Hastighedscontroller til DC motor

Mini-projekt Design af indlejret software

Jakob Thiesen, Joachim Andersen, Kasper Rødtnes

Elektronik og IT, 4. Semester

Indhold

Kapitel 1

Design af Hastighedscontroller

1.1 Introduktion til opgaven

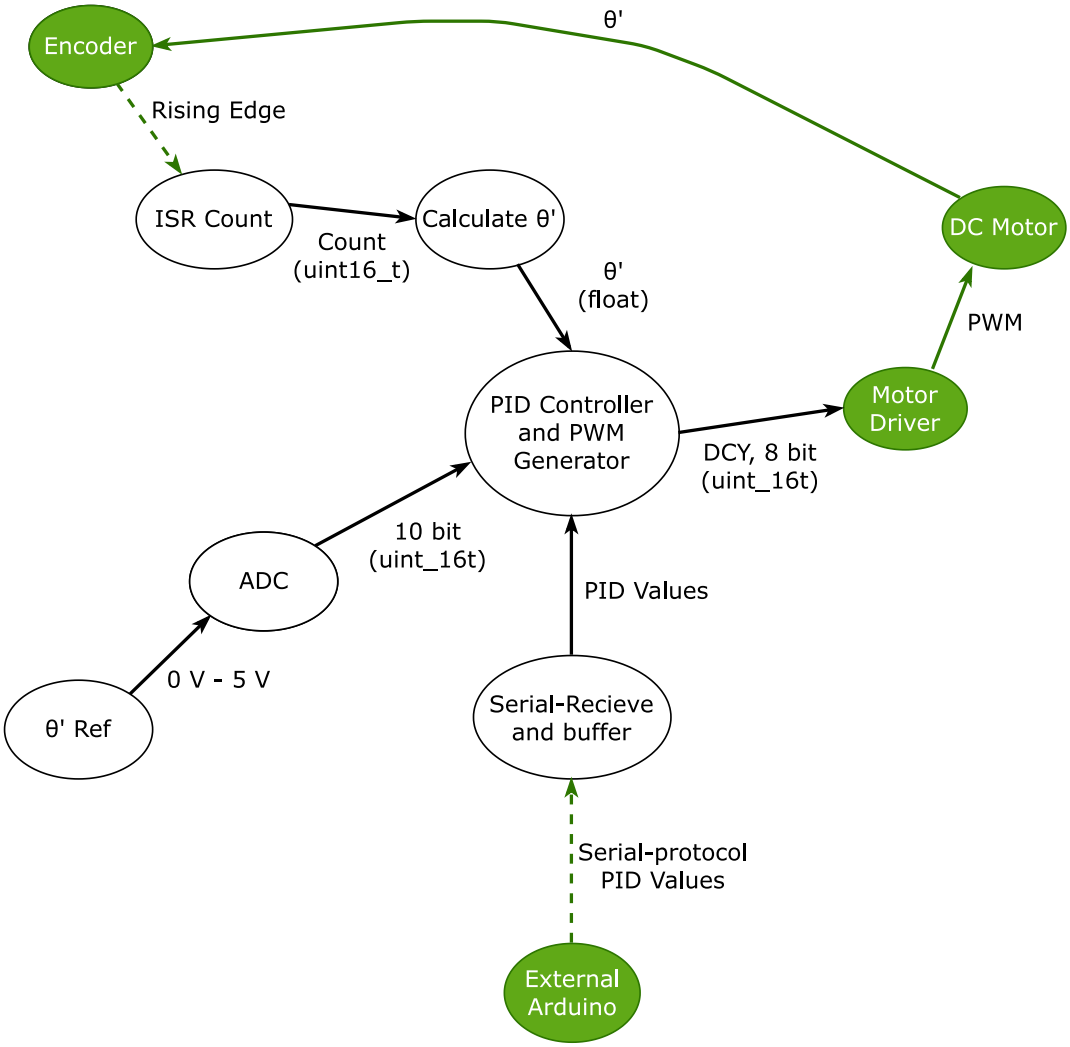
Formålet med dette miniprojekt er at designe en hastighedscontroller til en DC motor. Der er fra workshoppen hvor problemstillingen stammer fra stillet følgende krav til hastighedskontrolleren

- Hastighedscontrolleren skal implementeres på en Arduino UNO.
- Følgende opgaver skal udføres ved fast samplingsfrekvens:
  - Controlleren skal læse hastighed fra motorens quadratur-encoder-
  - Controlleren skal styre hastigheden gennem PWM udgang og og digital pin til omdrejningsretning.
  - Controlleren skal læse hastigheds reference fra analog indgang med potentiometer tilkoblet
  - Cotrolleren skal læse P,I,D parametre fra seriel port, en passende seriel protokol med delimiter-tegn skal designes
- Designet skal overholde følgende:
  - Controlleren skal designes med operativsystem, med tasks til opgaver systemet skal udføre
  - Der skal være kommunikation mellem tasks, eksempelvis via et beskedbuffer system

Designet implementeres vha. FreeRTOS på Arduino UNO, baseret på at dette til en vis grad er industristandard på microcontroller systemer.

1.2 Design struktur

Ud fra de stillede krav i opgaven, er der udviklet et data-flow diagram. Dette er gjort med henblik på at skabe både overblik og grænseflader mellem de forskellige opgaver der skal køre. Dette diagram kan ses i figure 1.1.



Figur 1.1: Data-flow diagram over PID controlleren, grøn indikere ting der sker uden for systemet.

