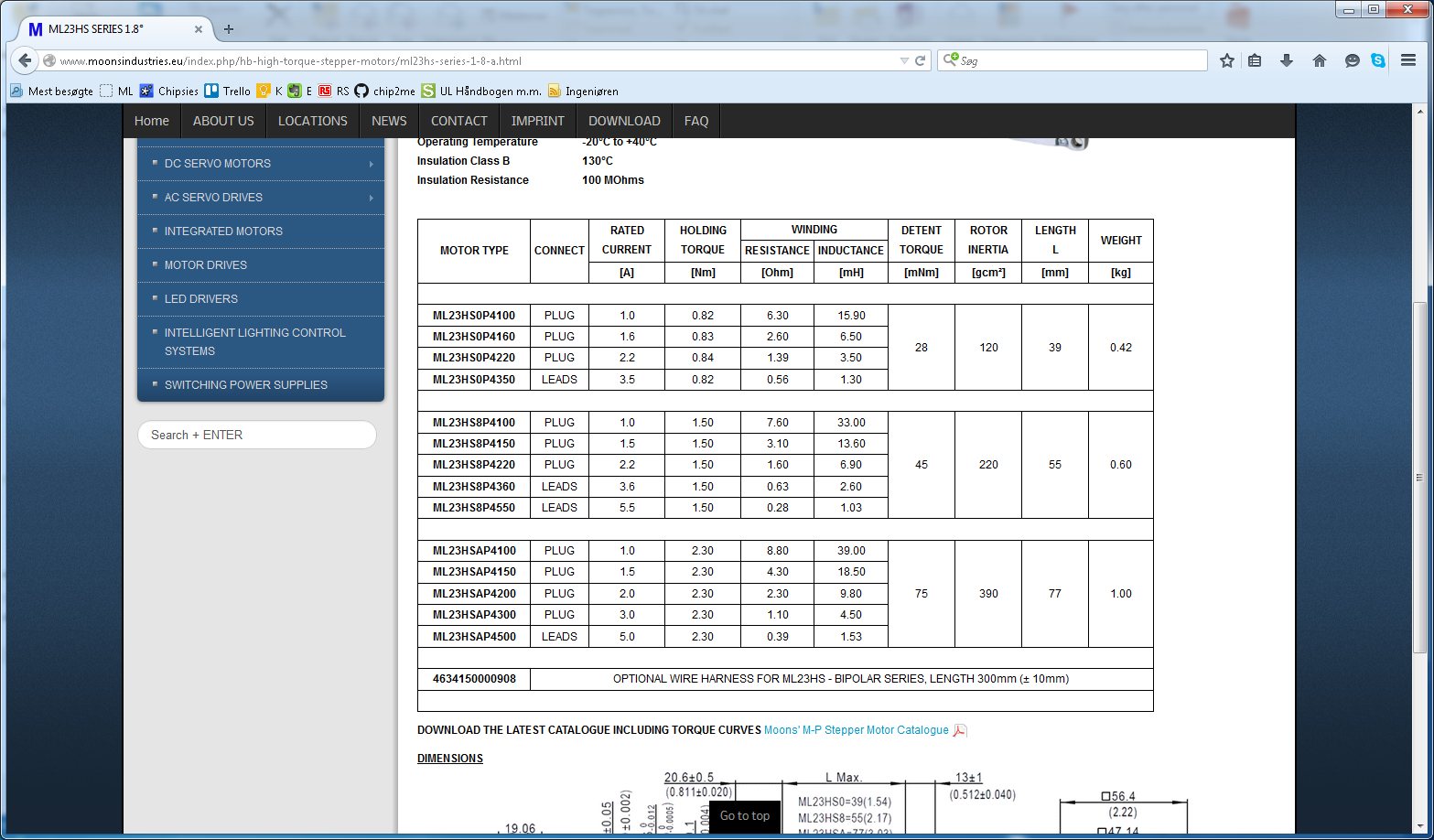
Skype/Teamviewer 5. august 2015 - Maximiliam Hase (miControl).

mcTools ikke klar til tuning af stepmotor (kommer i senere release)

Nuværende version af mcTools V1.08.00.00

Maximiliam sendte et python script ”mcDSA-S65-tuning\_en.py” samt et bibliotek (from pi\_curr\_betragsoptimum import CalcCurrKpKi)

