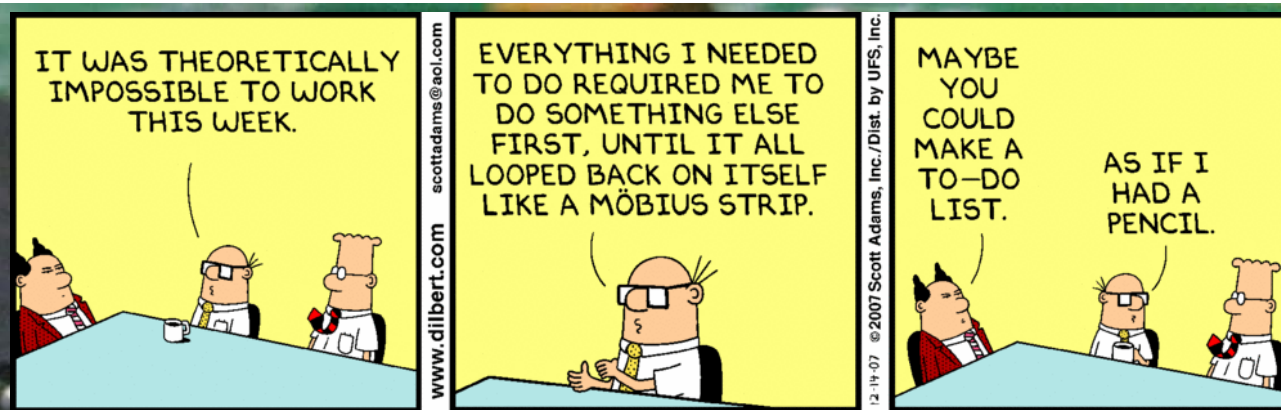
Lucerne University of Applied Sciences and Arts
HOCHSCHULE LUZERN
Technik & Architektur

Source: Dilbert.com



Closed Loop Control: PID & Turn

"Control works best with a loop-back…"

Prof. Erich Styger
erich.styger@hslu.ch
+41 41 349 33 01

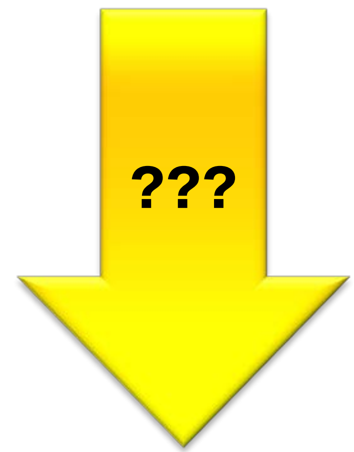
Scriptum:
Closed Loop Control

Learning Goals

- Crash-Course
  - Closed Loop Control
  - PID

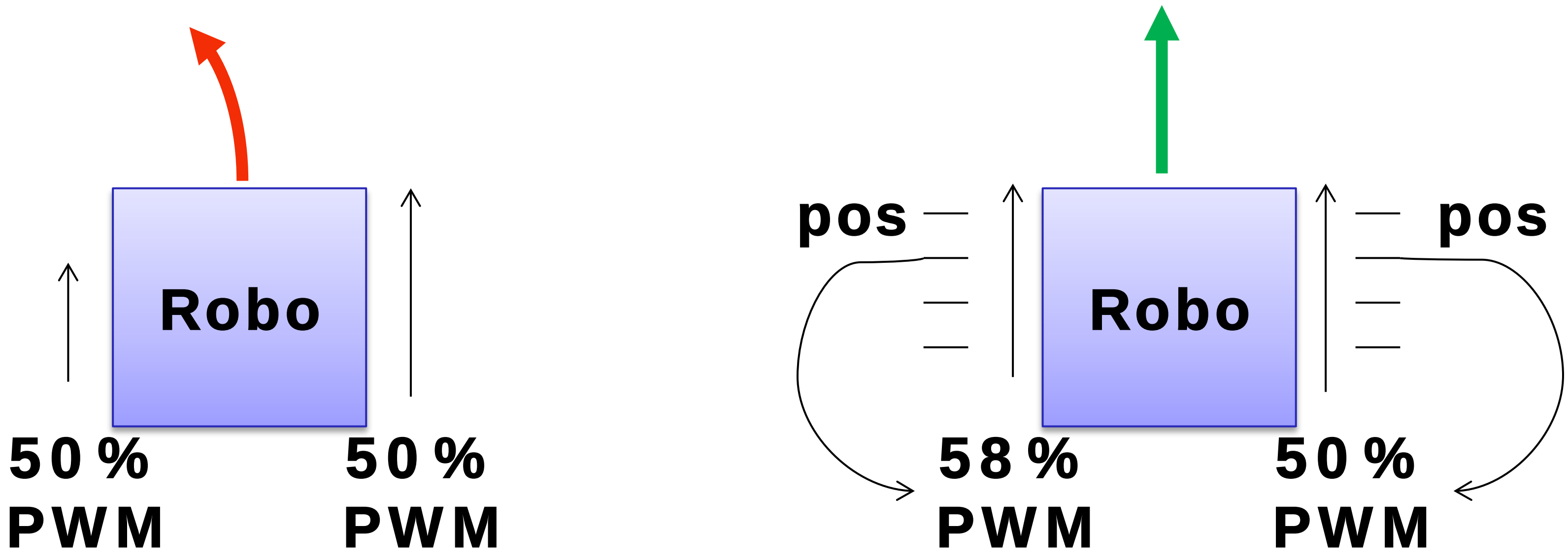
- ‘control’ vs. ‘closed loop control’

