MTD2A_binary_output

MTD2A: Model **T**rain **D**etection **A**nd **A**ction – arduino library https://github.com/MTD2A/MTD2A
Jørgen Bo Madsen / V1.6 / 25-10-2025

MTD2A_binary_output er en brugervenlig avanceret og funktionel C++ klasse til tidsstyret håndtering af output til relæer, LED'er og meget mere. MTD2A understøtter parallel processering og asynkron eksekvering.

Klassen er blandt en række logiske byggeklodser, der løser forskellig funktioner. Fælles for dem alle:

- Understøtter en bred vifte af inputsensorer og outputenheder
- Er enkle at bruge til at bygge komplekse løsninger med få kommandoer
- Fungere Ikke-blokerende, procesorienteret og tilstandsdrevet
- Tilbyder omfattende kontrol- og fejlfindingsinformation
- Grundigt dokumenterede med eksempler

Indholdsfortegnelse

Funktionsbeskrivelse

MTD2A_binary_output processen består af 4 funktioner:

1. `MTD2A_binary_output object_name ("object_name" , outputTimeMS, beginTimeMS, endTimeMS, { BINARY | PWM }, pinBeginValue, pinEndValue);`
2. `object_name.initialize ( pinNumber, startPinValue );`
   Kaldes i `void setup ();` og efter `Serial.begin (9600);`
3. `object_name.activate ();` Aktiverer en gang og virker først igen når processen er afsluttet (COMPLETE)
4. `MTD2A_loop_execute ();` Kaldes som det sidste I `void loop ();`

Alle funktioner benytter default værdier og kan derfor kaldes med ingen og op til max antal parametre.
Dog skal parametre angives i stigende rækkefølge startende fra den første. Se eksempl herunder:

```
MTD2A_binary_input object_name;
MTD2A_binary_input object_name
  (”object_name”);
  (”object_name” , outputTimeMS);
  (”object_name” , outputTimeMS, beginTimeMS);
  (”object_name” , outputTimeMS, beginTimeMS, endTimeMS);
  (”object_name” , outputTimeMS, beginTimeMS, endTimeMS, pinOutputMode);
  (”object_name” , outputTimeMS, beginTimeMS, endTimeMS, pinOutputMode, pinBeginVlaue);
  (”object_name” , outputTimeMS, beginTimeMS, endTimeMS, pinOutputMode, pinBeginVlaue, pinEndValue);
```

Defaults:

```
  (”Object name” , 0              , 0              , 0              , BINARY         , HIGH           , LOW);
```

Eksempel

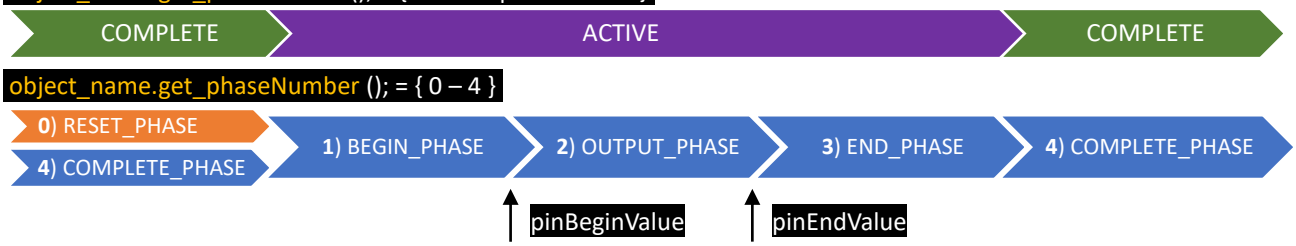
```
// Two blinking LEDs. One with symmetric interval and another with asymetric interval.

#include <MTD2A.h>
using namespace MTD2A_const;

MTD2A_binary_output red_LED   ("Red LED",   400, 400);  // 0.4 sec light, 0.4 sec no light
MTD2A_binary_output green_LED ("Green LED", 300, 700, 0, PWM, 96);  // 0,3 / 0.7 sec PWM dimmed

void setup() {
  Serial.begin(9600);
  while (!Serial) { delay(10); } // ESP32 Serial Monitor ready delay

  byte RED_LED_PIN   = 9;  // Arduino board pin number
  byte GREEN_LED_PIN = 10; // Arduino board pin number
  red_LED.initialize   (RED_LED_PIN);
  green_LED.initialize (GREEN_LED_PIN);
  Serial.println("Two LED blink");
}

void loop() {
  if (red_LED.get_processState()   == PENDING) {
    red_LED.activate();
  }
  if (green_LED.get_processState() == PENDING) {
    green_LED.activate();
  }

  MTD2A_loop_execute();
}
```

Flere eksempler og youtube video:
https://github.com/MTD2A/MTD2A/tree/main/examples
DEMO video: https://youtu.be/vLySY92JdAM

Proces faser

Afhægig af den aktuelle konfiguration gennemføres processen i mellem 1 og 5 faser.

