Troubleshooting of your BLDC motor

Tomas DRESLER
Ondrej HOLY
Tadeas HOLLER

life.augmented

Agenda

Presentation

Time		
0:00	Why troubleshooting?	Tomas Dresler
0:15	Prerequisites	
0:30	What do you need to validate?	
0:45	Tips for solving runtime issues	
1:15	Tips for tuning PI regulators	

Why troubleshooting?

Why troubleshooting?

- Each motor application is different:
  - Load, changing over time
  - Electrical parameters differ from application to application
  - The assumptions in the SDK target mainstream engines, not special ones
    - i.e., very low inductance windings (high-speed motors)
  - Very high load variations (no-load vs. heavy load on the clutch)

- Every designer was a beginner
  - And made some assumptions that were not right,
  - Or omitted some crucial detail,
  - Or reused old design without checking the setup too much etc.

Prerequisites

Prerequisites

- What do I need to successfully debug my motor application?
  - Multichannel Digital Signal Oscilloscope
    - with current probe
    - and several (high)-voltage probes
  - Insulation transformer(s) (for high-voltage applications)
  - Multimeter

- What else?
  - Board schematics
  - DAC outputs
  - Debugger (possibly insulated)
  - Serial to USB cable (possibly insulated)

Prerequsites

- DAC outputs
  - Display up to 2 run-time values of variables used in the FOC algorithm using oscilloscope
  - Allow immediate look at internal processes
  - I_{alpha}, I_{beta} show quality of current reading – if noisy or distorted, validate current reading path and setup t_{noise}, t_{raise} according to FOC SDK User Manual
  - b-emf$_{alpha}$, b-emf$_{beta}$, observed el. angle – if noisy or unstable, G2 is too high or measurement of R, L, K_e of your motor are wrong (within range of tens of %)
  - Options for DAC output supported by MC FOC SDK:

DAC	PWM	SPI
Must be supported by MCU and your HW Very precise	Needs RC filter Cheap, output has exponential delay	Requires external DAC Very precise