- Terminology
  - Control
  - Closed loop control
  - Plant (‘Regelstrecke’)
  - Step Response (‘Sprungantwort’)
  - PID Controller



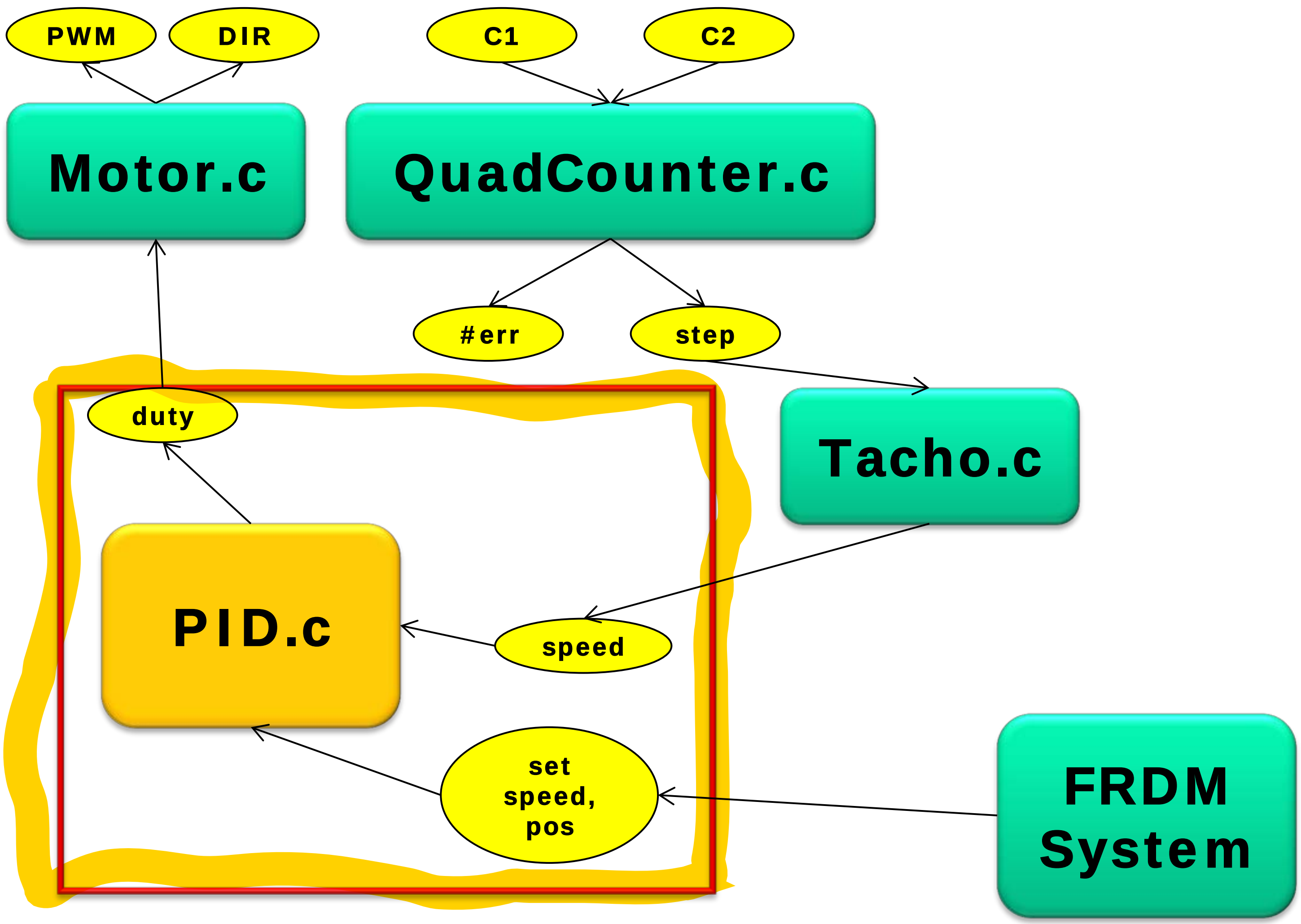
Problem and Solution

- Problem
  - Same PWM ➔ different speed
  - Mechanical: Friction, Gear Play, ...
- Solution
  - Use Position Encoders to provide feedback

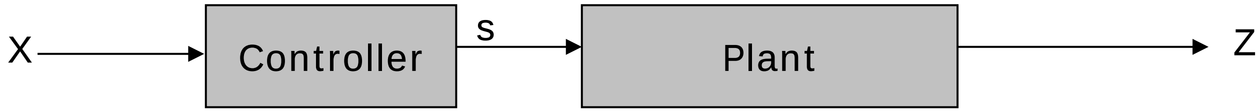


Lucerne University of