object_name.get_processState (); = { ACTIVE | COMPLETE }



object_name.get_phaseNumber (); = { 0 – 4 }

0. Den initelle fase når programmet starter samt når funktion reset (); kaldes eller 4) COMPLETE_PHASE.
1. Start forsinkelse. Hvis sat til 0 eller ikke defineret springes fasen over.
2. Output tidsperiode. Starter med pinBeginValue og slutter med pinEndValue.
   Hvis sat til 0 eller ikke defineret springes fasen over, og der skrives en advarsel.
3. Slut forsinkelse. Hvis sat til 0 eller ikke defineret springes fasen over.
4. Processen er afsluttet COMPLETE og klar til ny aktivering object_name.activate ();

Globale nummerkonstanter: RESET_PHASE, BEGIN_PHASE, OUTPUT_PHASE, END_PHASE & COMPLETE_PHASE

Det **øjeblikkelige** fasesskift kan identificeres med funktion: object_name.get_phaseChange (); = { true | false }

Proces status

Ved overgang til BEGIN_PHASE, OUTPUT_PHASE eller END_PHASE skifter ProcessState til ACTIVE.
Ved overgang til COMPLETE_PHASE eller RESET_PHASE skifter processState til COMPLETE.

Timing

Se dokumentet MTD2A.pdf og afsnittet ”Kadance ” og ”Synkronisering” samt ”Eksekveringshastighed”.

Initialisering

object_name.initialize (pinNumber, startPinValue); Kaldes i void setup (); og efter Serial.begin (9600);
Output skrives til det digitale benforbindelsesnummer, der er specificeret i object_name.initialize (pinNumber);
Kaldes funktionen ikke, eller engives der en benforbinelser der ikke understøttes, bliver der ikke skrevet til benforbindelsen: pinWriteToggl = DISABLE og pinNumber = 255.

Output på benforbindelsen kan inverteres (omvendes) ved at sætte pinOutput til INVERTED
Det betyde at der byttes om på HIGH og LOW og PWM værdi regnes til 255 – PWM værdi.
object_name.initialize (pinNumber, {NORMAL | **INVERTED**});

Benforbindelse initialiseres med den værdi der angivet i startPinValue
startPinValue skrives **øjeblikkeligt** til pinNumber Udelades parameteren angives LOW

object_name.initialize (pinNumber, {**NORMAL** | INVERTED}, **startPinValue**);

Hvis benforbindelsenummer er initialiseret korrekt med object_name.initialize (pinNumber), er det muligt løbende at styre om der skal skrives til benforbinelsen eller ej med funktionen:
object_name.set_pinWriteToggl ({**ENABLE** | DISABLE});

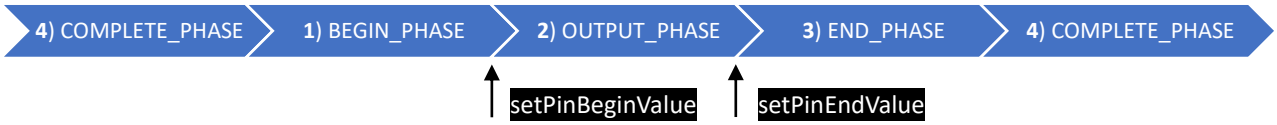
Det er også muligt at skrive direkte til benforbindelsen med funktionen:
object_name.pinWriteValue (PinValue); binary {HIGH | LOW} / PWM {0-255}
object_name.pinWriteValue (PinValue, {BINARY | P_W_M); Valgfri parameter. Ingen default værdi.
Dette sker **øjeblikkeligt**, uanset om processen er aktiv eller ej, og uafhængigt af MTD2A_loop_execute ();
Som udgangspunkt er benforbindelsen **udefineret**. Se Digital Pins | Arduino Documentation

Aktivering

Processen aktiveres med funktionen: object_name.activate (); Dermed skifter processState til ACTIVE
Efterfølgende aktivering har ingen effekt, så længe processen er aktiv. Så snart processState skifter til COMPLETE kan processen aktiveres igen.

Processen kan til hver en tid nulstilles med funktionen: object_name.reset (); Funktionen nultiller alle styrings- og process variable **øjeblikkeligt**, før MTD2A_loop_execute (); , og gør klar til ny start. Alle funktionskonfigurerede variable og standard værdier bibeholdes. Procesfasen skifter til RESET_PHASE
Hvis benforbindelsen er defineret korrekt, skrives startPinValue til benforbindelsen **øjeblikkeligt**.

Der er muligt at skrive begyndelseværdier pinBeginValue og slut værdier pinEndValue til benforbindelsen.



Activate funktioner benytter "function overloading". Det betyder at funktionen kan kaldes med ingen og op til 4 parametre. Dog skal parametre angives i stigende rækkefølge startende fra den første.

Der benyttes **ikke** default værder. Alle eksisterende værdier forbliver de samme, med mindre de angives som parametre i funktionskaldet. Se eksempel herunder:

```
object_name.activate ();
object_name.activate (setPinBeginValue);
object_name.activate (setPinBeginValue, setPinEndValue);
object_name.activate (setPinBeginValue, setPinEndValue, setPWMcurveType);
object_name.activate (setPinBeginValue, setPinEndValue, setPWMcurveType, setOutputTimeMS);
```

setPWMcurveType angiver hvilken kurve der skal følges i tidsrummet outputTimeMS Kurven starter med værdien setPinBeginValue og slutter med værdien setPinEndValue.
setOutputTimeMS angiver tidsrummet mellem setPinBeginValue og setPinEndValue i millisekunder.