life.augmented

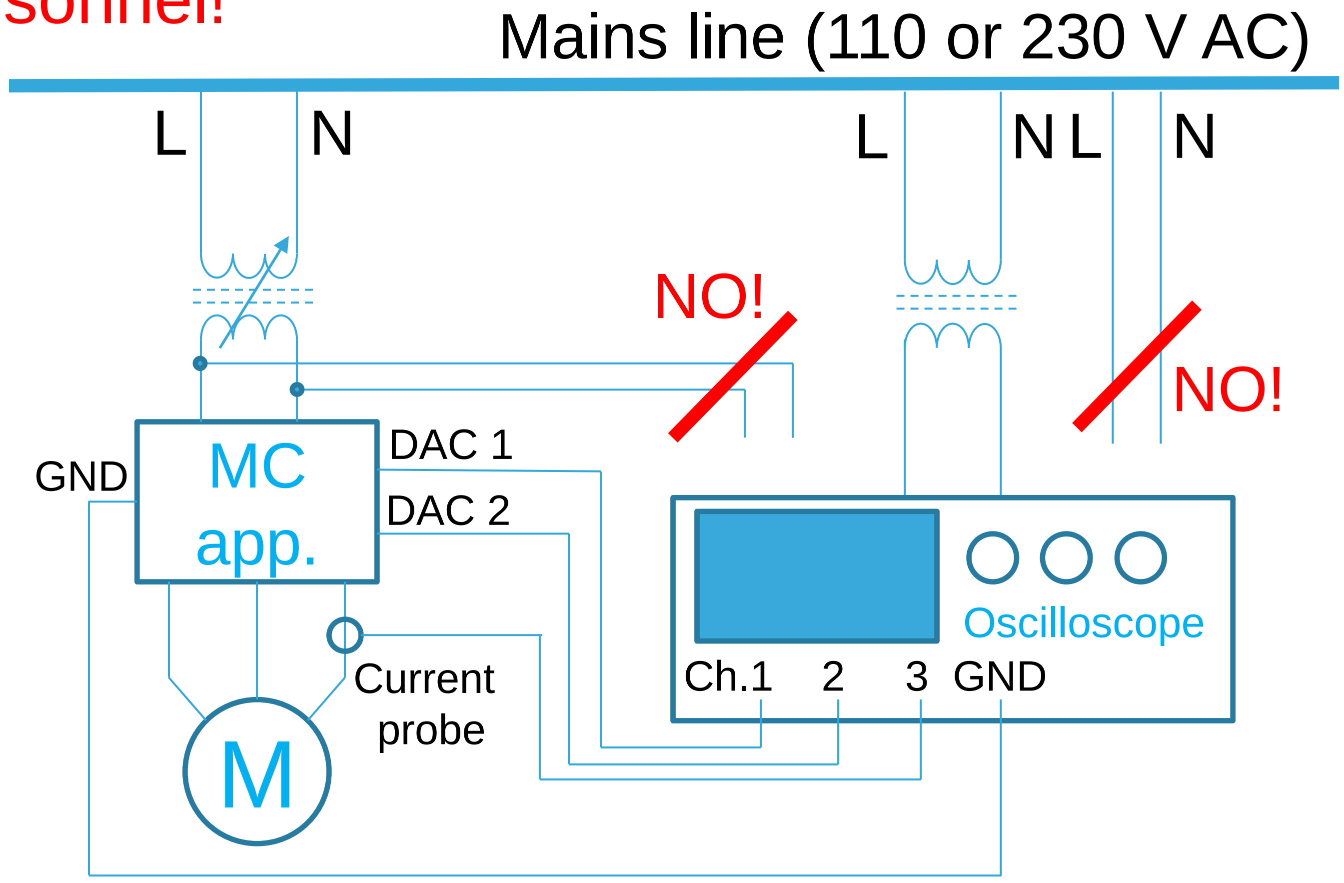
Prerequisites

- UART (via USB) connection allows natively
  - debugging with MC Workbench in Monitor mode,
  - changing runtime parameters (coefficients of many PI regulators, G2, required speed or torque etc.),
  - observing speed and tuning PI coefficients of speed regulator in runtime

Prerequisites

- Insulation transformer(s), oscilloscope usage in high-voltage systems

- Operate only by certified personnel!
  - Reduce risk of equipment damage
  - Reduce risk of injury or death
  - Only for mains (AC) rated applications



What do you need to validate?

Validation

- Before I switch the application on
  - Proper jumper setup and soldering switch configuration for:
    - Current reading topology (1-shunt, 3-shunt)
    - Speed sensor configuration (Hall, encoder, its derived supply: 5V/3.3V)
    - Overcurrent, over-temperature reaction mechanism
  - Polarity and rating of supply voltage, setup and range of DC-DC converter
  - Is isolation needed between mains and your application and your oscilloscope?
  - Does my power supply have current limiting option?
    Initially set to small value (tens of mA)!
  - Are my DAC outputs connected to oscilloscope?

Validation

- After I switch the application on
  - Is the supply current within a reasonable limit?
    Without motor running, only few tens of mA are suggested.
    Only after this check, set current limitation to operating limit!
  - Is your application alive and communicating (via UI, USART etc.)?
  - When I start the motor, is the overcurrent reported by power supply or by overcurrent mechanism of the power stage?
  - Do I see the currents reported by DAC outputs on my oscilloscope when my motor runs?

Tips for solving runtime issues

Tips for solving runtime issues

- Initial application engineer’s decision tree:
  1. If it burns, check HW
  2. If it doesn’t move, check power stage setup (PWM, ADC, OC)
  3. If it moves a little, check alignment, start-up and rev-up
  4. If it moves, but fails at speed change, check G2 and Speed PI regulator
  5. If it moves, but speed is unstable, check Speed PI regulator
  6. If it moves well, you’re finished! Congratulations!

- Application design
  1. Does it need speed control? Use torque mode!
  2. Does it really need speed control? Use speed mode, but tune Speed PI well!

Tips for solving runtime issues

- I don’t see changes of my SW in the application
  - Did you generate updated header files in correct folder or placed them there?
  - Did you recompile your SW after applying the changes?
  - Did you load new SW to your application?
  - Isn’t the debugger the other USB cable connected you your PC?

Tips for solving runtime issues

- Does your application read, evaluate and react on SDK error states?
  - If not, rethink your SW and ask the question again!

- “FOC duration” error shows up, motor doesn’t move at all
  - The PWM frequency is too high for CPU to handle FOC algorithm in time
  - Increase “FOC execution rate” by one in Drive settings
  - Revalidate your IRQ priorities – MC FOC SDK must always have the highest interrupt priority of all! Your motor depends on it!