Applied Sciences and Arts
HOCHSCHULE
LUZERN
Technik & Architektur

High Level Overview



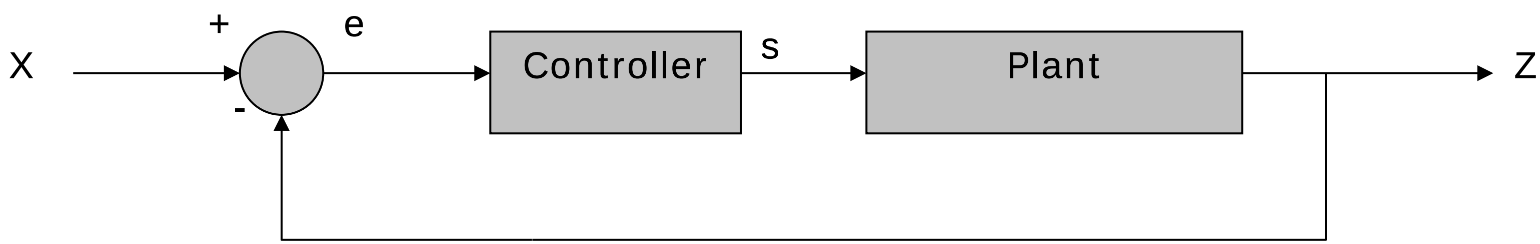
Control



- **Transmission** of Information
- **Processing** of Information
- Sensors as **Information Producer**
- Actuators as **Information Consumer**

- Properties
  - Open Loop
  - No Feedback

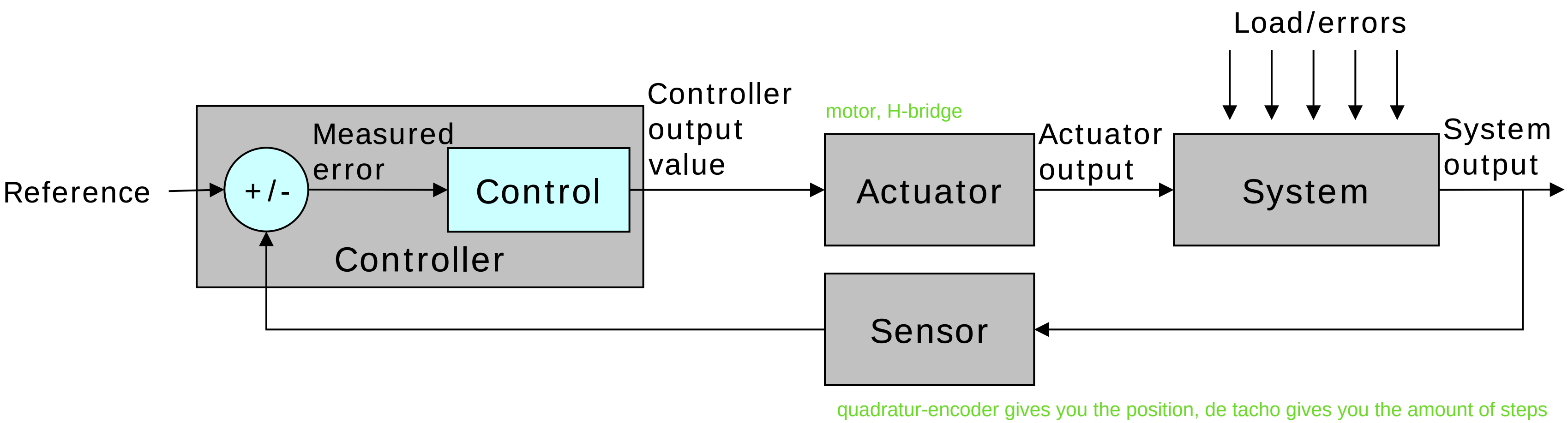
Closed Loop Control (vs. Control)