Stop og reset (restart) timer

Det er muligt at sætte det nyt start tidspunkt for aktuel timerproces, og stoppe aktuel timerproces i utide.
RESET_TIMER Sætter et nyt starttidspunkt som hentes fra den globalt synkroniserede tid globalSyncTimeMS
STOP_TIMER afbryder tidskontrol processen

```
object_name.set_outputTimer ( {STOP_TIMER | RESET_TIMER} );
object_name.set_beginTimer ( {STOP_TIMER | RESET_TIMER} );
object_name.set_endTimer ( {STOP_TIMER | RESET_TIMER} );
```

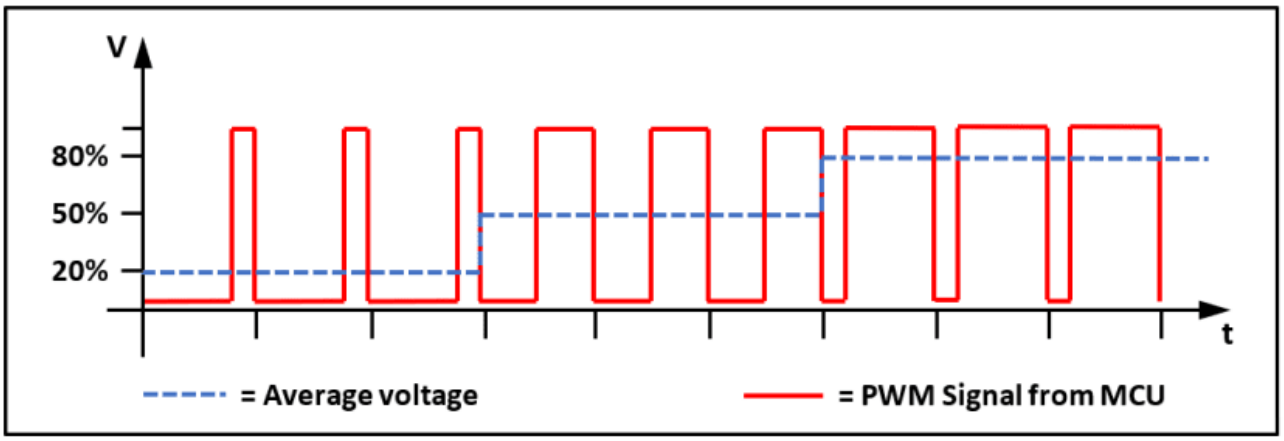
Det nye starttidspunkt hentes fra den globalt synkroniserede tid og kan aflæses med funktionen:
MTD2A::globalSyncTimeMS ();

Tidsperioden sættes som standard når objektet instantieres (aktiveres).
Det er muligt at definere nye tidsperider for alle tre tidsperioder:

```
object_name.set_outputTimeMS ( {0 - 4294967295} );
object_name.set_beginDelayMS ( {0 - 4294967295} );
object_name.set_endDelayMS ( {0 - 4294967295} );
```

PWM

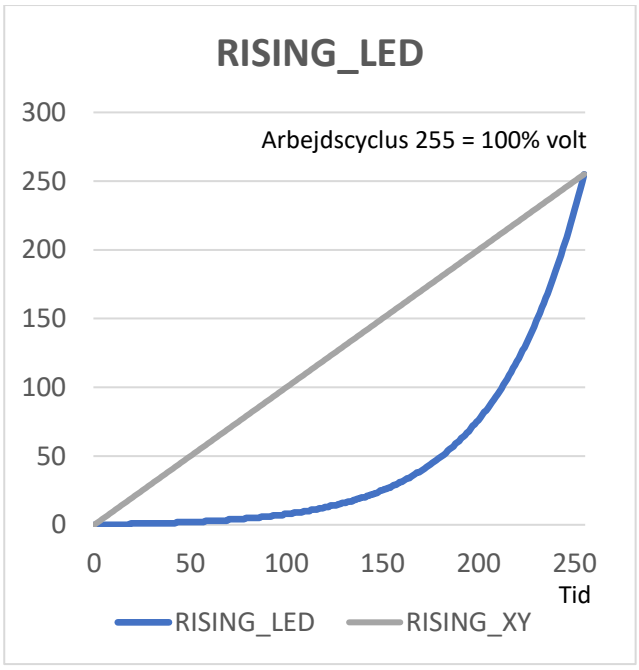
PWM (Pulse Width Modulation) er en teknik, der bruges til at kontrollere den gennemsnitlige effekt, der leveres til en elektrisk enhed ved at variere bredden af en række impulser. I bund og grund er det en digital metode til at skabe en analog effekt ved hurtigt at tænde og slukke for et signal