Tips for solving runtime issues

- Best PWM frequency?
  - Low enough to reduce switching losses (down to ~8 kHz)
  - High enough to manage low inductance motors (up to ~30 kHz) and above acoustic noise band (human ear can listen up to 22 kHz)
  - CPU load can be reduced by increasing “FOC execution rate”

- Best start-up settings (start with, tune to your application later)
  - Speed ramp duration 3000 ms
  - Speed ramp final value 30% of maximum motor speed
  - Current ramp final value 50% of nominal motor current
  - Include alignment (2000 ms, 50% of nominal motor current)
  - Minimum start-up speed set as 15% of maximum motor speed

Tips for solving runtime issues

- “Over current” detected immediately at start-up
  - Wrong current sensing topology – single-shunt instead of three-shunt? Did you check your jumpers?
  - Isn’t power stage damaged by overcurrent? Check your MOSFETs for short circuit!
  - Verify and correct current path gain, ADC input, polarity of PWM outputs
  - The current regulation bandwidth is too high – in Drive settings, reduce current regulation bandwidth down to 2000 rad/s for 3-shunt and 1000 rad/s for 1-shunt. Maximum value is 9000 rad/s, resp. 4500 rad/s.
  - PWM frequency is too low – phase current may rise too quickly esp. in low-inductance motors – increase PWM frequency, to avoid “FOC duration” error see previous slide

Tips for solving runtime issues

- "Speed feedback" appears after sudden acceleration or deceleration

  The runtime uses speed variation for detecting anomalies, thus big speed change over short time can trigger this error

  - Use speed ramps to reduce acceleration
  - Observer gain G2 is too high and speed estimation generates too much noise:
    - Decrease G2 by 2, 4, 6, 8 and retry…
    - Detect by displaying b-emf alpha and beta on the DACs – they shall be sinusoidal. Too much noise or distortion means too high G2 estimation
  - Test in torque mode. If this mode runs stable, speed PI regulator needs re-tuning
  - Too low variance threshold. Increase gradually up to 100% or increase speed FIFO depth (as power of 2).
    Beware, this parameter significantly reduces sensitivity for rotor locked condition detection!

Tips for solving runtime issues

- "Speed feedback" appears immediately after start-up
  - Start-up validity can be detected too early. Return to Start-up parameters and increase # of successful consecutive start-up tests (not more than 4-5) and minimum start-up speed to 15% of maximum motor speed
  - Use Rev-up algorithm to align gradually forced torque ramp with sensor-less control algorithm output – this avoids sudden speed change at the transition between open loop and closed loop

- "Start-up failure" appears after motor initially moves, but stops before Rev-up
  - The torque (current) at higher start-up speed is insufficient to accelerate to validation speed
    - Decrease acceleration rate – set speed ramp final value to 30% of motor max. speed
    - Increase start-up current – from ~50% up to 100% of nominal current
    - Enable alignment phase for more deterministic start-up or use advanced start-up

Tips for solving runtime issues

- My motor doesn’t start immediately
  - You have opted for alignment – motor waits for stabilization in specific position
  - “On-the-fly start-up” has been selected – detection phase has started and waits some time for rotation signal – if none is found, standard alignment and start-up applies

Tips for solving runtime issues

- Speed or control is unstable below 5% of maximal motor speed
  - The sensorless algorithm is stable down to 5-7% of maximum speed. Below that speed the noise from current reading is too high for proper rotor position estimation.
  - Solution is twofold:
    1. Use Hall sensors for low speed region and sensorless at medium-to-high speeds
    2. Use HFI sensorless algorithm (only available for anisotropic motors and MCUs with FPU)
- Speed control is unstable at various speeds
  - Load changes in non-linear fashion
  - May be too light at low and high speeds (i.e. air pump) and heavy at medium speeds
  - May require different Speed PI tuning at different speeds with linear interpolation between the regions