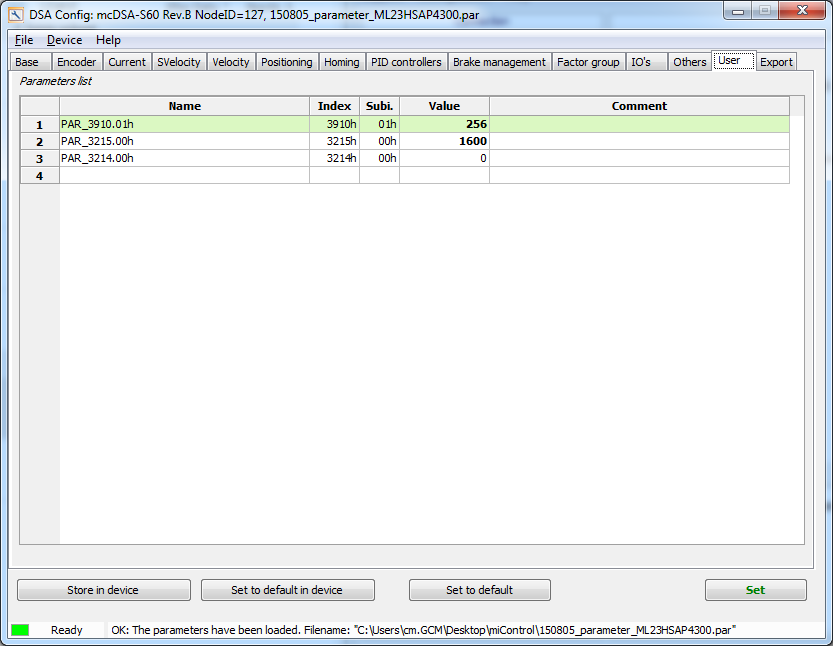
I datablad for motoren (her Moons ML23HSAP4300) aflæses inductance, resistance.



Data sættes ind i script, og scriptet køres for at få de korrekte data.

Se i bilag kode der er markeret der hvor der skal ændres i scriptet.

Efter man har kørt scriptet og fået de korrekte data, så skal disse programmeres ned i styringen.  
Dette kan f.eks udføres vha. mcTools således:



## Remarks motor wiring:

Moons wire A, B, C, D hænger sammen A+C og B+D

miControl forventer A+B og C+D

## Remarks Ue / Up wiring:

Pas på at Ue ikke bliver > 30VDC.

Ved car-battery applications vil batteriet normalt tage den overskydende energi (no problems at CleanKeeper)

Pas især på ved power supply med høj indg. Modstand samt lifting applications. Brake chopper is available.

Ingen kendte applications notes med beskyttelses-circuit mellem Ue og Up.

BILAG – Script for tuning af stepper:

# -\*- coding: iso-8859-1 -\*-

''' Script to assist to determine the currentcontorller parameters for a stepper motor WITHOUT ENCODER

Operating a stepper without encoder means, that the controller operates without feedback. Depending on the position

of the rotor fix current values get created and this leads to steps (Microstep). According to the

desired velocity a fixed rotating field will be created.

To build a working motor torque it is important that the current increase (Kp, Ki, Kvff, Kaff) and the amount

of the current (Curr) are correct. Is the current increase to low the motor can't follow the rotating field properly

and will loose steps. Is the current increase to high the motor will generate noise and will heat up.

Kp, Ki, Kaff are determined calculational.

The desired current should not be higher than the specified motor current.

To determine the desired current (Curr) and the current increase empirical choose the highest operating point

(highest load torque, speed and acceleration).

Parameters that have to be set:

1. 0x3210.0 - CURR\_Kp - P - value calculated with L(inductance) and R(resistance) of the motor

2. 0x3211.0 - CURR\_Ki - I - value calculated with L(inductance) and R(resistance) of the motor

3. 0x3214.0 - CURR\_Kvff - current feedforward, determined empirical. Values 0..100

4. 0x3215.0 - CURR\_Kaff - current acceleration feedforward, determined automatically

5. 0x3200.0 - CURR\_DesiredValue - desired current (amplitude)'''

from mc.dsa import \*

**from pi\_curr\_betragsoptimum import CalcCurrKpKi**

import time

#-----------------------------------------------------------------------------

# Inputdata:

# insert the coorect data according to the steppermotor which is in use

NodeId = 127 # CAN NodeId of the device

Up = 24.0 # Power supply Voltage [V]

# Motor:

L = 4.5E-3 # Motor-inductance [H]

R = 1.1 # Motor-resistance [Ohm]

MotPolN = 200 # Number of fullsteps per round

Microsteps = 256 # 256 (default), 128, 64, 32, 16, 8, 4, 2, 1

Vel = 55 # velocitiy [rpm]