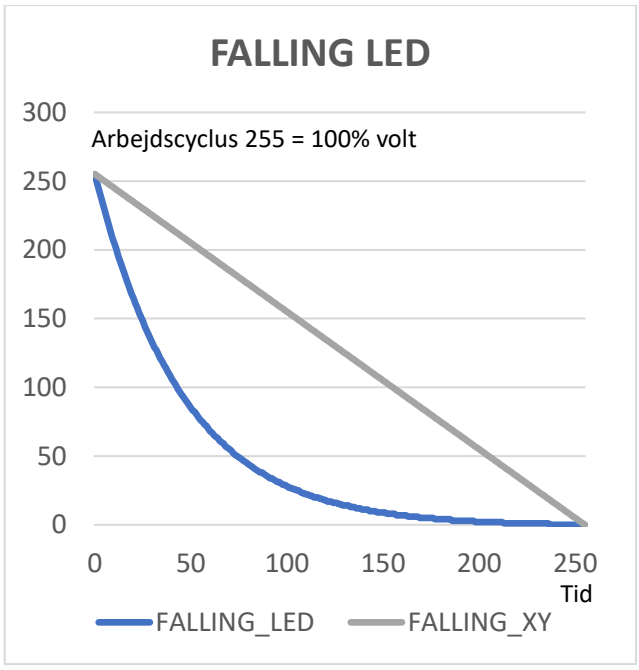


16 forskellige matematiske PWM-kurver er designet: 2 lineære, 4 eksponentielle og 10 potensfunktioner. Funktionerne er beregnet til at udjævne fysiske forhold, der har et ikke-lineært forhold mellem tid og funktion. F.eks. LED fade ind/fade ud og acceleration og deacceleration af DC-motorer (tog) og servo.

LED fading

PWM er en god måde at kontrollere lysstyrken på LED'er. Forholdet mellem arbejdscyklussen og den opfattede lysstyrke er dog slet ikke lineært. Mennesker opfatter lysstyrkeændringen ikke-lineært. Det menneskelige øje reagerer logaritmisk på lys og har en bedre følsomhed ved lav luminans end høj luminans.

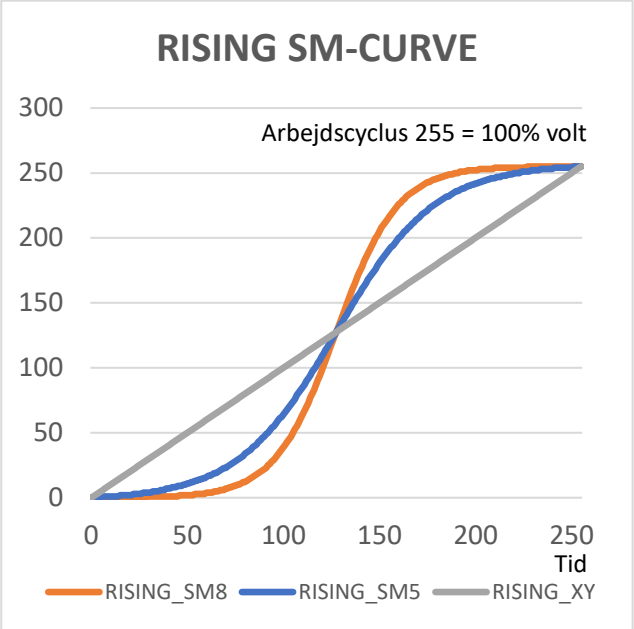


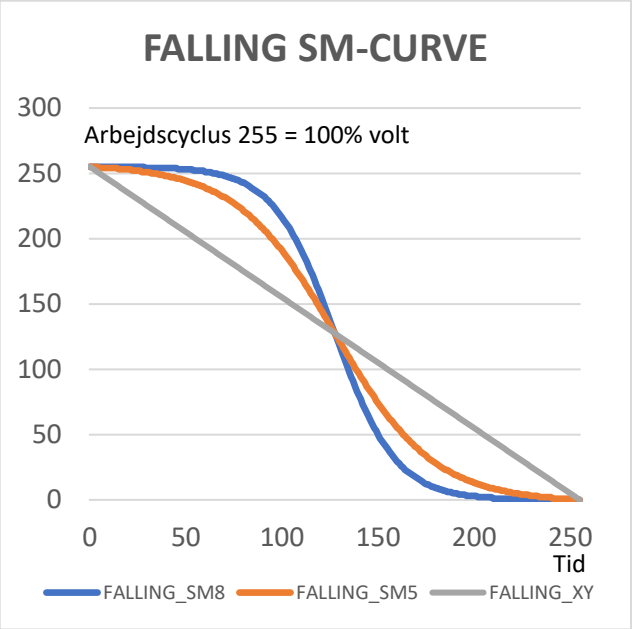


Arduino eksempel: MTD2A/examples/math_fade_LED at main · MTD2A/MTD2A
DEMO video: https://youtu.be/8TV6nOdXBno

S-kurver

[Sigmoid S-kurver] Kan bruges, hvor blød acceleration og deceleration er at foretrække. For eksempel, at køre en (tungt belastet) servomotor fra punkt a til b, eller en servomotor, der sænker/rejser bomme op og ned ved en jernbaneoverskæring.





