ledningerne til motorene viste det sig at den ene kanal på motorcontrolleren kørte langsommere end den anden. Problemet blev isoleret til, at Arduinoen ikke sendte PWM med samme frekvens til begge kanaler. I Arduino referencen for `AnalogWrite()` ser man også, at pin 5 og 6 PWM kører fra en anden timer end de andre PWM pins. Tilfældigvis kørte den ene motor på begge af disse pins.

**Det blev der rettet**

Pin 5 og 9 blev byttet så Swagwayen kører lidt hurtigere forlæns end baglæns, men med samme forskel på begge hjul.

Ombytningen af de to pins blev gjort ved at bryde kobberbanerne på printet og lodde to ledninger på, der blev ikke lavet et nyt board.

## 6 Mekanik

Mekanikken i Swagwayen består af to 24V DC motorer som hver kan trække 8A. Motorene blev anskaffet fra en gammel kørestol, som havde en top fart af cirka 13 km/h. Motorene drives af to 18 Ah $\frac{1}{2}$  batterier.

## 7 Konklusion

Vores mål var at lave en balance robot, hvor vi havde tilvalgt at lave styring til begge sider, hvis tiden række til det.

Swagwayen fungerer som den står idag, og kan balancere på egen hånd. Det visste sig, at lave styring så den kunne dreje var i overkanten af hvad vi kunne nå inden

for den givne tidsramme, hvori vi dog opfyldte vores minimumskrav. Swagwayen kan også køre frem og tilbage med en fører.

## 8 Perspektivering

Som nævnt i projektbeskrivelsen, ville vi gerne med mere tid, have haft implementeret styring i form af stepper motorer til Swagwayen. Derudover ville det være interessant at have haft kraftigere motorer med planetgear for at kunne køre hurtigere og holde Swagwayen mere stabil. De store gear i de nuværende motorer gør den for tung, og inertimomentet for stort. Hvis vi fortsætter med projektet på et tidspunkt, skal der også tilføjes encodere til motorene for at kunne genimplementere PID for bedre regulering.

## Tabeller

1	Motorcontroller v5.2 sandhedstabel	22
2	Pin forbindelser på Arduino	30

## Figurer

1	Den mekaniske udformning af Swagway	7
2	Et accelerometers interne MEMS-struktur	9
3	6DOF IMU breakoutboard fra Sparkfun	10
4	Eksempel på I <sup>2</sup> C-netværk	11
5	Motor regulering	19
6	Diagram over simpel H-bro	20
7	Eksempel på PWM med varierende dutycycle	21
8	Diagram over Motorcontroller v2.1	24
9	Oscilloskop billede af stigetid på en P-kanal HEXFET gate	25
10	En HEXFET på Motorcontroller v5.2 der er brændt af	28
11	Diagram over Motorcontroller v5.2	29
12	Tidligere versioner af Motorcontrolleren	30

## Litteratur

[ADXL345, 2011] ADXL345 (2011). ADXL345. Datablad. Tilgængelig på [http://www.analog.com/static/imported-files/data\\_sheets/ADXL345.pdf](http://www.analog.com/static/imported-files/data_sheets/ADXL345.pdf)  
Hentet 11. maj 2011.

[ATMega, 2011] ATMega (2011). 8-bit Atmel Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash. Datablad. Tilgængelig på [http://www.atmel.com/dyn/resources/prod\\_documents/doc8271.pdf](http://www.atmel.com/dyn/resources/prod_documents/doc8271.pdf) Hentet 11. maj 2011.

[Espensen, 1989] Espensen, P. S. (1989). *Grundlæggende regulerings teknik*. Bogfondens forlag, 1. udgave. ISBN-10: 87-7463-192-6.

[Fraklin et al., 1998] Fraklin, G. F., Powell, J. D. og Workman, M. (1998). *Digital control of dynamic systems*. Addison-Wesley, 3. udgave. ISBN-10: 0-201-33153-5.

[ITG-3200, 2011] ITG-3200 (2011). ITG-3200 Product Specification Revision 1.7. Datablad. Tilgængelig på <http://invensense.com/mems/gyro/documents/PS-ITG-3200A.pdf> Hentet 11. maj 2011.