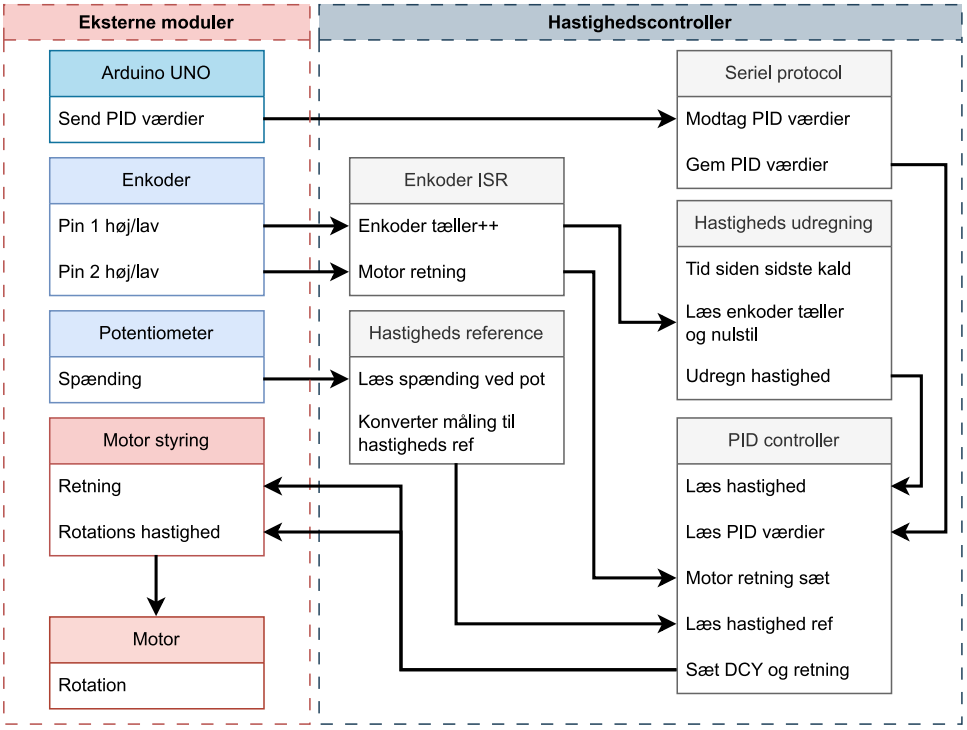
For at skabe nogle stramme og veldefinerede rammer som de forskellige opgaver kommunikere igennem, mere specifikt den data der sendes fra en opgave til en anden, er de forskellige værdier imellem opgaverne defineret. Dette kan ses mere specifikt i afsnit 1.3.1.

1.3 Modularisering

For at opfylde kravene stillet til hastighedskontrolleren i afsnit 1.1 modulariseres systemet således:

- Encoder ISR
- Hastigheds udregning
- Hastigheds ref
- Serial protocol
- PID/PWM controller

Dette er udviklet på bagrund af det data-flow diagram som ses i afsnit 1.2. De fem modulers/tasks relationer til hinanden kan ses på figur 1.2



Figur 1.2: Block diagram over modulerne i hastighedscontrolleren

På figur 1.2 er hele systemet delt op i to kategorier; de eksterne og interne moduler, for at tydeliggøre hvilke moduler tilhører hastighedscontrolleren. For hvert modul er deres "pligter"tilskrevet under titlen for at tydeliggøre deres opgaver.

1.3.1 Data struktur

For at undgå at mere end en funktion forsøger at skrive til den samme variabel, implementeres der en datastruktur således at alle variabler er beskyttet af en semaphore. Dette er gjort i form af en struct som indeholder alle data som bliver modificeret af forskellige opgaver vist i udsnit 1.1.

```
24 struct ctrl {
25   float speed;
26   uint16_t speedRef;
27   int motorDCY;
28   float p, i, d;
29 } data;
```

Udsnit 1.1: Opsætning af data struct

Data structen er som nævnt beskyttet af en semafor, semaforen benyttes af funktionen UPDStruct, der kan ses i udsnit 1.2, som anvendes hvergang der ændres en værdi som andre tasks har adgang til.

```
376
377 void UPDStruct(ctrl updStruct, uint8_t index) {
378
379   if (xSemaphoreTake(mutexStruct, 10) == pdTRUE) {
380     switch (index) {
381       case speed_:
382         data.speed = updStruct.speed;
383         break;
384       case speedRef_:
385         data.speedRef = updStruct.speedRef;
386         break;
387       case motorDCY_:
388         data.motorDCY = updStruct.motorDCY;
389         break;
390       case p_:
391         data.p = updStruct.p;
392         break;
393       case i_:
394         data.i = updStruct.i;
395         break;
396       case d_:
397         data.d = updStruct.d;
398         break;
399       case p_ + i_ + d_:
```

```
400             data.p = updStruct.p;
401             data.i = updStruct.i;
402             data.d = updStruct.d;
403             break;
404
405           default:
406             data = updStruct;
407             break;
408       }
409
410       xSemaphoreGive(mutexStruct);
411     }
412 }
```

Udsnit 1.2: Function til opdatering af data i data structen

Funktionen tjekker om semaforen "mutexStruct"er i brug, hvis den er kan funktionen ikke køre, hvis semaforen ikke er i brug kører funktionen og giver som det sidste igen semaforen fri. Denne opstilling gør effektivt at der aldrig, er to tasks som kan ædnre variable i data structen.

1.3.2 Enkoder måling

For at der ikke er pulser fra enkoderen som overses anvendes en interupt service rutine (ISR). Initialiseringen af rutinen, pin opsætning og EncoderISR er vist i udsnit 1.3

```
1  #define isrPin 2  //interupt pin
2  #define encPin 3  //encoder direction determination pin
3
4  uint32_t encoderCount;
5  bool motorDirection;
6
7  void setup() {
8      pinMode(isrPin, INPUT);
9      pinMode(encPin, INPUT);
10     attachInterrupt(
11     digitalPinToInterrupt(isrPin), // Pin to attach ISR to
12     EncoderISR, // Name of function to call
13     RISING // ISR is called on rising edge
14     );
15 }
```

Udsnit 1.3: "Minimal working example"af enkoder ISR

Som det fremgår af udsnittet er mængden af tid EncoderISR kører forsøgt minimeret ved ikke at lave hastighedsudregningen i rutinen.

1.3.3 PID

Til at lave PID kontrollen, er der anvendt et eksternt bibliotek, grundet at opgaven ikke lyder på at designe et PID system, men en motor kontrol som et real-tids system.
funktionen heder "TSKPIDControl"og er bygget op sådan at, den initialisere en lokal variabel, af typen ctrl, samme type som den struct alle funktioner bruger til at sende og hente data med. Denne struct kopier så alle værdier fra data, som er den globale struct, som alle funktioner kan hente fra og skrive til. Dette er grundet, at når funktionen bruger den lokale variable, så er det sikkert at alt data, er hentet fra samme tid, og der ikke er en anden funktion der har opdateret noget imens den bruges.
Herefter udregnes P, I og D værdier, og det tjekkes om motoren rotere den korrekte retning. Gør den ikke, beder PID funktionen driveren om at få motoren til at rotere i den modsatte retning.
Til slut opdateres den globale variable "data"med den nye udregnede duty-cycle værdi til motoren.
En tidsmåling af hele PID funktionen er fortaget som et gennemsnit af 10000 iterationer, og fundet til omkring 90 µs.

1.3.4 Seriel kommunikation

En af opgaverne til miniprojektet går ud på at der kan sendes PID værdier fra en mikrokontroller til en anden gennem serial kommunikation. Vi skal selv finde på en "protokol", eller pakkestruktur til PID værdierne. Vi har forbundet to Arduino Uno'er og vil benytte deres UART til serial kommunikation.