Arduino eksempel: [MTD2A/examples/servo_math_curve at main · MTD2A/MTD2A]
DEMO video: [https://youtu.be/rhQtu0iKFl8]

Potensfunktioner

Potensfunktioner er til udjævning af acceleration og deceleration af tog.

For eksempel er brug af potensfunktioner velegnet til pendulkørsel med modeltog. Lokomotivet accelererer jævnt op i hastighed, kører med en fast hastighed i et stykke tid, deaccelererer jævnt ned i hastighed og stopper helt. Efter en kort pause gentages processen i den modsatte retning.

Inerti and friktion

Start og stop af tog (lokomotiv og vogne) involverer både inerti og friktion.

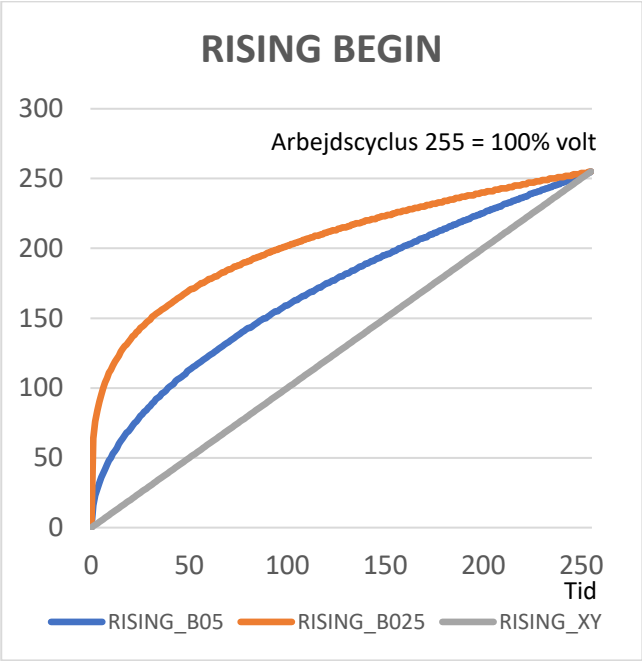
Inerti er egenskaben ved et objekt, der får det til at modstå ændringer i dets bevægelsestilstand. For at starte eller stoppe bevægelse er der brug for en ubalanceret kraft for at overvinde inerti og, i de fleste virkelige scenarier, friktion.

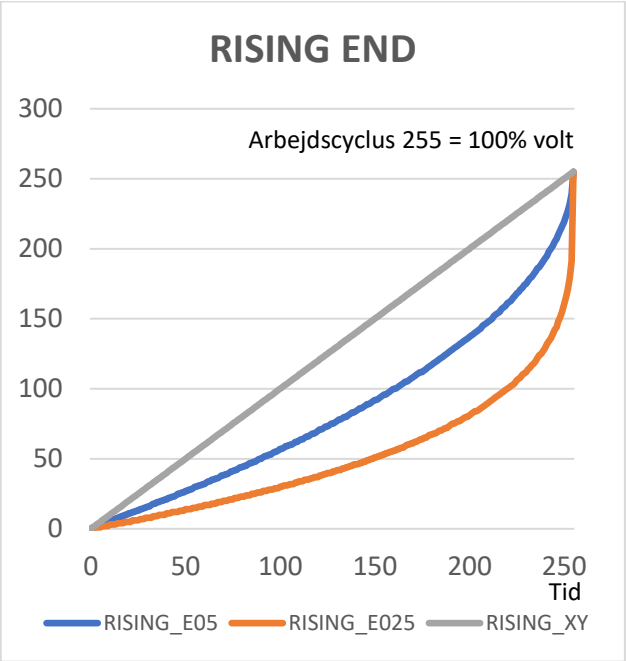
1. En bil, der accelererer, kræver en kraft for at overvinde bilens inerti, som ønsker at forblive stationær.
2. En rullende kugle stopper til sidst på grund af friktion. Uden friktion ville den fortsætte med at rulle.
3. Skubbe en kasse: Du skal skubbe med tilstrækkelig kraft til at overvinde statisk friktion og få kassen i bevægelse. Når du bevæger dig, skal du fortsætte med at anvende kraft for at overvinde kinetisk friktion og opretholde hastigheden.