Tips for tuning PI regulators

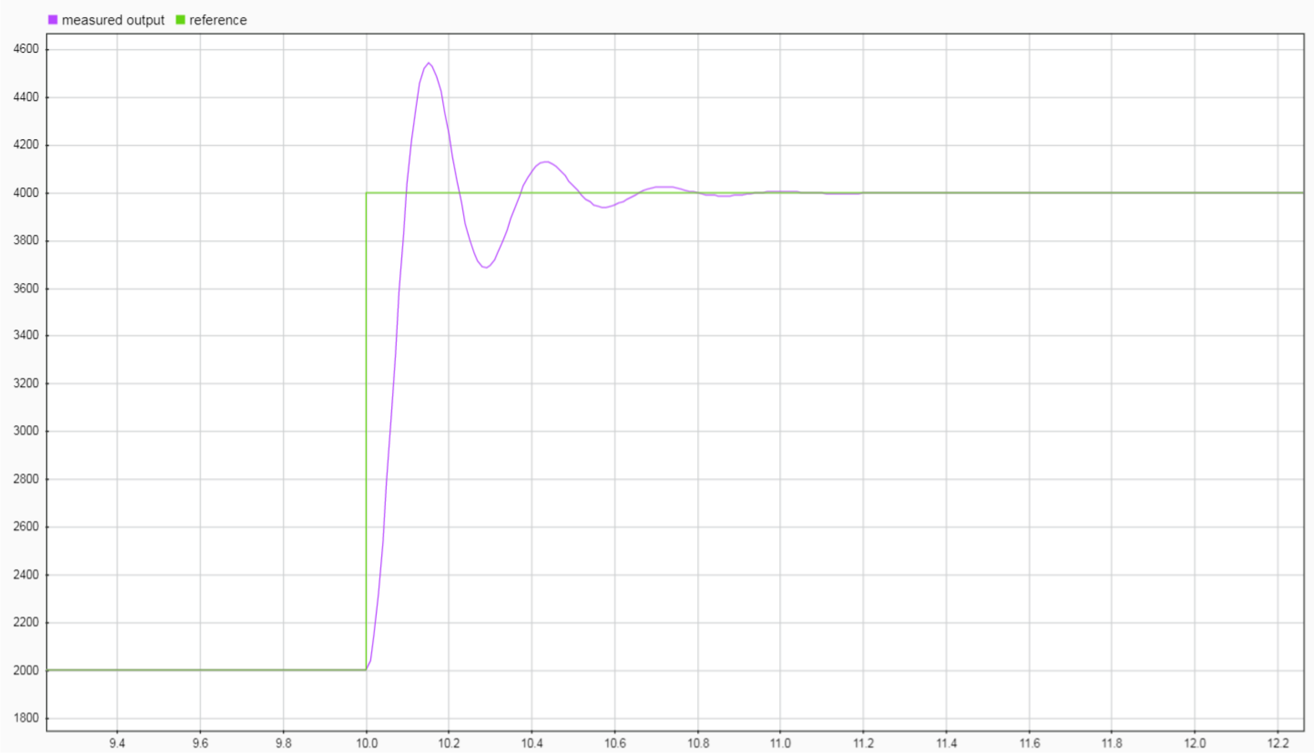
life.augmented

Tips for tuning PI regulators

- There are many PI(D) regulators in the SDK:
  - I_q, I_d current regulators within FOC model
  - PLL PI in the position/speed reconstruction block
  - Flux weakening PI regulator
  - Speed PI regulator
- Most of them are tuned by Workbench, but response of some of them depends on the physical application around the SW model, i.e.
  - Speed PI depends on the load inertia
  - I_q, I_d current regulators depend on power stage, inductance, PWM frequency

Tips for tuning PI regulators

- There exist many methods for proper tuning, let’s start with manual
- The tuning requires a step change in the required quantity (speed, load, current) and measurement of the regulated quantity response
- An example is change of the required speed as a step from 2000 RPM to 4000 RPM. The response is the real speed over time:



life.augmented

Tips for tuning PI regulators

- On previous slide, we saw damped oscillation response – this may be unwanted behavior, so let’s mention some rules of thumb!