Datapakken med PID værdier ser således ud

```
char txMessage[] = "$P %3 |I %8 |D %2#";
```

Her svarer **$** og **#** til markører der angiver, hvor pakken henholdsvist starter og slutter. P, I og D er også markører der bruges når datapakken skal pakkes ud igen. Et P, I eller D efterfulgt af markører % angiver hvor tal-værdien starter, mens | angiver hvor tal-værdier slutter. Når der modtages en pakke over UART, så skal modtageren dekode den her pakke, dvs skille den ad, i de forskellige P, I og D værdier. Koden understøtter lige nu kun heltal mellem 0 og 9 som talværdier.

Koden fungerer på den måde, at tasken til Serial kommunikationen poller serial bufferen. Set i udsnit 1.4

```
348     if (Serial.available() > 0) {
349       incomingData = GetSerialData();
350     }
```

Udsnit 1.4: Polling af serial buffer

Lige så snart der står noget i serial bufferen, så begynder GetSerialData() at hente det. GetSerialData bliver ved med at læse buffren indtil den har fundet datapakkens slut karakter (#). Hver eneste karakter til og med slut karakteren gemmes i et array. Som kan ses på udsnit 1.5.

```
236 serialData GetSerialData() {
237   serialData incomingData;
238   uint8_t moreData = 0, currentChar, terminator = '#', index
        = 0;
239   while ((Serial.available() > 0) && (moreData == 0)) {
240     currentChar = Serial.read();
241     //Serial.print(currentChar);
242     if (currentChar != terminator) {
243       incomingData.serialData[index] = currentChar;
244       index++;
245     } else {
246       incomingData.serialData[index] = '#';
247       incomingData.serialDataArraySize = index;
248       index = 0;
249       moreData = 1;
250     }
251   }
252   return incomingData;
253 }
```

Udsnit 1.5: Definition af GetSerialData funktion

Når slut karakteren er fundet, så returneres der en struct som indeholder et array med den komplette pakke, men også længden på pakken.
Det næste der skal ske er at den modtagne pakke skal dekodes. Det gøres ved at sende den returnerede struct fra GetSerialData() igennem en anden funktion GetPIDValues() som bruges til at finde de individuelle PID værdier. Denne funktion kaldes ud i taskens hoved loop.

```
uint8_t *receivedValues = GetPIDValues(&incomingData);
```

Funktionen GetPIDValues, på udsnit 1.6, returnerer en pointer til et array som indeholder de forskellige PID værdier.

```
255 uint8_t *GetPIDValues(serialData *incomingData) {
256   static uint8_t values[3];
257   values[0] = GetPVal(incomingData);
258   values[1] = GetIVal(incomingData);
259   values[2] = GetDVal(incomingData);
260   return &values[0];
261 }
```

Udsnit 1.6: Definition af GetPIDValues funktion

Måden GetPVal(), GetIVal() og GetDVal() fungerer på er fuldstændig identisk, så vi kigger kun på GetPVal() i udsnit 1.7.

```
263 uint8_t GetPVal(serialData *incomingData) {
264   uint8_t pVal = 0, currentChar = 0, index = 0, pFound = 0;
265
266   while (incomingData->serialData[index] != '|' && (index < (
        uint8_t)sizeof(incomingData->serialData)) && pFound ==
        0) {
267     currentChar = (char)incomingData->serialData[index];
268     if (currentChar != 'P') {
269       index++;
270     } else {
271       do {
272         currentChar = (char)incomingData->serialData[index];
273         if ((char)currentChar == '%' && pFound == 0) {
274           index++;
275           pVal = (uint8_t)incomingData->serialData[index] - '
                0';  //Convert to int
276           pFound = 1;
277         } else {
278           index++;
279         }
280       } while (pFound == 0);
281     }
282   }
283   return pVal;
284 }
```

Udsnit 1.7: Definition af GetPVal() funktion

For at finde P-værdien, så scanner funktionen først den modtagne besked indtil den finder P karakteren. Når den har fundet P så scanner den videre til den finder start karakteren for tal-værdien (%). Når den har fundet denne værdi så ved

vi at den efterfølgende karakter er den tilhørende tal-værdi til P-værdien. Måden dette er lavet på gør at der kun kan findes en enkelt (1) karakter.

Det er en "char"der modtages. Det betyder at f.eks en ASCII karakter "9"har værdien 57. For at få det lavet om til alm. heltal så trækker vi ASCII værdien "0"fra denne da denne har værdien 48. Så hvis vi modtog "9"så laver vi det om til heltallet 9 ved at side $'9' - '0' = 57 - 48 = 9$

Det samme gentages for I og D værdierne i GetPIDValues(). Når alle 3 værdier er fundet, så returneres en pointer til et array, hvor P, I og D værdierne så kan aflæses. Når værdierne er aflæst så sendes de gennem UPDStruct(), set på udsnit 1.8, som er semaphor beskyttet, der så opdaterer en global struct med de her P, I, D værdier som så kan bruges af de andre tasks.

```
339 ||  UPDStruct(newPIDVals, (p_ + i_ + d_));
```

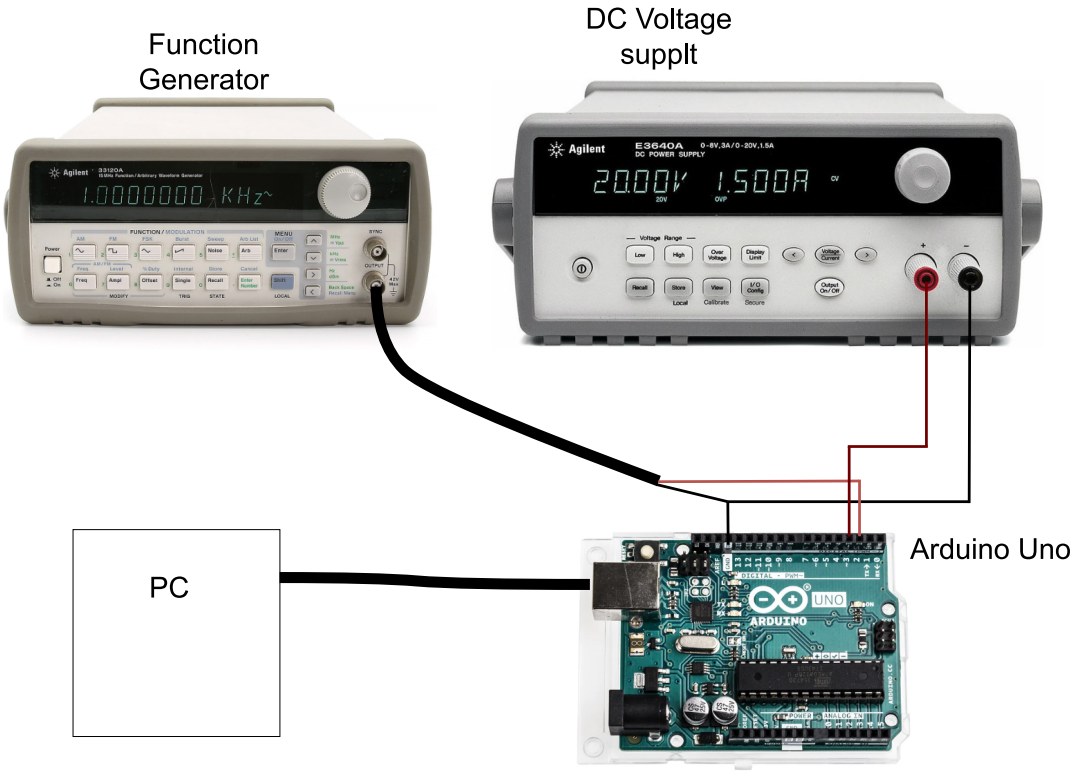
Udsnit 1.8: Opdatering af data struct

1.3.5 Test protokol

For at teste, og vurdere om de forskellige opgaver, også kendt som tasks, fungere som forventet, er det udtænkt forskellige tests af dem.