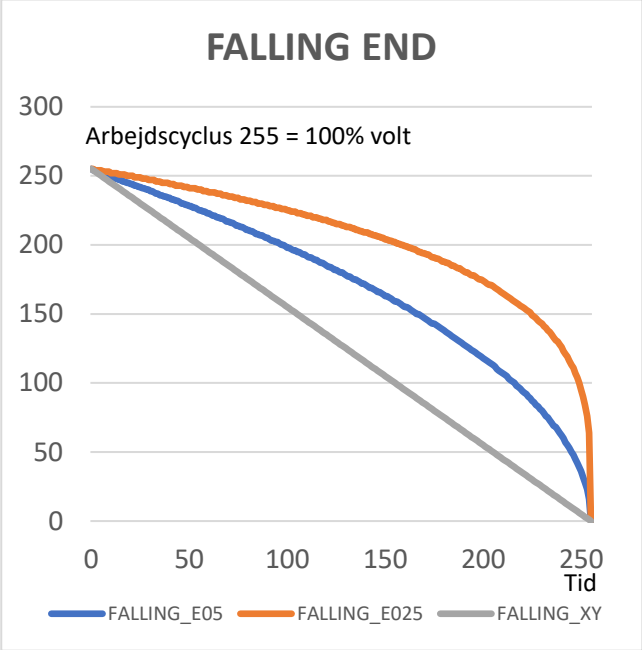
Inerti og friktion er grundlæggende begreber i fysik, der forklarer, hvorfor objekter bevæger sig (eller ikke bevæger sig), og hvordan bevægelse kan startes og stoppes. Forholdet mellem tid og funktion er **ikke lineært**.

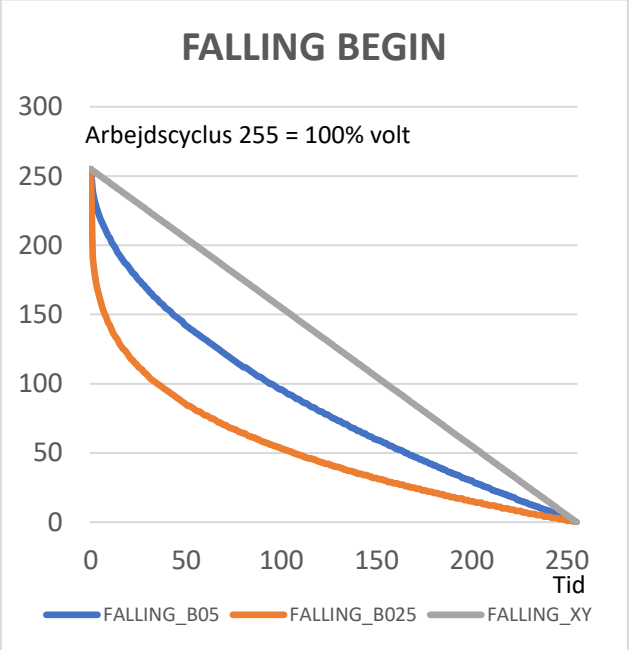
Desuden kan der være en effekt i jævnstrøms motorer kaldet "ankerreaktion". Afhængigt af hvordan motoren er fremstillet, kan der være en asymmetri mellem rotation frem og tilbage.

De følgende 8 effektfunktioner vil resultere i mere jævn togacceleration og -deceleration.









Arduino eksempel: [MTD2A/examples/PWM_power_curves at main · MTD2A/MTD2A]
DEMO video: https://youtu.be/Fi9D1hrzT9M
Avanceret pendul kørsel med matematiske PWM kurver og H-broer: https://youtu.be/1i-cGc6Dk4E

De forskellige kurver er navngivet som globale konstanter (MTD2A_const.h):