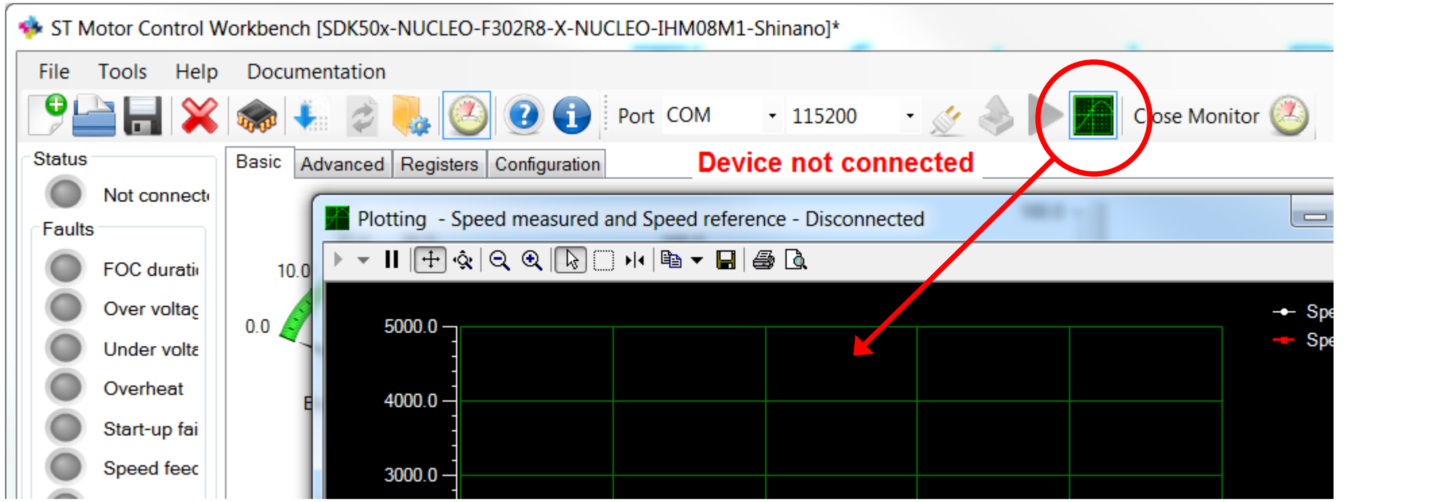
- If there is an overshoot, change the ratio K_p/K_i to avoid it or vice versa

- If the shape is exponentially closing to the required value, keep the ratio K_p/K_i, but proportionally change both K_p and K_i to increase or decrease the slope.

- Try this at different speeds and loads and choose conservative estimation – avoid undamped oscillation at all costs!