Hastigshedmåling

For at teste om funktionen til at måle antal enkoder pulser samt retning, og derfra udregne omdrejninger fungere som forventet er test opstillingen vist på figur 1.3 blevet lavet. Her forbindes en funktions-generator til pin 2 på arduinoen, generatoren sættes til "square wave"og "high level"til 5 V og "low level"til 0 V. "rise time"og "fall time"sættes til mindst mulig værdi. På denne måde kan enkoderen simuleres. For at simulere retningen af enkoderen bruges en DC power supply, forbundet til pin 3 på arduinoen. Den sættes til 5 V, ikke 20 som angivet på figuren. For at teste den ene retning sættes forsyningen til 5 V, således at når en rising edge kommer fra generatoren er pin 3 høj, altså 5 V, og den anden retning simuleres så ved at sætte forsyningen til 0 V.



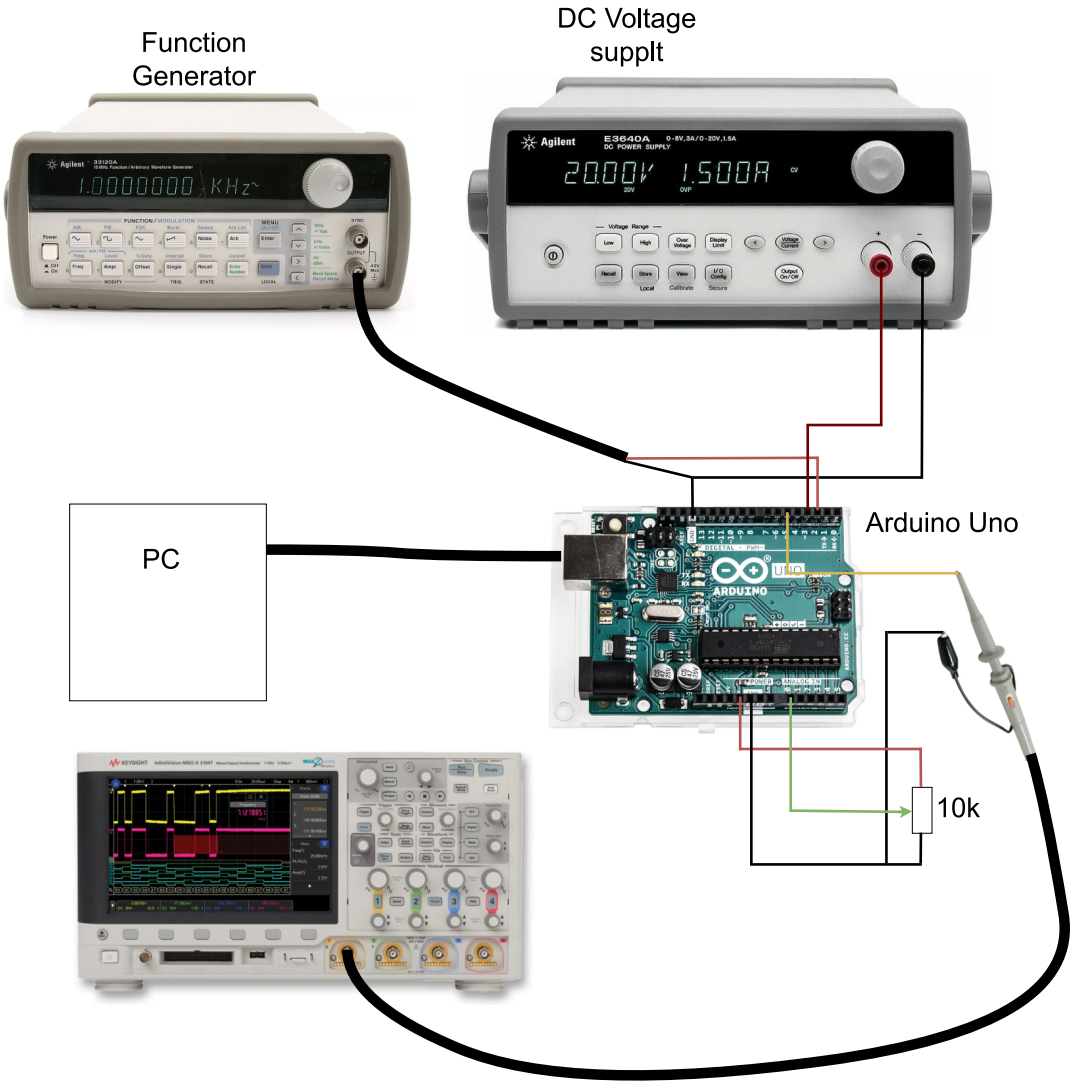
Figur 1.3: Test af funktion til at måle enkoder pulser.

Generatoren sættes til 100 Hz, 200 Hz og 500 Hz. Det indsættes i koden på arduinoen at den skal printe antal pulser siden sidst. Arduinoen kører TSKSpeedCalculation for hver 100 ms, og derfor forventes det at der ved 100 Hz tælles 10 pulser, 200 Hz tælles 20 pulser og ved 500 Hz tælles 50 pulser.
Testen er udført og det er fundet at antal pulser er som angivet i følgende lise 1.3.5. Disse resultater er fundet til at være acceptable. Det kan dog ses, at der måles mere korrekt ved højere antal pulser pr. sekund.

- 100 Hz input, målt antal pulser mellem 9 og 11.
- 200 Hz input, målt antal pulser mellem 19 og 21.
- 500 Hz input, målt antal pulser mellem 49 og 51.

PID test

Denne test er lavet således at, reference funktion testes og PID og PWM funktion testes. Samme optiling som i sektion 1.3.5. Her tilføjes der dog et potentiometer til reference værdi, og et oscilloskop til at måle PWM output fra PID funktionen. opstillingen kan ses på figur 1.4.