- Closed Loop
- Formal model for
  - Measure
  - Compare
  - Control
- Input is desired value ‘*X*‘
- Measure actual value ‘*Z*‘
- Calculate error *‘e’* as Z-X
- Processes control algorithm in controller
- Send new set value ‘s‘ to plant

English:
‚**Control**‘ für Steuern und Regeln verwendet.
‚**Closed loop control**‘ entspricht unserem Regelungsbegriff

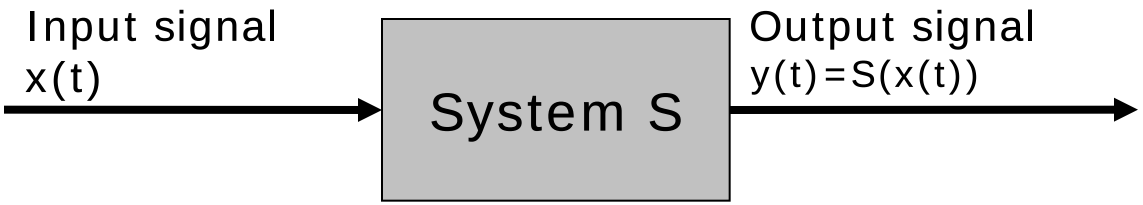
System Architecture



- Generation of controller output values for actuator
- Measure sensor values
- Generate new controller output values
- Defined behavior of the system
- Consideration of errors

Definition: Input and Output Signals

- Input value $x_1(t)$:
  - System generates output value with function S: $y_1(t) = S(x_1(t))$
- Result: formal description of the system
  - Controller as linear differential equation system
  - Description of the closed loop control system

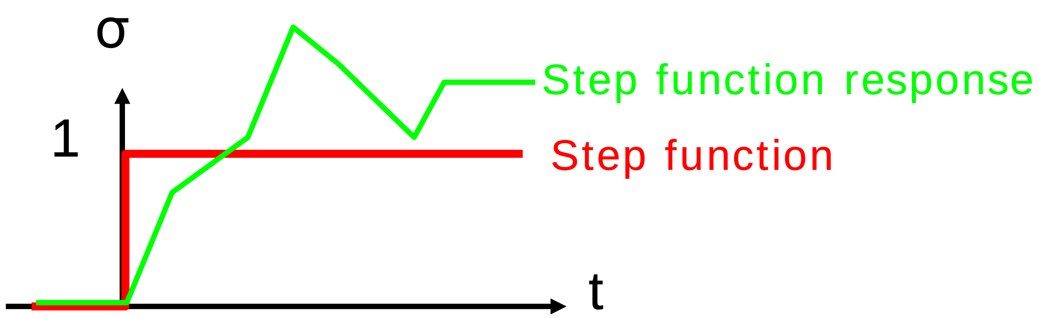


Need input/start values of x(t) for the equation solution.

Definition Input Values

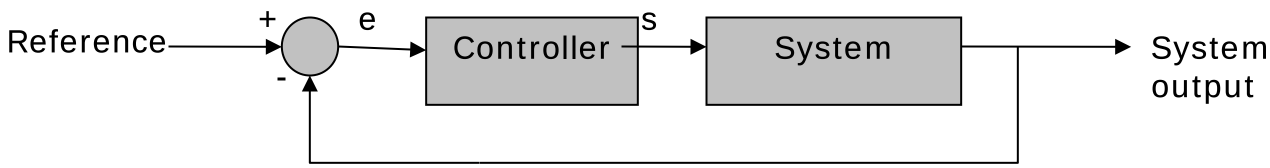
- Need **input values** to determine the behaviour
  - Calculation of x(t)
  - Allows to **solve** the equation
- Typical function: **step function**
  - Heaviside step function (0->1)
  - Simple ☺

$$\sigma(t) = \begin{cases} 0 \text{ for t} \leq 0 \\ 1 \text{ for t} > 0 \end{cases}$$