## A Arbejdsdeling

Indledning Fælles

Sensor teori Carl-Emil

Sensor: IMU Mathias

Sensor: I<sup>2</sup>C Mathias

Sensor: I<sup>2</sup>C biblioteker Mathias

Styring Carl-Emil

Hoved loop Mathias

Filter Carl-Emil

Regulering Carl-Emil

Output Carl-Emil

Motorcontrolere Fælles

Auxiliary Fælles

Mekanik Carl-Emil

Konklusion Fælles

Perspektivering Fælles

## B Kildekode

### B.1 swagway.ino

```
1  /*************************************************************************/
2  /* swagway.ino -- Swagway onboard software */
3  /*
4  * Author: Mathias Dannesbo <neic@neic.dk> and
5  *         Carl-Emil Grøn Christensen
6  * Time-stamp: <2012-05-07 18:07:19 (neic)>
7  * Part of the Swagway project
8  * https://github.com/neic/Swagway
9  */
10 /*****
11
12 #include <Wire.h>
13 #include <math.h>
```

```

14 #include "ITG3200.h"
15 #include "ADXL345.h"
16
17 // IMU
18 ADXL345 acc = ADXL345();
19 float accSampleRate;
20 ITG3200 gyro = ITG3200();
21 float gyroSampleRate;
22
23 // General
24 int xa, ya, za;
25 float xg, yg, zg;
26
27 unsigned long sinceLastSend;
28
29 bool newAccData, newGyroData;
30
31 double accAngle, gyroAngle, estAngle;
32
33 // Kalman filter
34 const float Q_angle = 0.001; // Process noise covariance for
      the accelerometer - Sw
35 const float Q_gyro = 0.003; // Process noise covariance for
      the gyro - Sw
36 const float R_angle = 0.03; // Measurement noise covariance -
      Sv
37
38 double angle = 0; // It starts at 0 degrees
39 double bias = 0;
40 double P_00 = 0, P_01 = 0, P_10 = 0, P_11 = 0;
41 double dt, y, S;
42 double K_0, K_1;
43
44 // Motor
45
46 const int directionPinLeft = 7; //HIGH when forward
47 const int forwardPinLeft = 9;
48 const int backwardPinLeft = 6;
49
50 const int directionPinRight = 8; //HIGH when forward
51 const int forwardPinRight = 10;
52 const int backwardPinRight = 5;
53
54 // PID
55
56 const int targetAngle = 0;
57 const float Ex = 2.6; //Exponential value
58 const float Kp = 2; //Proportional value
59

```

```

60 void setup()
61 {
62     Serial.begin(115200);
63     Wire.begin();
64
65     //Init the acc
66     acc.init(ADXL345_ADDR_SDO_LOW);
67     acc.setFullRes(true);
68     acc.setRange(3);
69     acc.setVoltage(3.3);
70     acc.setOutputRate(10); //25Hz*2^(10-8)=100Hz. See table 7
                           //in ADXL345 datasheet
71
72     //Calculate the accSampleRate
73     accSampleRate = 25*pow(2,(acc.getOutputRate()-8)); //See
                           //table 7 in ADXL345 datasheet
74
75     //Init the gyro
76     gyro.init(ITG3200_ADDR_ADO_LOW);
77     gyro.setSampleRateDiv(79); //Set the sample rate to 8000Hz
                               //(79+1)=100Hz
78
79     //Calculate the gyroSampleRate
80     if (gyro.getFilterBW() == BW256_SR8)
81     {
82         gyroSampleRate = 8000 / (gyro.getSampleRateDiv()+1);
83     }
84     else
85     {
86         gyroSampleRate = 1000 / (gyro.getSampleRateDiv()+1);
87     }
88
89     pinMode(directionPinLeft, OUTPUT);
90     pinMode(forwardPinLeft, OUTPUT);
91     pinMode(backwardPinLeft, OUTPUT);
92     pinMode(directionPinRight, OUTPUT);
93     pinMode(forwardPinRight, OUTPUT);
94     pinMode(backwardPinRight, OUTPUT);
95     //Calibration
96     gyro.zeroCalibrate(250,2);
97
98     //Dump settings
99     dumpIMUsettings();
100 }
101
102 void loop()
103 {
104     if (acc.isRawDataReady())
105     {