Figur 1.4: Test af PID og ref funktion.

et 10 k ohm potentiometer sættes således at det miderste ben går til pin A0 på arduinoen, og det forsynes med 5V på den ene side, og GND på den anden side. oscilloscopet forbindes til pin 5 på arduinoen.
Koden ændres således at arduinoen laver en "Serial.Print"af reference værdien. Potentiometeret indstilles så til 8 bit værdier på 50, 100, 150 og 200. Det forventes således så at, ved at indstille generatoren til at generere samme antal pulser i sekundet som ref værdien, at PWM outputtet ikke skal ændre sig.
således at, ref sættes til 50, generator sættes til 50 Hz. Output skal så være en duty-cycle på 50/255 også kendt som 19.6 % Det vil også forventes at ændringer på generatorens frekvens vil få denne duty-cycle til at ændre sig, således at hvis

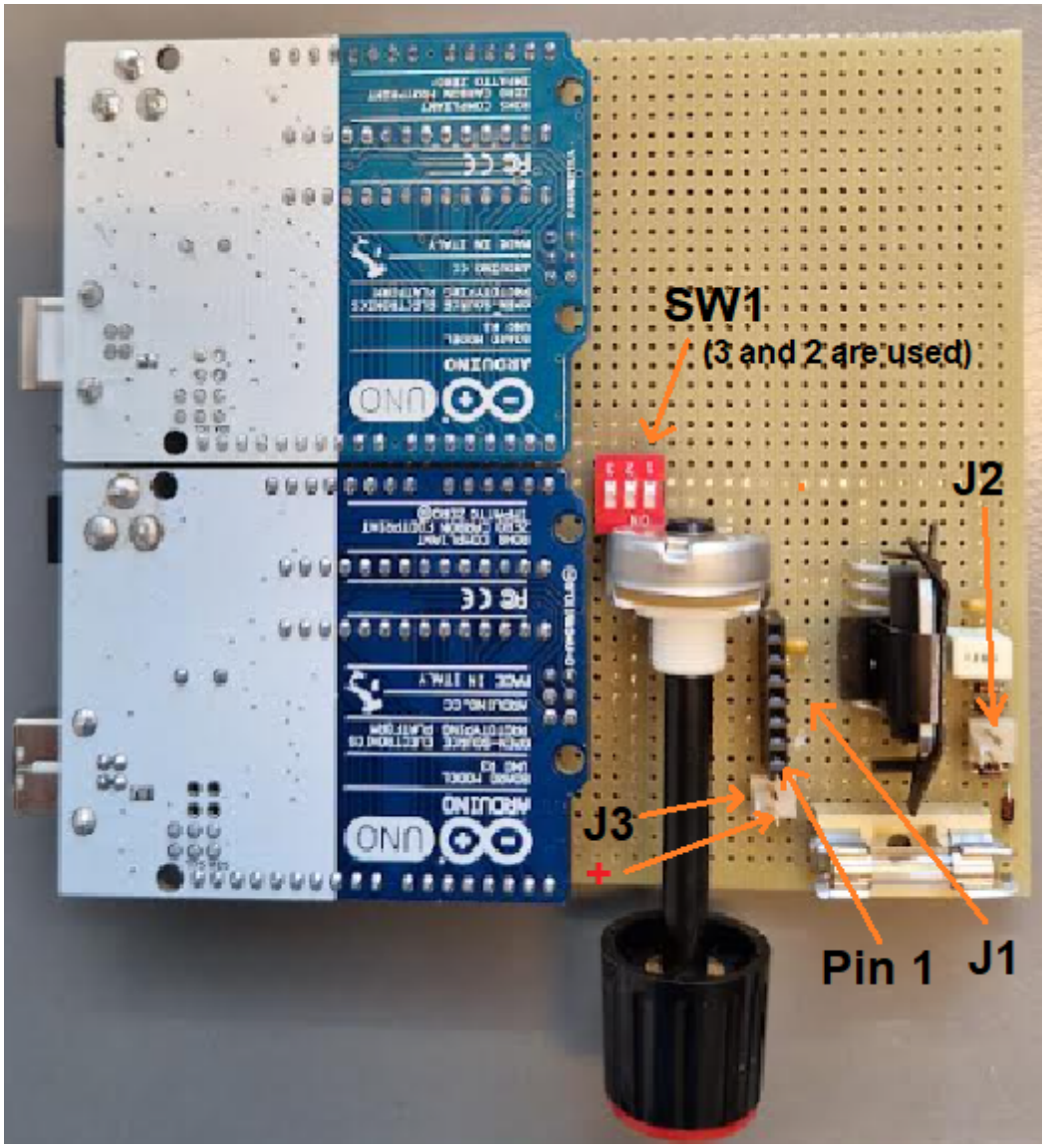
generatoren sættes til 40 Hz, så vil duty-cycle stige, og omvendt, hvis frekvensen hæves.
Vi har ikke direkte nedskrævede tal på denne test, men den er lavet og fundet funktionel.

System test

Hele systemet er til sidst testet på det set-up som er vist i afsnit 1.3.6. Det er fundet komplet funktionelt. Det er muligt at sende PID værdier fra den ene arduino til den anden. De korrekte PID værdi modtages. Motoren følger den hastighed som sættes på referencen og i tilfælde af motoren køre forkert retning, ændre arduinoen på retningen, således motoren rottere korrekt retning.