Output value S(σ(t)) is the **Step Function Response**

PID Controller Base Types

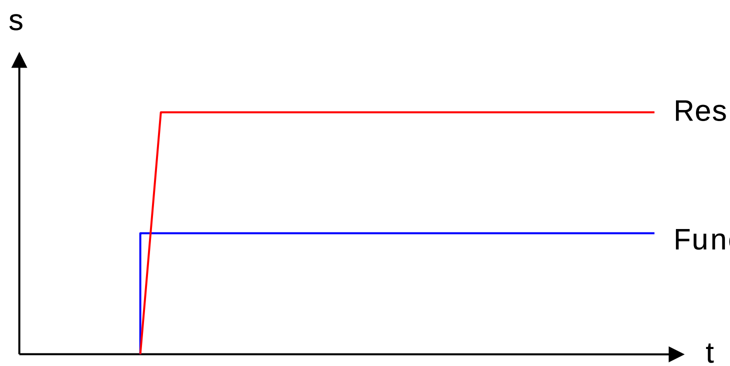


- Different controller types to affect behavior
- 3 base types
  - P: Proportional term
  - I: Integral term
  - D: Derivative term

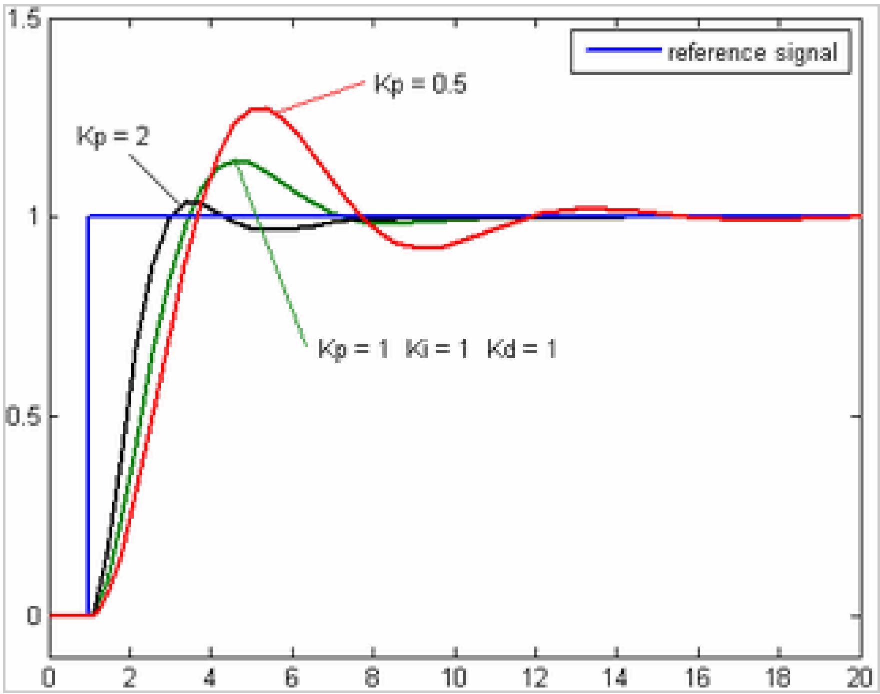
Combination of all three aspects:
PID Controller