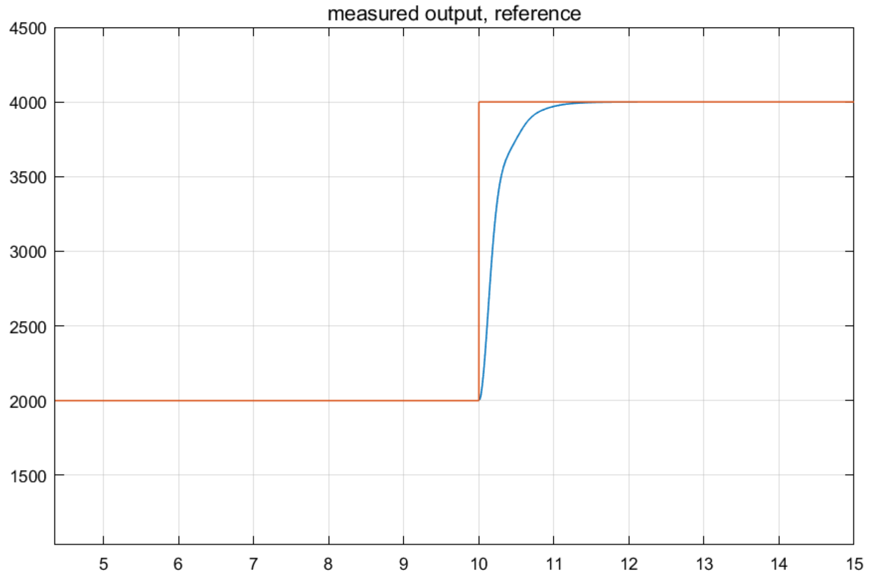
Tips for tuning PI regulators

- Such chart can be displayed in MC workbench, Monitor mode, via Plotter window



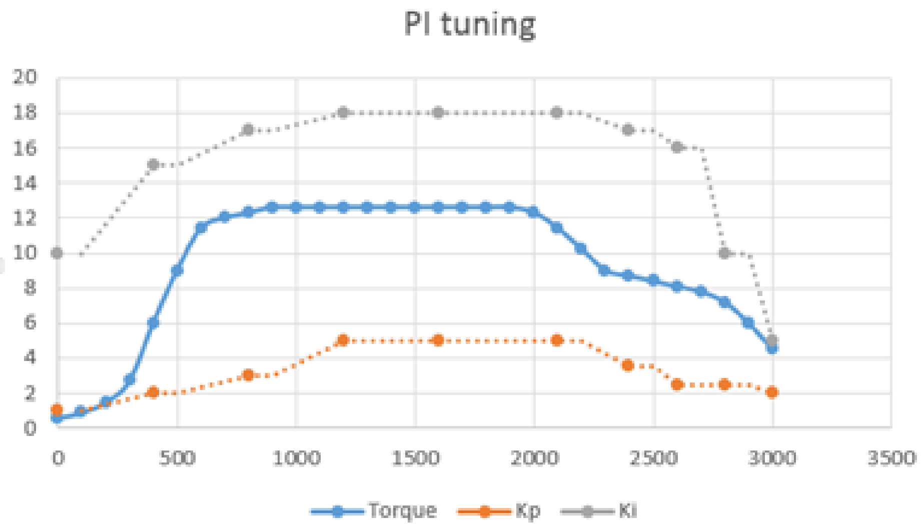
- Otherwise use DACs and oscilloscope

- Optimum reaction, if overshot is undesired, is on the following picture:



Tips for tuning PI regulators

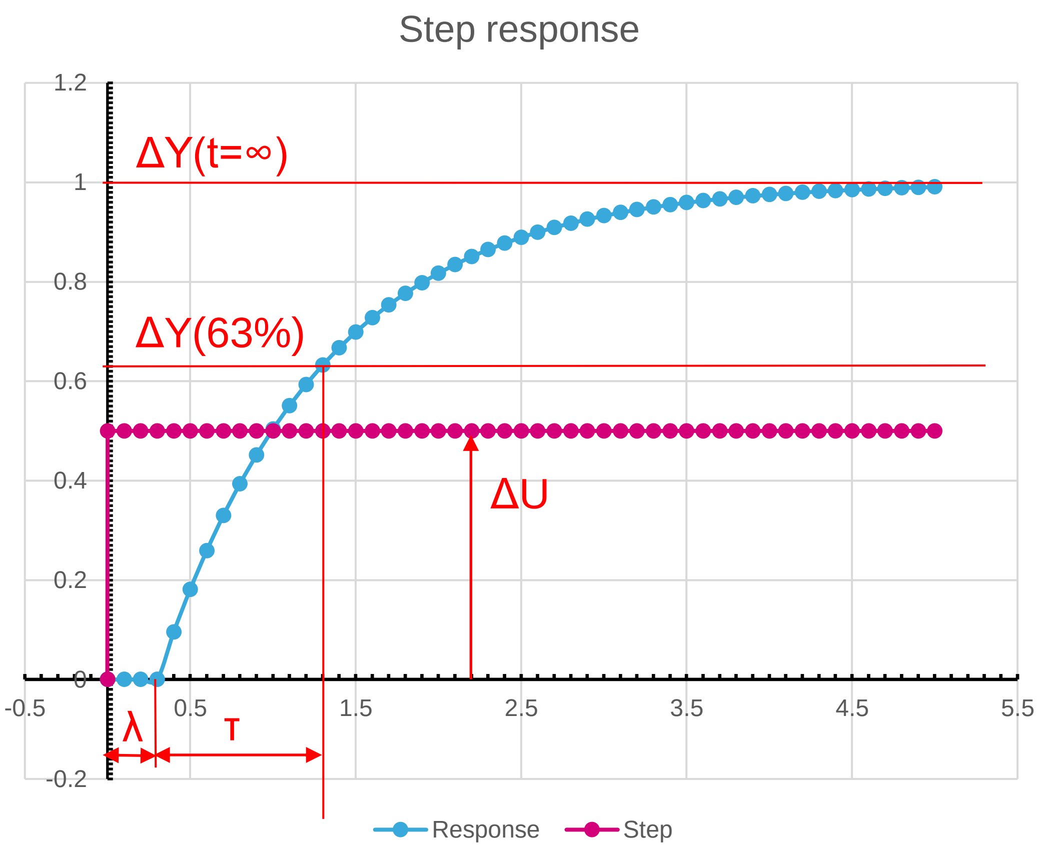
- If the load torque changes with speed, different tuning may be needed at different speeds. An example of optimal value of K_p, K_i vs. speed is on the following picture:



- Such K_p, K_i interpolation may need to be implemented in user SW for the Speed PI

Tips for tuning PI regulators

- Another more analytical methods need some measurements in the torque mode:
  - Determine stabilized speed response Y(t=∞) on torque step U:
  - Determine Process dead time (λ)
  - Determine Time constant (τ) when speed crosses 63% of the speed step (between stabilized value and initial speed)
  - Read PI process frequency f_s (usually 2 kHz)



Tips for tuning PI regulators

- Typically the motor with connected load is a first order system (with simple exponential response) with process delay λ