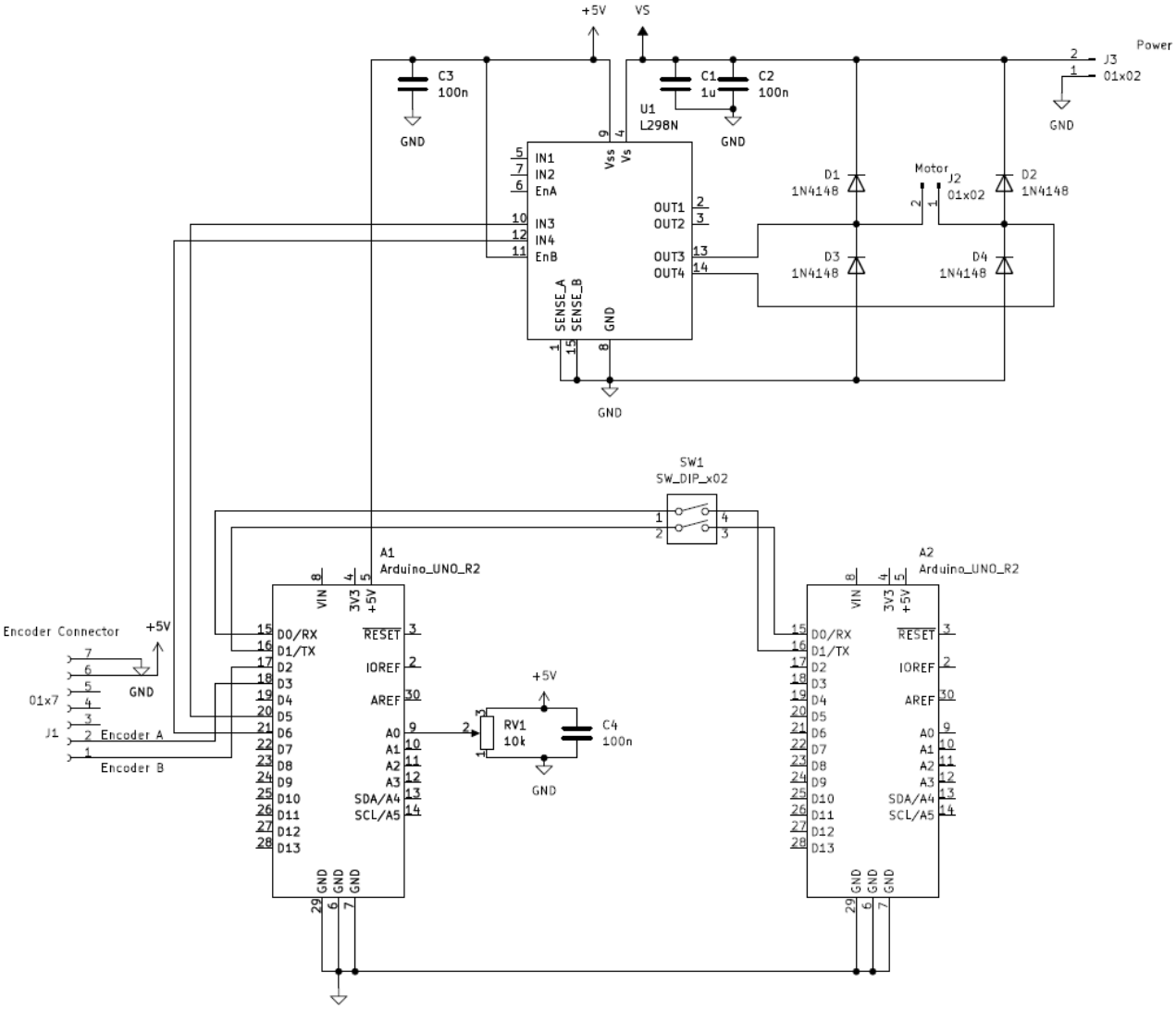
1.3.6 Motor Driver

Et arduino shield er blevet udviklet til at styrer en motor med encoder. Dette shield gør det muligt nemt at koble to arduino uno'er sammen, og med en DIP-SWITCH kan deres RX TX kobles fra, til upload af ny kode mm. Der findes også forbindelser til encoder og et on-board potentiometer til at sætte motorens hastighed. Dette shield kan ses på figur 1.5.



Figur 1.5: Arduino shield til motor styring.

Et diagram over det udviklede shield kan ses i figur 1.6.



Figur 1.6: Arduino shield til motor styring schematic.

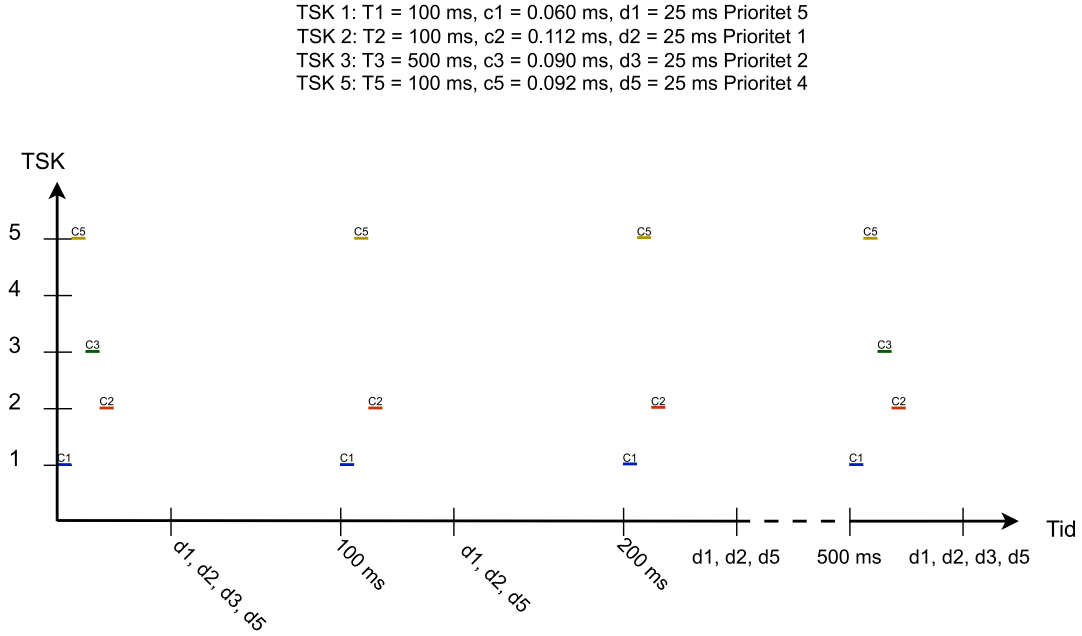
1.3.7 Task opsætning og schedulering

Tiden det tager at udfører alle tasks, eller opgaver er fundet til at være som vist på liste 1.3.7.

- Task 1, TSKSpeedCalculation $C_1 = 60$ µs
- Task 2, TSKSpeedRef $C_2 = 112$ µs
- Task 3, TSKSerialPID $C_3 = 90$ µs
- Task 5, TSKPIDControl $C_5 = 92$ µs

Task 1, 2 og 5 er alle sat til at køre hvert med intervaller på 100 ms. Task 3 er sat til et interval på 500 ms. I tilfælde af at alle task skal køre, er deres dead-line sat således at de alle skal være færdige inden der er gået 100 ms, og de skal til at køre igen. Deres dead-line er derfor sat til 25 ms.
Alt dette kan ses på figur 1.7. Ud fra tiden hver task bruger, og deres deadline er det også muligt at finde U, Utilization, som ses i ligning 1.1. Det ses også at U er mindre end 1 og at systemet derfor vil være stabilt.

$$U = \frac{d_1}{T_1} + \frac{d_2}{T_2} + \frac{d_3}{T_3} + \frac{d_5}{T_5} \rightarrow \frac{25ms}{100ms} + \frac{25ms}{100ms} + \frac{25ms}{500ms} + \frac{25ms}{100ms} = \frac{4}{5} \leq 1 \qquad (1.1)$$



Figur 1.7: Skedulering af de forskellige tasks.