Proportional / Gain Term

- Simple amplification of the current error value



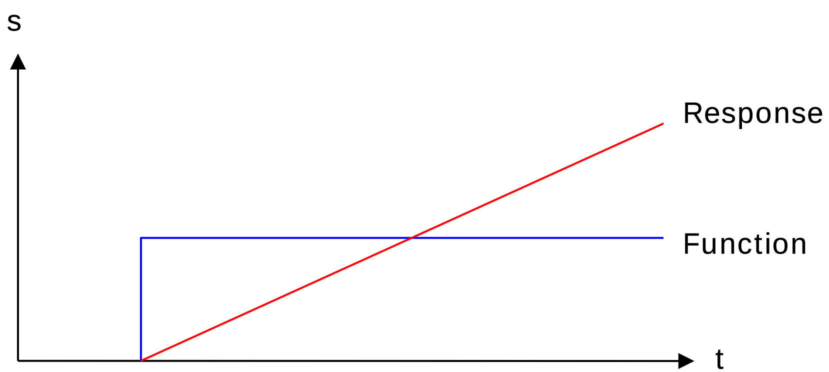
$$s(t) = K_p \bullet e(t)$$



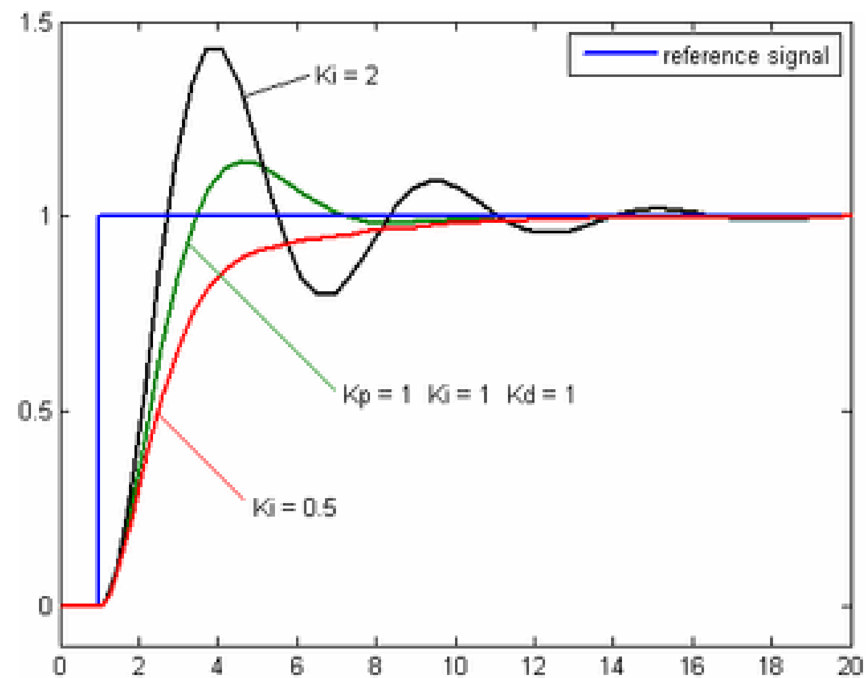
Source: Wikipedia

Integral Term

- Proportional to the magnitude of error and the duration of the error
- Errors are accumulated over time (integration of the error) => Anti-Windup!!!



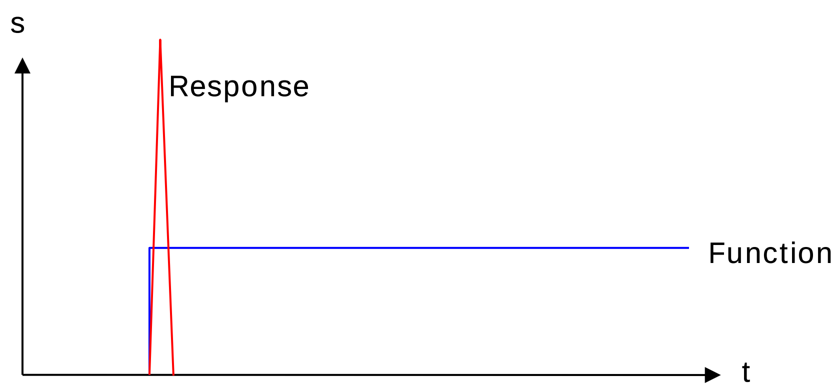
$$s(t) = K_i \int_0^t e(t') \bullet dt'$$