- In such case, simple calculation is needed:
  - Process gain $K = \frac{\Delta Y(t=\infty)}{\Delta U}$
  - Here the ΔY has unit of *dpp* (digits per PWM) and ΔU is change in Compare register of MC timer

- Let's consider PI(D) regulator of this form:

$$y(s) = K_p\left(1 + \frac{1}{T_i \cdot s} + T_d \cdot s\right)e(s)$$

Tips for tuning PI regulators

- Ziegler-Nichols method (overshoot, aggressive)

Type	K_p	τ_i	τ_d
P	$\frac{1}{K}\frac{\tau}{\lambda}$	∞	0
PI	$\frac{0.9}{K}\frac{\tau}{\lambda}$	$3.33 \cdot \lambda$	0
PID	$\frac{1.2}{K}\frac{\tau}{\lambda}$	$2.0 \cdot \lambda$	$0.5 \cdot \lambda$

Tips for tuning PI regulators

- Cohen-Coon method (moderate to conservative)

Type	K_p	τ_i	τ_d
P	$\frac{1}{K}\frac{\tau}{\lambda}\left(1+\frac{1}{3}\frac{\lambda}{\tau}\right)$	∞	0
PI	$\frac{1}{K}\frac{\tau}{\lambda}\left(0.9+\frac{1}{12}\frac{\lambda}{\tau}\right)$	$\lambda\left(\frac{30+3\frac{\lambda}{\tau}}{9+20\frac{\lambda}{\tau}}\right)$	0
PID	$\frac{1}{K}\frac{\tau}{\lambda}\left(\frac{4}{3}+\frac{1}{4}\frac{\lambda}{\tau}\right)$	$\lambda\left(\frac{32+6\frac{\lambda}{\tau}}{13+8\frac{\lambda}{\tau}}\right)$	$\lambda\left(\frac{4}{11+2\frac{\lambda}{\tau}}\right)$

life.augmented

Tips for tuning PI regulators

- Conversion to discrete PI(D) regulator with sampling frequency f_s

$$K_p' = K_p$$

$$K_i' = \frac{K_p}{\tau_i}\frac{1}{f_s}$$

$$K_d' = K_p \tau_d f_s$$

- Calculated values shall be expressed as ratios $\frac{K_x'}{2^N}$ and put in MC workbench, K_x' being in 16-bit range (±32767)

Tips for tuning PI regulators

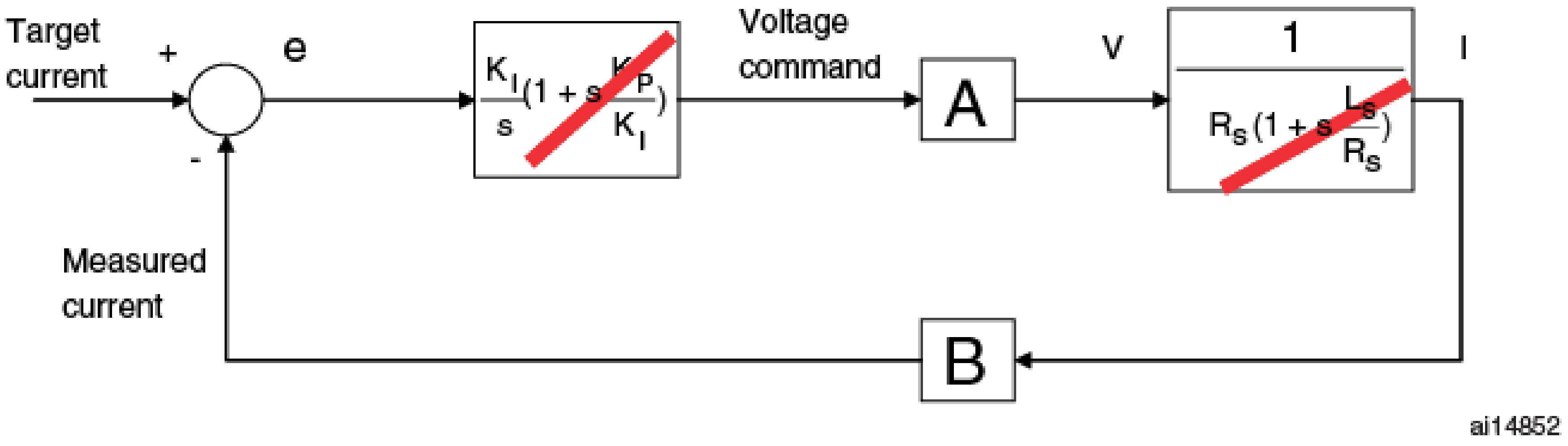
- After calculating the PI coefficients, one still needs to tune them manually