Pos = Microsteps\*MotPolN\*10

TunningCurrent = 700

CurrKvff = 0

#-----------------------------------------------------------------------------

def GetAvgCurrFollowingError (n=16):

curr = 0

for i in xrange(n):

curr += d.CurrFollowingError()

return curr/n

def GetMaxCurrFollowingError (n=32):

curr = 0

for i in xrange(n):

c = abs(d.CurrFollowingError())

if c > curr:

curr = c

return curr

def GetAvgCurr (n = 16):

curr = 0

for i in xrange(n):

curr += d.ActCurr()

return curr/n

def StepperTunning (curr = 300):

kp,ki = CalcCurrKpKi(l\_H=L, r\_Ohm=R, up\_V=Up, tsamp\_s=100E-6, fpwm\_Hz=50E3)

kp /= 5

ki /= 5

d.Disable()

d.ClearError()

d.SdoWr(0x3910,1, 1) # MotMicrosteps = Fullstep

d.ModePos()

d.Curr(curr)

d.CurrKp(100)

d.CurrKi(100)

d.SdoWr(0x3214,0, 0) # Kvff

d.SdoWr(0x3215,0, 0) # Kaff = R\_mOhm \* 24000/Up\_mV

d.Vel(10)

d.Enable()

for p in xrange(4):

time.sleep(0.100)

if d.CmdCurr() > 0:

break

else:

d.Movr(1)

err = 0

if abs(GetAvgCurrFollowingError()) > curr/2:

print "ERROR: Motor coil not connected !"

err = -1

if not err:

d.CurrKp(0)

d.CurrKi(0)

d.Disable()

d.Enable()

ok = False

r = 0

dr = 100

while not ok:

d.SdoWr(0x3215,0, r) # Kaff = R\_mOhm \* Up\_mV/24000

d.Mova(d.ActPos())

r += dr

fc = GetAvgCurrFollowingError()

if fc < -(20\*curr/100):

err = -2

break

if fc < (5\*curr/100):

ok = True

d.Disable()

return err,kp,ki,r

#-----------------------------------------------------------------------------

d = Dsa(NodeId)

print "Start..."

err,kp,ki,r, = StepperTunning(curr=TunningCurrent)

if not err:

d.CurrKp(kp)

d.CurrKi(ki)

d.SdoWr(0x3214,0, CurrKvff) # Kvff

d.SdoWr(0x3215,0, r) # Kaff = R\_mOhm \* 24000/Up\_mV

d.SdoWr(0x3910,0, MotPolN) # MotPolN

d.SdoWr(0x3910,1, Microsteps) # MotMicrosteps

print "Values of Parameters:"

print " CURR\_Kp (0x3210.0) =", kp

print " CURR\_Ki (0x3211.0) =", ki

print " CURR\_Kvff (0x3214.0) =", CurrKvff

print " CURR\_Kaff (0x3215.0) =", r

print " MOTOR\_PolN (0x3910,0) =", MotPolN

print " MOTOR\_Microstepsd (0x3910,1) =", Microsteps

print "Done."

#-----------------------------------------------------------------------------

# driving test

if 1 and not err: #CHANGE from 0 to 1

d.Disable()

d.PrintInfos()

d.ClearError()

d.ModePos()

d.Curr(TunningCurrent)

d.VelAcc\_dV(1000)

d.VelAcc\_dT(1000)

d.VelDec\_dV(1000)

d.VelDec\_dT(1000)

d.Vel(Vel)

d.Enable()

#- assistance functions --------------------------------------------------------

def \_mov (d, vel, pos, rel=False):

d.SdoWr(0x3300,0, vel) # Vel

if rel:

d.SdoWr(0x3791,0, pos) # Movr

else:

d.SdoWr(0x3790,0, pos) # Mova

reached = False

while not reached:

status = d.SdoRd(0x3002,0)

if status & (1<<4): # Target reached ?

reached = True

elif status & (1<<1): # Error ?

# insert steps to correct error here

break

return reached

def Mova (d, vel, pos):

return \_mov(d, vel, pos, False)

def Movr (d, vel, pos):

return \_mov(d, vel, pos, True)

#- mainloop ----------------------------------------------------------

while 1:

Movr(d, Vel, Pos)

time.sleep(1.2)

print d.ActPos(), d.CmdPos(), d.ActCurr(), d.TrgCurr(), d.CmdCurr(), d.CurrFollowingError()

Mova(d, Vel, 0)

time.sleep(1.2)

print d.ActPos(), d.CmdPos(), d.ActCurr(), d.TrgCurr(), d.CmdCurr(), d.CurrFollowingError()