```

```

106     acc.readAccRaw(&xa,&ya,&za);
107     accAngle = atan2(xa,ya)*180/3.1415; // calcutalte the X-Y
108         -angle
109     newAccData = true;
110 }
111 if (gyro.isRawDataReady())
112 {
113     gyro.readGyro(&xg,&yg,&zg);
114     gyroAngle += zg/gyroSampleRate;
115     newGyroData = true;
116 }
117 if (newAccData && newGyroData)
118 {
119     estAngle = kalman(accAngle, zg, micros()-sinceLastSend);
120     // sendToGraph();
121     newAccData = newGyroData = false;
122     float pwm = pid(estAngle);
123     motorControl(pwm,pwm);
124     Serial.print(estAngle);
125     Serial.println(pwm);
126
127     sinceLastSend = micros();
128 }
129 }
130 }
131
132 double kalman(double newAngle, double newRate, double dtime
133 ) {
134     // KasBot V2 - Kalman filter module - http://www.arduino.
135         cc/cgi-bin/yabb2/YaBB.pl?num=1284738418 - http://www.x
136         -firm.com/?page_id=145
137     // with slightly modifications by Kristian Lauszus
138     // See http://academic.csuohio.edu/simond/courses/eec644/
139         kalman.pdf and http://www.cs.unc.edu/~welch/media/pdf/
140         kalman_intro.pdf for more information
141     dt = dtime / 1000000; // Convert from microseconds to
142         seconds
143
144     // Discrete Kalman filter time update equations - Time
145         Update ("Predict")
146     // Update xhat - Project the state ahead
147     angle += dt * (newRate - bias);
148
149     // Update estimation error covariance - Project the error
150         covariance ahead
151     P_00 += -dt * (P_10 + P_01) + Q_angle * dt;
152     P_01 += -dt * P_11;
153     P_10 += -dt * P_11;

```

```

146     P_11 += +Q_gyro * dt;
147
148     // Discrete Kalman filter measurement update equations -
149     // Measurement Update ("Correct")
150     // Calculate Kalman gain - Compute the Kalman gain
151     S = P_00 + R_angle;
152     K_0 = P_00 / S;
153     K_1 = P_10 / S;
154
155     // Calculate angle and resting rate - Update estimate
156     // with measurement zk
157     y = newAngle - angle;
158     angle += K_0 * y;
159     bias += K_1 * y;
160
161     // Calculate estimation error covariance - Update the
162     // error covariance
163     P_00 -= K_0 * P_00;
164     P_01 -= K_0 * P_01;
165     P_10 -= K_1 * P_00;
166     P_11 -= K_1 * P_01;
167
168     return angle;
169 }
170
171 float pid(float input)
172 {
173     float output;
174     if (input>0)
175     {
176         output = pow(input,Ex) + Kp*input;
177     }
178     else
179     {
180         output = -pow(-input,Ex) + Kp*input;
181     }
182     return constrain(output, -255, 255);
183 }
184
185 void motorControl(int left, int right)
186 {
187     if (left < 0)
188     {
189         digitalWrite(directionPinLeft, HIGH);
190         digitalWrite(backwardPinLeft, LOW);
191         analogWrite(forwardPinLeft, -left);
192     }
193     else
194     {

```

```

192     digitalWrite(directionPinLeft, LOW);
193     digitalWrite(forwardPinLeft, LOW);
194     analogWrite(backwardPinLeft, left);
195 }
196 if (right < 0)
197 {
198     digitalWrite(directionPinRight, HIGH);
199     digitalWrite(backwardPinRight, LOW);
200     analogWrite(forwardPinRight, -right);
201 }
202 else
203 {
204     digitalWrite(directionPinRight, LOW);
205     digitalWrite(forwardPinRight, LOW);
206     analogWrite(backwardPinRight, right);
207 }
208 }
209 /* Serial communication */
210
211 void sendToGraph()
212 {
213     Serial.print("<");
214     Serial.print(gyroAngle); //0
215     Serial.print(",");
216     Serial.print(accAngle); //1
217     Serial.print(",");
218     Serial.print(estAngle); //2
219     Serial.print(",");
220     Serial.print(micros()-sinceLastSend); //3
221     Serial.println(">");
222 }
223
224 void dumpIMUsettings()
225 {
226     Serial.println();
227     Serial.println("=====IMU Settings=====");
228     Serial.println();
229     Serial.println("          ---Gyro---");
230     Serial.print("Sample rate (Hz) = ");
231     Serial.println(gyroSampleRate,0);
232     Serial.println();
233     Serial.println("          ---Acc---");
234     Serial.print("Sample rate (Hz) = ");
235     Serial.println(accSampleRate,0);
236     Serial.print("Full resolution = ");
237     Serial.println(acc.getFullRes());
238     Serial.print("Range (g) = ");
239     Serial.println(pow(2,(1+acc.getRange())),0);
240 }
```

```

241   Serial.print("Scale factor X           (LBS/g) = ");
242   Serial.println(acc.scaleFactor[0],0);
243   Serial.print("Scale factor Y           (LBS/g) = ");
244   Serial.println(acc.scaleFactor[1],0);
245   Serial.print("Scale factor Z           (LBS/g) = ");
246   Serial.println(acc.scaleFactor[2],0);
247   Serial.println();
248   Serial.println("=====end IMU Settings=====");
249   Serial.println("=====");
250   Serial.println();
251 }
```