- These equations are valid in the range $0.1 < \frac{\lambda}{\tau} < 1$, otherwise other type of tuning is needed

Tips for tuning PI regulators

- Another method works in time domain with a priori knowledge of motor and load inertia (~L_s) a mechanical resistance (~R_s)

- By substituting K_P/K_I with L_S/R_S ratio, one can perform pole-zero cancellation as shown below, calculating K_P as on previous slide:



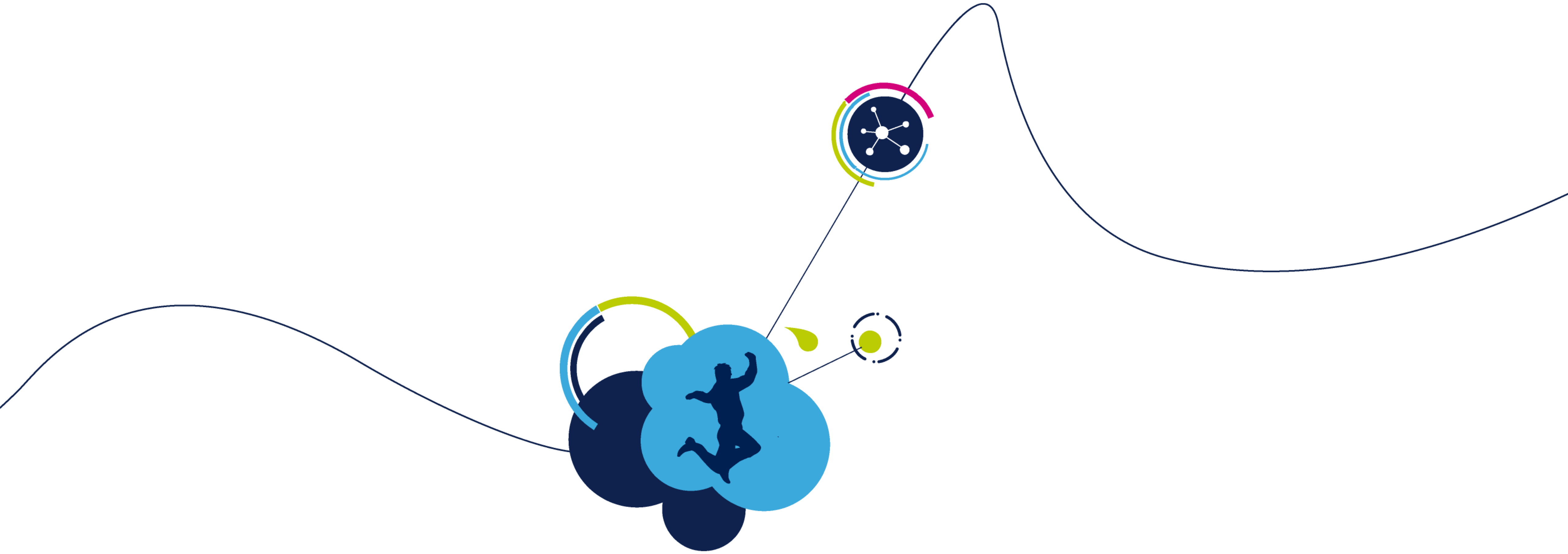
Tips for tuning PI regulators

- In case of second-order and more complex or interlinked systems (combined exponential behavior), the response measurement and calculation are complex and discussed in control theory literature, i.e. zero-pole mapping regulators
- Such example shall use PID regulator with derivative component or even more complex (polynomial) regulators
- Small filter with $\tau' = 0.1\tau$ for the error component may be introduced before PI(D) regulator, too

Tips for tuning PI regulators

Used literature:

- http://educypedia.karadimov.info/library/pidtune2.pdf
- http://www.kirp.chtf.stuba.sk/moodle/pluginfile.php/66882/mod_resource/content/0/tidsdiskret_pid_reg.pdf
- https://pdfs.semanticscholar.org/116c/e07bcb202562606884c853fd1d19169a0b16.pdf
- Matlab: pidtune, pidtool
- https://www.biricha.com



Thank you!

life.augmented