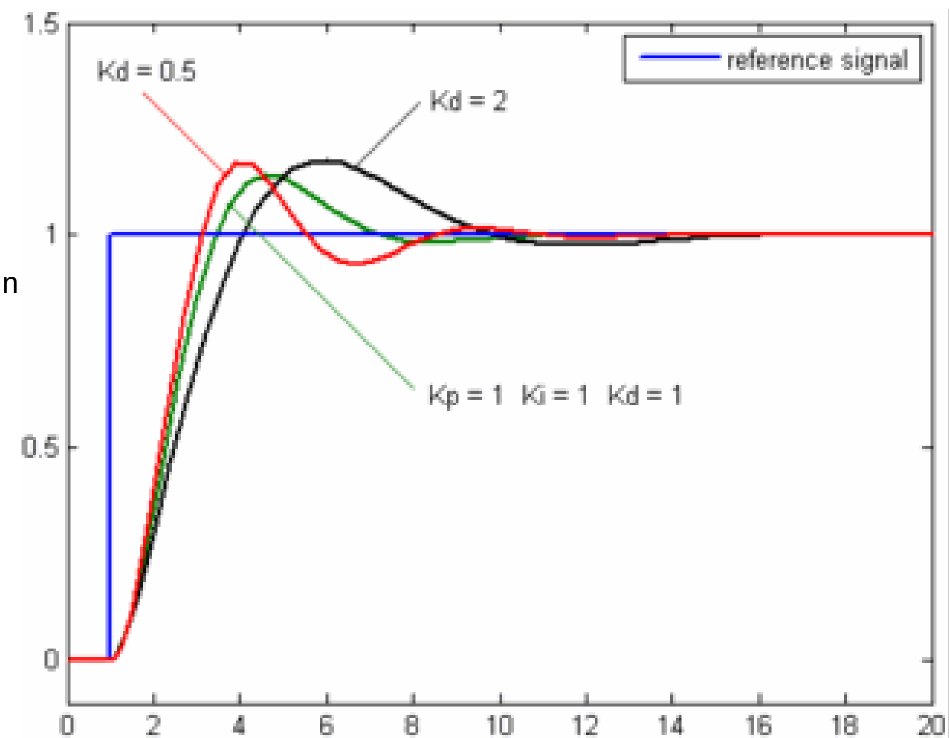
Source: Wikipedia

Derivative Term

- Slope of error (first derivative with respect to time)
- Used to reduce the magnitude of the overshot



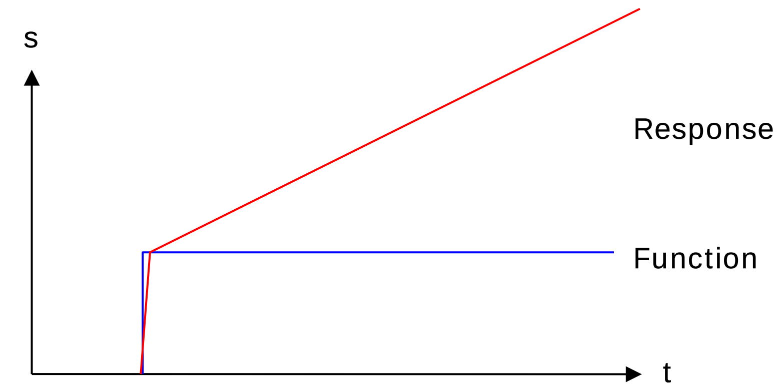
$$s(t) = K_d \frac{d}{dt} e(t)$$



Source: Wikipedia

PI Controller

- Combination P und I terms
- P: Speed
- I: Accuracy

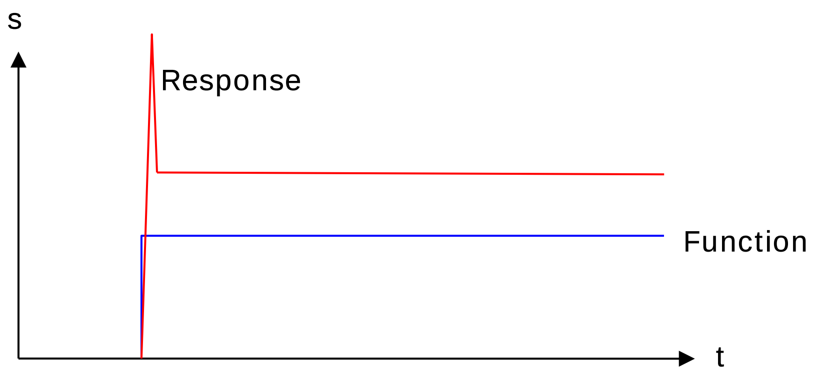


$$s(t) = K_p \bullet e(t) + K_i \int_0^t e(t') \bullet dt'$$

```
Software Controller:

esum += e;
s = Kp*e + Ki*Ta*esum;
```

PD Controller

- Combination P and D term
- P + D: makes it fast



$$s(t) = K_p \bullet e(t) + K_d \frac{d}{dt} e(t)$$

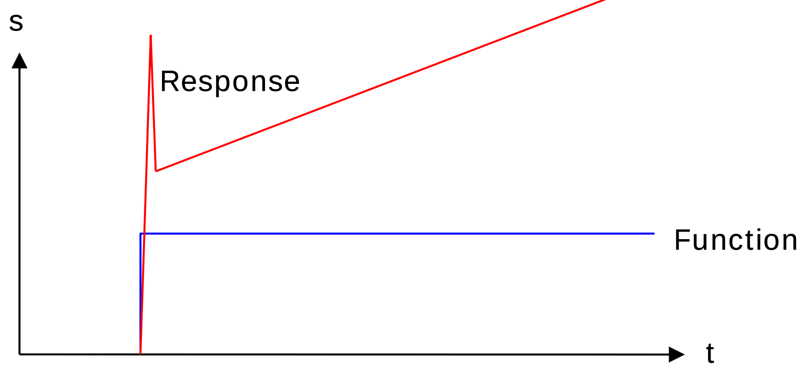
```
Software Controller:

s = Kp*e + Kd*(e-eprev)/Ta;
eprev = e;
```

could be eleminted with the sample frequency

PID Controller

- Combination P, I and D term
- P + D: speed
- I: accuracy

Genauigkeit



$$s(t) = K_p \bullet e(t) + K_d \frac{d}{dt} e(t) + K_i \int_0^t e(t') \bullet dt'$$

```
Software Controller:

esum = esum+e;
s = Kp*e + Ki*Ta*esum + Kd*(e-eprev)/Ta;
eprev = e;
```