## B.2 ADXL345.h

```

1  ****
2  /* ADXL345.h -- ADXL345/I2C library for Arduino */
3  /*
4  * Author: Mathias Dannesbo <neic@neic.dk> and
5  *         Carl-Emil Grøn Christensen
6  * Time-stamp: <2012-04-04 17:37:05 (neic)>
7  * Part of the Swagway project
8  * https://github.com/neic/Swagway
9  */
10 /* Inspired by the ITG3200 Arduino library at
11 /* http://code.google.com/p/itg-3200driver
12 ****
13
14 #ifndef ADXL345_h
15 #define ADXL345_h
16
17 #if defined(ARDUINO) && ARDUINO >= 100
18 #include "Arduino.h"
19 #else
20 #include "WProgram.h"
21 #endif
22
23 #define ADXL345_ADDR_SDO_HIGH 0x1D
24 #define ADXL345_ADDR_SDO_LOW 0x53
25
26 // Registers
27 #define BW_RATE          0x2C // RW SETUP: Output rate and low
28 // power mode
29 #define POWER_CTL        0x2D // RW SETUP: Power control
30 #define INT_SOURCE       0x30 // R INTERRUPT: Status
31 #define DATA_FORMAT      0x31 // RW SETUP: Self-test and data
32 // format
33 #define DATA_X0          0x32 // R SENSOR: Data
34
35 // Bitmaps
```

```

34 #define STANDBY_MODE    0x00 // 0000 0000
35 #define MEASURE_MODE    0x08 // 0000 1000
36
37 class ADXL345
38 {
39     public:
40         float scaleFactor[3];
41         float voltage;
42
43     ADXL345();
44
45     void init(unsigned int address);
46
47     // SETUP: Mode
48     void setStandbyMode();
49     void setMeasureMode();
50
51     // SETUP: Output Rate
52     byte getOutputRate();
53     void setOutputRate(byte _SampleRate);
54
55     // SETUP: Data format
56     bool getFullRes();
57     void setFullRes(bool fullRes);
58     int getRange();
59     void setRange(int range);
60
61     // INTERRUPT
62     bool isRawDataReady();
63
64     // SETUP: Data processing
65     void setVoltage(float _voltage);
66     void updateScaleFactor();
67
68     // SENSOR: Read
69     void readAccRaw(int *_AccX, int *_AccY, int *_AccZ);
70     void readAcc(float *_AccX, float *_AccY, float *_AccZ);
71
72     void writemem(uint8_t _addr, uint8_t _val);
73     void readmem(uint8_t _addr, uint8_t _nbytes, uint8_t __buff
74         []);
75
76     private:
77         uint8_t _dev_address;
78         uint8_t _buff[6];
79     };
80 #endif /* ADXL345_h */