I Bilag A er den fulde kode til Motorcontroller programmet vedhæftet.

Bilag A

Fuld kode til hastighedscontroller

```
1  #include <Arduino.h>
2  #include <Arduino_FreeRTOS.h>
3  #include <semphr.h>
4  #include <PID_v1.h>
5
6  #define speed_ 1
7  #define speedRef_ 2
8  #define motorDCY_ 3
9  #define p_ 4
10 #define i_ 5
11 #define d_ 6
12 //Pin IO
13 #define isrPin 2    //interupt pin
14 #define encPin 3    //encoder direction determination pin
15 #define pwmPin1 5   // PWM output pin
16 #define pwmPin2 6
17
18 #define setDir false
19 //Hardcoded values
20 #define encPPR 100.0   //encoder pulses per rotation
21
22 #define waitTimePIDCalc 100
23 #define waitTimeRef 100
24 #define waittimePIDGet 500
25 #define waittimeSpeedCalc 100.0
26
27 unsigned long timer = 0;
28
```

```
29 struct ctrl {
30   float speed;
31   uint16_t speedRef;
32   int motorDCY;
33   float p, i, d;
34 } data;
35
36 struct pid_setting {
37   double setpoint, input, output;
38   float p, i, d;
39 } pid_init;
40
41 struct serialData {
42   uint8_t serialData[25];
43   uint8_t serialDataArraySize;
44 };
45
46 PID pid_data(&pid_init.input,
47              &pid_init.output,
48              &pid_init.setpoint,
49               pid_init.p,
50               pid_init.i,
51               pid_init.d,
52              DIRECT);
53
54 uint16_t encoderCount;
55 bool motorDirection;
56 bool runDir;
57
58 SemaphoreHandle_t mutexStruct;
59
60 void setup() {
61   data.p = 3;
62   data.i = 8;
63   data.d = 0.8;
64
65   runDir = setDir;
66
67   pinMode(12, OUTPUT);
68   digitalWrite(12, HIGH);
69   Serial.begin(115200);
```

```
70    mutexStruct = xSemaphoreCreateMutex();
71    PIDControlInit();
72
73    pinMode(isrPin, INPUT_PULLUP);
74    pinMode(encPin, INPUT_PULLUP);
75    pinMode(pwmPin1, OUTPUT);
76    pinMode(pwmPin2, OUTPUT);
77    pinMode(A0, INPUT);
78    attachInterrupt(digitalPinToInterrupt(isrPin), EncoderISR,
          RISING);
79
80    while (!Serial)
81      ;
82    Serial.println("TASK1");
83    //Task1 TSKSpeedCalculation (Priority 1)
84    // Task one takes roughly 60 us
85    xTaskCreate(TSKSpeedCalculation,  // Task function
86                "Speed Calculation",  // Task name for humans
87                150,
88                NULL,  // Task parameter
89                5,     // Task priority
90                NULL);
91    //Task2 TSKSpeedRef (Priority 4)
92    //Serial.println("TASK2");
93    // Task two takes roughly 112 us.
94    xTaskCreate(TSKSpeedRef,      // Task function
95                "Speed Ref ADC",  // Task name for humans
96                150,
97                NULL,  // Task parameter
98                1,     // Task priority
99                NULL);
100
101   // Task3 TSKSerialPID (Priority 3)
102   // Task three takes roughly 90 us.
103   Serial.println("TASK3");
104   xTaskCreate(TSKSerialPID,             // Task function
105               "Serial Communication",  // Task name for
                      humans
106               150,
107               NULL,  // Task parameter
108               2,     // Task priority
```

```
109                 NULL);
110
111
112   // //Task5 TSKPIDControl (Priority 4)
113   // Task five takes roughly 92 us.
114   Serial.println("TASK5");
115   xTaskCreate(TSKPIDControl,          // Task function
116               "PID Control block",  // Task name for humans
117               150,
118               NULL,  // Task parameter
119               4,     // Task priority
120               NULL);
121 }
122
123
124 void TSKSpeedRef() {
125   while (1) {
126     ctrl speedRefStruct;
127     //Serial.println("Task2");
128     float sensorValue = analogRead(A0) / 4;
129
130     // float sensorValue = 300;
131
132     speedRefStruct.speedRef = sensorValue;
133
134     // speedRefStruct.speedRef = 127;
135
136     UPDStruct(speedRefStruct, speedRef_);
137     vTaskDelay(waitTimeRef / portTICK_PERIOD_MS);
138   }
139 }
140
141 void EncoderISR() {
142   encoderCount++;
143   motorDirection = digitalRead(encPin);
144 }
145
146 void TSKSpeedCalculation() {
147   while (1) {
148     //Serial.println("Task1");
149     ctrl speedCal = data;
```

```
150
151     //speedCal.speed = ((float)((encoderCount / encPPR)) /
            ((0.001 / 60)));
152
153     //speedCal.speed = ((encoderCount*1.0) / (encPPR*1.0)) *
            (60000.0 / (millis()-timer));
154     speedCal.speed = ((encoderCount * 1.0) * (1000 / (millis
            () - timer)));
155     encoderCount = 0;
156     timer = millis();
157     //Serial.println(speedCal.speed);
158     UPDStruct(speedCal, speed_);
159     // Serial.print("Data:");
160     // Serial.println(data.speed);
161     vTaskDelay(waittimeSpeedCalc / portTICK_PERIOD_MS);
162   }
163 }
164
165 // void TSKMotorPWM() {
166 //   while (1) {
167 //     Serial.println("Task4");
168 //     analogWrite(pwmPin, data.motorDCY);
169 //     vTaskDelay(500 / portTICK_PERIOD_MS);
170 //   }
171 // }
172
173 void PIDControlInit() {
174   pid_data.SetMode(AUTOMATIC);
175   pid_data.SetOutputLimits(-2550, 2550);
176 }
177
178
179 void FunnyFunctionDoingShit() {
180   //Serial.println("This is some shit");
181 }
182
183 void TSKPIDControl(void *pvParameters) {
184   while (1) {
185
186     ctrl pid_ctrl = data;
187     pid_init.input = pid_ctrl.speed * 0.34;
```

```
188     pid_init.setpoint = pid_ctrl.speedRef;
189
190     if (pid_init.p != pid_ctrl.p || pid_init.i != pid_ctrl.i
           || pid_init.d != pid_ctrl.d) {
191       pid_data.SetTunings(pid_ctrl.p,
192                           pid_ctrl.i,
193                           pid_ctrl.d);
194       pid_init.p = pid_ctrl.p;
195       pid_init.i = pid_ctrl.i;
196       pid_init.d = pid_ctrl.d;
197       UPDStruct(pid_ctrl, p_ + i_ + d_);
198     }
199     pid_data.Compute();
200
201
202
203     pid_ctrl.motorDCY = pid_ctrl.speedRef + (pid_init.output
           / 10);
204
205     if (pid_ctrl.motorDCY >= 255) {
206       pid_ctrl.motorDCY = 255;
207     } else if (pid_ctrl.motorDCY <= 0) {
208       pid_ctrl.motorDCY = 0;