MIN_PWM_VALUE MAX_PWM_VALUE

NO_CURVE
RISING_XY
RISING_LED
RISING_SM8
RISING_B05
FALLING_B05

FALLING_XY
FALLING_LED
RISING_SM5
RISING_B025
FALLING_B025

FALLING_SM8
RISING_E05
FALLING_E05

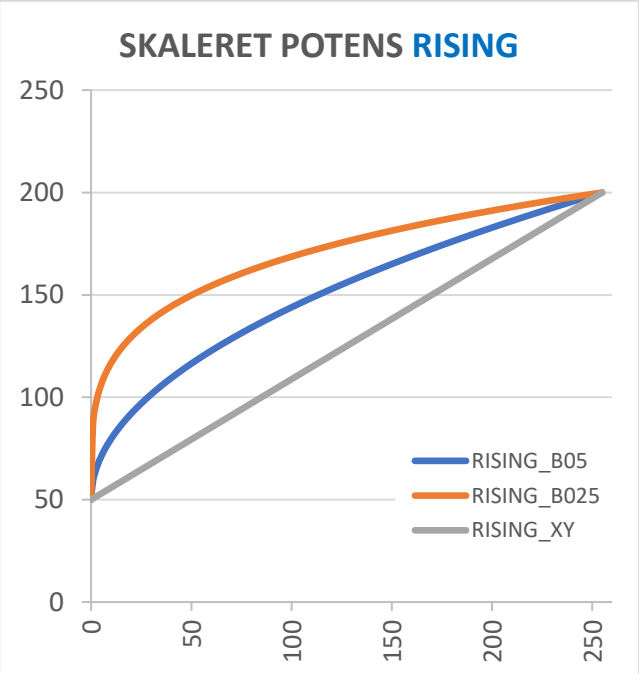
FALLING_SM5
RISING_E025
FALLING_E025

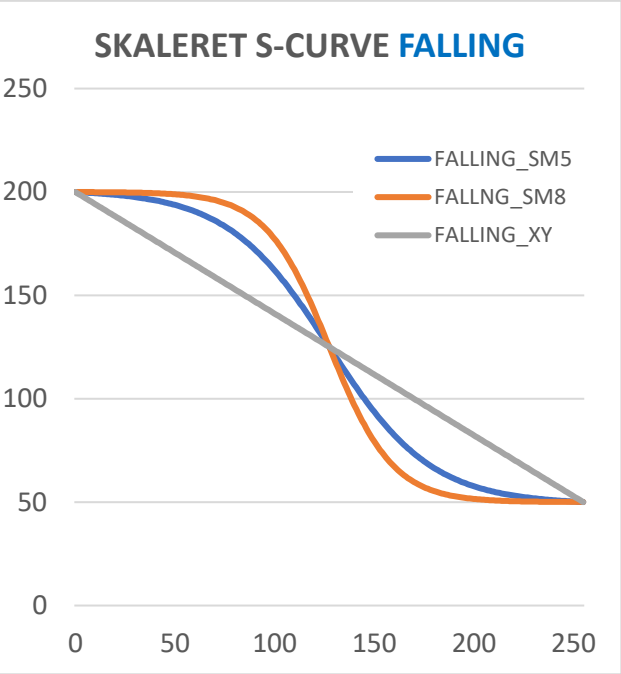
Alle kurver kan skaleres. Hvis værdien setPinBeginValue og værdien setPinEndValue er defineres skaleres hele kurven inden for tidsperioden outputTimeMS

Eksempler på skalering:

```
motor.activate (50, 200, RISING_XY);
motor.activate (50, 200, RISING_B05);
motor.activate (50, 200, RISING_B025);
```

```
servo.activate (200, 50, FALLING_XY);
servo.activate (200, 50, FALLING_SM8);
servo.activate (200, 50, FALLING_SM5);
```

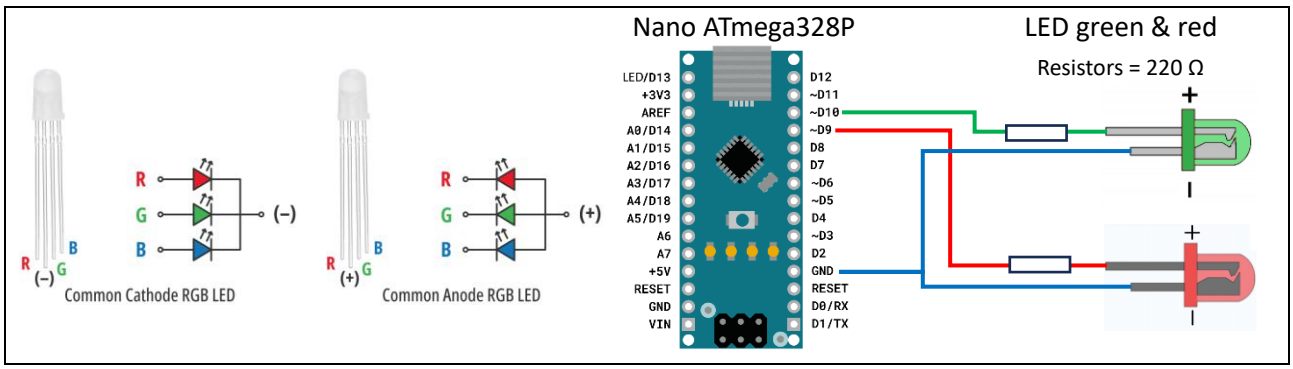




De enkelte beregnede punkter på kurven kan aflæses med funktionen: object_name.getOuputValue ();

Eksempler på configuration

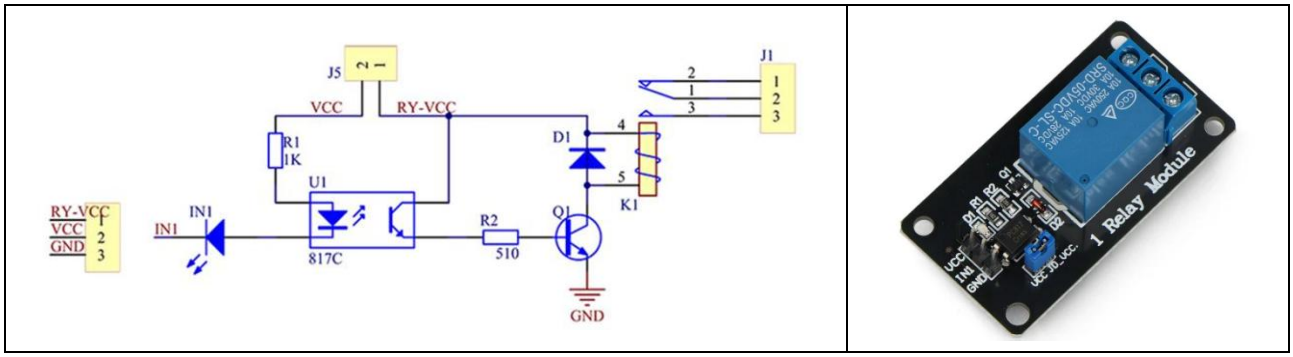
En multifarvet LED med fælles **katode** aktiveres ved at gå fra LOW til HIGH. PWM 0 -> 255 (fuld lys).
En multifarvet LED medfælles **anode** aktiveres ved at gå fra HIGH til LOW. PWM 255 -> 0 (fuld lys).