```

### B.3 ADXL345.cpp

```

1  /*************************************************************************/
2  /* ADXL345.cpp -- ADXL345/I2C library for Arduino */
3  /* */                                                 */
4  /* Author: Mathias Dannesbo <neic@neic.dk> and      */
5  /*          Carl-Emil Grøn Christensen                */
6  /* Time-stamp: <2012-05-11 04:52:13 (neic)>        */
7  /* Part of the Swagway project                      */
8  /* https://github.com/neic/Swagway                  */
9  /* */                                                 */
10 /* Inspired by the ITG3200 Arduino library at       */
11 /* http://code.google.com/p/itg-3200driver           */
12 /* ************************************************************************/
13
14 #include "ADXL345.h"
15 #include <Wire.h>
16
17 ADXL345::ADXL345()
18 {
19 }
20
21 void ADXL345::init(unsigned int address)
22 {
23     _dev_address = address;
24     setStandbyMode();
25     setMeasureMode();
26 }
27
28 void ADXL345::setStandbyMode()
29 {
30     writemem(POWER_CTL, STANDBY_MODE);
31 }
32
33 void ADXL345::setMeasureMode()
34 {
35     writemem(POWER_CTL, MEASURE_MODE);
36 }
37
38 byte ADXL345::getOutputRate()
39 {
40     readmem(BW_RATE, 1, &_buff[0]);
41     return(_buff[0]);
42 }
43
44 void ADXL345::setOutputRate(byte _rate)
45 {
46     _rate %= 16; //Prevent overflow
47     writemem(BW_RATE, _rate);

```

```

48 }
49
50 bool ADXL345::getFullRes()
51 {
52     readmem(DATA_FORMAT, 1, &_buff[0]);
53     return (_buff[0] >> 3);
54 }
55
56 void ADXL345::setFullRes(bool _fullRes)
57 {
58     readmem(DATA_FORMAT, 1, &_buff[0]);
59     writemem(DATA_FORMAT, ((_buff[0] & ~(1 << 3)) | (_fullRes
60         << 3)));
61
62 int ADXL345::getRange()
63 {
64     readmem(DATA_FORMAT, 1, &_buff[0]);
65     return (_buff[0] & B00000011);
66 }
67
68 void ADXL345::setRange(int range)
69 {
70     range %= 4; //Prevent overflow
71     readmem(DATA_FORMAT, 1, &_buff[0]);
72     writemem(DATA_FORMAT, ((_buff[0] & ~3) | range));
73 }
74
75
76 bool ADXL345::isRawDataReady()
77 {
78     readmem(INT_SOURCE, 1, &_buff[0]);
79     return (_buff[0] >> 7);
80 }
81 void ADXL345::setVoltage(float _voltage)
82 {
83     voltage = _voltage;
84     updateScaleFactor();
85 }
86
87 void ADXL345::updateScaleFactor()
88 {
89     int rangeScale=256;
90     if (!getFullRes())
91     {
92         rangeScale = pow(2,(8-getRange()));
93     }
94     scaleFactor[0] = rangeScale*0.89013671875+rangeScale
95         *0.0439453125*voltage;

```

```

95     scaleFactor[1] = rangeScale*0.89013671875+rangeScale
96         *0.0439453125*voltage;
97     scaleFactor[2] = rangeScale;
98 }
99 void ADXL345::readAccRaw(int *_AccX, int *_AccY, int *_AccZ)
100 {
101     readmem(DATAX0, 6, &_buff[0]);
102     *_AccX = _buff[1] << 8 | _buff[0];
103     *_AccY = _buff[3] << 8 | _buff[2];
104     *_AccZ = _buff[5] << 8 | _buff[4];
105 }
106
107 void ADXL345::readAcc(float *_AccX, float *_AccY, float *
108     _AccZ)
109 {
110     int x, y, z;
111     readAccRaw(&x,&y,&z);
112     *_AccX = x / scaleFactor[0];
113     *_AccY = y / scaleFactor[1];
114     *_AccZ = z / scaleFactor[2];
115 }
116 void ADXL345::writemem(uint8_t _addr, uint8_t _val) {
117     Wire.beginTransmission(_dev_address); // start
118         transmission to device
119     Wire.write(_addr); // send register address
120     Wire.write(_val); // send value to write
121     Wire.endTransmission(); // end transmission
122 }
123 void ADXL345::readmem(uint8_t _addr, uint8_t _nbytes, uint8_t
124     _buff[]) {
125     Wire.beginTransmission(_dev_address); // start transmission
126         to device
127     Wire.write(_addr); // sends register address to read from
128     Wire.endTransmission(); // end transmission
129
130     Wire.beginTransmission(_dev_address); // start transmission
131         to device
132     Wire.requestFrom(_dev_address, _nbytes); // send data n-
133         bytes read
134     uint8_t i = 0;
135     while (Wire.available()) {
136         _buff[i] = Wire.read(); // receive DATA
137         i++;
138     }
139     Wire.endTransmission(); // end transmission
140 }
```