209     }
210
211
212     if (motorDirection != setDir) {
213       runDir = !runDir;
214     }
215
216     if (runDir == true) {
217       digitalWrite(pwmPin1, false);
218       analogWrite(pwmPin2, pid_ctrl.motorDCY);
219     } else {
220       digitalWrite(pwmPin2, false);
221       analogWrite(pwmPin1, pid_ctrl.motorDCY);
222     }
223
224     UPDStruct(pid_ctrl, motorDCY_);
225     vTaskDelay(waitTimePIDCalc / portTICK_PERIOD_MS);
226   }
```

```
227 }
228
229
230
231
232 uint8_t GetSerialDataArrSize(uint8_t *arr) {
233   uint8_t arrSize = 0;
234 }
235
236 serialData GetSerialData() {
237   serialData incomingData;
238   uint8_t moreData = 0, currentChar, terminator = '#', index
        = 0;
239   while ((Serial.available() > 0) && (moreData == 0)) {
240     currentChar = Serial.read();
241     //Serial.print(currentChar);
242     if (currentChar != terminator) {
243       incomingData.serialData[index] = currentChar;
244       index++;
245     } else {
246       incomingData.serialData[index] = '#';
247       incomingData.serialDataArraySize = index;
248       index = 0;
249       moreData = 1;
250     }
251   }
252   return incomingData;
253 }
254
255 uint8_t *GetPIDValues(serialData *incomingData) {
256   static uint8_t values[3];
257   values[0] = GetPVal(incomingData);
258   values[1] = GetIVal(incomingData);
259   values[2] = GetDVal(incomingData);
260   return &values[0];
261 }
262
263 uint8_t GetPVal(serialData *incomingData) {
264   uint8_t pVal = 0, currentChar = 0, index = 0, pFound = 0;
265
266   while (incomingData->serialData[index] != '|' && (index < (
```

```
      uint8_t)sizeof(incomingData->serialData)) && pFound ==
      0) {
267     currentChar = (char)incomingData->serialData[index];
268     if (currentChar != 'P') {
269       index++;
270     } else {
271       do {
272         currentChar = (char)incomingData->serialData[index];
273         if ((char)currentChar == '%' && pFound == 0) {
274           index++;
275           pVal = (uint8_t)incomingData->serialData[index] - '
              0';  //Convert to int
276           pFound = 1;
277         } else {
278           index++;
279         }
280       } while (pFound == 0);
281     }
282   }
283   return pVal;
284 }
285 uint8_t GetIVal(serialData *incomingData) {
286
287   uint8_t iVal = 0, currentChar = 0, index = 0, pFound = 0;
288
289   while ((index < (uint8_t)sizeof(incomingData->serialData)
      && (pFound == 0))) {
290     currentChar = (char)incomingData->serialData[index];
291     if (currentChar != 'I') {
292       index++;
293     } else {
294       do {
295         currentChar = (char)incomingData->serialData[index];
296         if ((char)currentChar == '%' && pFound == 0) {
297           index++;
298           iVal = (uint8_t)incomingData->serialData[index] - '
              0';  //Convert to int
299           pFound = 1;
300         } else {
301           index++;
302         }
```

```
303         } while (pFound == 0);
304       }
305     }
306     return iVal;
307 }
308 uint8_t GetDVal(serialData *incomingData) {
309     uint8_t dVal = 0, currentChar = 0, index = 0, pFound = 0;
310 
311     while ((index < (uint8_t)sizeof(incomingData->serialData)
        && (pFound == 0))) {
312       currentChar = (char)incomingData->serialData[index];
313       if (currentChar != 'D') {
314         index++;
315       } else {
316         do {
317           currentChar = (char)incomingData->serialData[index];
318           if ((char)currentChar == '%' && pFound == 0) {
319             index++;
320             dVal = (uint8_t)incomingData->serialData[index] - '
                0';  //Convert to int
321             pFound = 1;
322           } else {
323             index++;
324           }
325         } while (pFound == 0);
326       }
327     }
328     return dVal;
329 }
330 
331 void ConvToFloatLoadStruct(uint8_t *receivedValues) {
332     ctrl newPIDVals;
333     newPIDVals.p = ((float)*receivedValues);
334     receivedValues++;
335     newPIDVals.i = ((float)*receivedValues);
336     receivedValues++;
337     newPIDVals.d = ((float)*receivedValues) / 10;
338 
339     UPDStruct(newPIDVals, (p_ + i_ + d_));
340 }
341 
```

```
342 void TSKSerialPID(void *pvParameters) {
343   TickType_t xLastWakeTime = xTaskGetTickCount();
344   static serialData incomingData;
345
346   while (1) {
347     //Serial.println("TSK3");
348     if (Serial.available() > 0) {
349       incomingData = GetSerialData();
350     }
351     uint8_t *receivedValues = GetPIDValues(&incomingData);
352     ConvToFloatLoadStruct(receivedValues);
353     /*
354   Serial.print("Final Values");
355   Serial.println();
356   Serial.print("P-VAL: ");
357   Serial.println(data.p);
358   Serial.print("I-VAL: ");
359   Serial.println(data.i);
360   Serial.print("D-VAL: ");
361   Serial.println(data.d);
362 */
363     //vTaskDelayUntil(&xLastWakeTime, 250 /
          portTICK_PERIOD_MS);
364     vTaskDelay(waittimePIDGet / portTICK_PERIOD_MS);
365   }
366 }
367
368
369 void UPDStruct(ctrl updStruct, uint8_t index) {
370
371   if (xSemaphoreTake(mutexStruct, 10) == pdTRUE) {
372     switch (index) {
373       case speed_:
374         data.speed = updStruct.speed;
375         break;
376       case speedRef_:
377         data.speedRef = updStruct.speedRef;
378         break;
379       case motorDCY_:
380         data.motorDCY = updStruct.motorDCY;
381         break;
```

```
382         case p_:
383           data.p = updStruct.p;
384           break;
385         case i_:
386           data.i = updStruct.i;
387           break;
388         case d_:
389           data.d = updStruct.d;
390           break;
391         case p_ + i_ + d_:
392           data.p = updStruct.p;
393           data.i = updStruct.i;
394           data.d = updStruct.d;
395           break;
396 
397         default:
398           data = updStruct;
399           break;
400       }
401 
402       xSemaphoreGive(mutexStruct);
403     }
404 }
405 
406 void loop() {
407 }
```