Eksempel på grøn LED der venter et halvt sekund og lyser herefter i et halvt sekund.

```
1. MTD2A_binary_output green_LED ("Green LED", 500, 500);
2. green_LED.initialize (10);
3. green_LED.activate ();
4. MTD2A_loop_execute ();
```

Et standard rellæ med optokobler der aktiveres med ved at gå fra HIGH til LOW.



Eksempel på et optokoblet rellæ, der venter et halvt sekund, og aktiveres herefter i et halvt sekund.
Alle standard værdier er omvendt af hvad der er default. Derfor skal alle parametre angives.

```
1. MTD2A_binary_output opto_relay ("Opto relay", 500, 500, 0, BINARY, LOW, HIGH);
2. opto_relay.initialize (12, NORMAL, HIGH);
3. opto_relay.activate ();
4. MTD2A_loop_execute ();
```

Alternativt `INVERTED` hvilket inverterer (omvender) alle pin output værdier.

```
1. MTD2A_binary_output opto_relay ("Opto relay", 500, 500);
2. opto_relay.initialize (12, INVERTED);
3. opto_relay.activate ();
4. MTD2A_loop_execute ();
```

Set og get funktioner

Set functions Green is default when no parameter	**Comment**
set_PinOutputMode ({ **BINARY** | P_W_M });	Select binary og PWM output
set_pinWriteToggl ({**ENABLE** | DISABLE});	Enable or disable pin writing
set_pinWriteMode ({**NORMAL** | INVERTED});	Invert all output (pin writing)
set_pinWriteValue ({BINARY {HIGH | LOW} / PWM {0-255}) , {BINARY | P_W_M});	Write directly to output pin (if enabled) **Optinal** parameter. No default
set_outputTimeMS ({0 - 4294967295})	Set new **output** time period milliseconds
set_beginDelayMS ({0 - 4294967295})	Set new **begin** delay time period millisec.
set_endDelayMS ({0 - 4294967295})	Set new **end** delay time period millisec.
set_outputTimer({ **STOP_TIMER** | RESET_TIMER})	Stop timer process immediately or reset
set_beginTimer ({ **STOP_TIMER** | RESET_TIMER})	Stop timer process immediately or reset
set_endTimer ({ **STOP_TIMER** | RESET_TIMER})	Stop timer process immediately or reset
set_debugPrint ({**ENABLE** | DISABLE});	Enable print phase number and text
set_errorPrint ({**ENABLE** | DISABLE});	Enable error messages

Get functions	**Comment**
get_processtState (); return bool {ACTIVE | COMPLETE}	Procedure process state.
get_pinWriteMode (); return bool {NORMAL | INVERTED}	Output is normal or inverted
get_pinWriteToggl (); return bool {ENABLE | DISABLE}	Write to pin is enabled or disabled
get_pinOutputValue() return uint8_t {HIGH | LOW} / {0 - 255}	Current output value
get_phaseChange (); return bool {true | false}	Momentarily phase change (one loop time)
get_phaseNumber (); return uint8_t {0 - 4}	Reset = 0, begin = 1, output = 2, end = 3, complete = 4.
get_setBeginMS (); return uint32_t milliseconds	Start time for **begin** proces
get_setOutputMS (); return uint32_t milliseconds	Start time for **output** proces
get_setEndMS (); return uint32_t milliseconds	Start time for **end** proces
get_reset_error (); return uint8_t {0-255}	Get error/warning number and reset number: Error [1 – 127] warning [128 – 255]

Operator overloading	**Function**
object_name_1 == object_name_2	bool processState_1 == processState_2
object_name_1 != object_name_2	bool processState_1 != processState_2
object_name_1 > object_name_2	bool processState_1 = ACTIVE & processState_2 = COMPLETE
object_name_1 < object_name_2	bool processState_1 = COMPLETE & processState_2 = ACTIVE
object_name_1 >> object_name_2	bool setOutputMS_1 > setOutputMS _2
object_name_1 << object_name_2	bool setOutputMS_1 < setOutputMS _2

print_conf();

object_name.print_conf ();

```
MTD2A_binary_output:
--------------------
  objectName   : LED_1
  processState : ACTIVE
  phaseText    : [3] End delay
  debugPrint   : DISABLE
  globalDebugPr: DISABLE
  errorPrint   : DISABLE
  globalErrorPr: ENABLE
  errorNumber  : 0 OK
  outputTimeMS : 2000
  beginDelayMS : 0
  endDelayMS   : 2000
  pinOutputMode: PWM
  PWMcurveType : 0
  pinBeginValue: 10
  pinEndValue  : 0
  pinNumber    : 9
  pinWriteToggl: ENABLE
  pinOutput    : NORMAL
  startPinValue: 0
  PinWriteValue: 0
  setOutputMS  : 2014
  setBeginMS   : 0
  setEndMS     : 4028
```