Microcontroller PID Controller

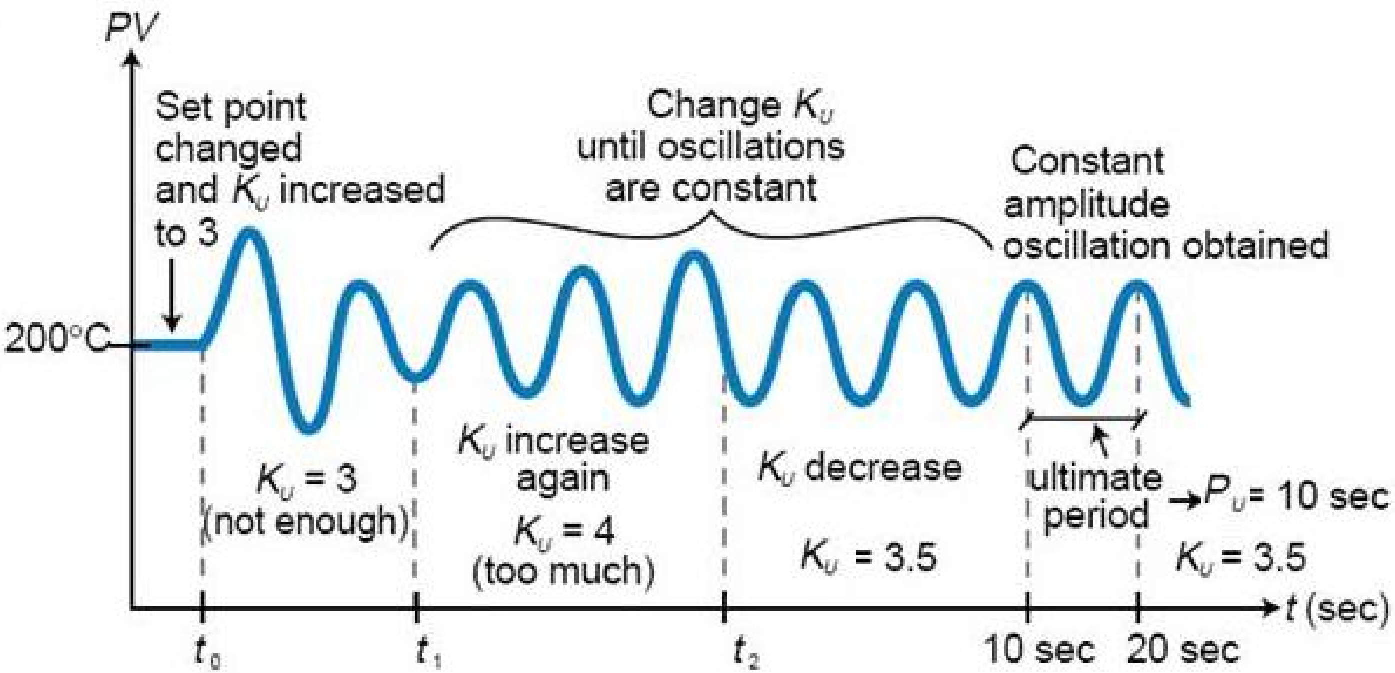
```
previous_error = 0
integral = 0
start:
 error = setpoint - actual_position
 integral = integral + (error*dt)    dt is the sampling time
 derivative = (error - previous_error)/dt
 output = (Kp*error) + (Ki*integral) + (Kd*derivative)
 previous_error = error
 wait(dt)
goto start
```

it make no sense to run this one faster then your system can react!

- I term used with appropriate sampling time
  - Small enough ➔ can be optimized out
  - Depends on sensor
  - Need „Anti-Wind-Up“
- Minimizing dead time
  - Appropriate Timing
  - Example: PWM period

Empirical PID Tuning: Ziegler-Nichols

- Only P: increase Kp until constant amplitude
- Decrease Kp and increase Ki and Kd
- Increase Anti-Windup to reach the goal set point

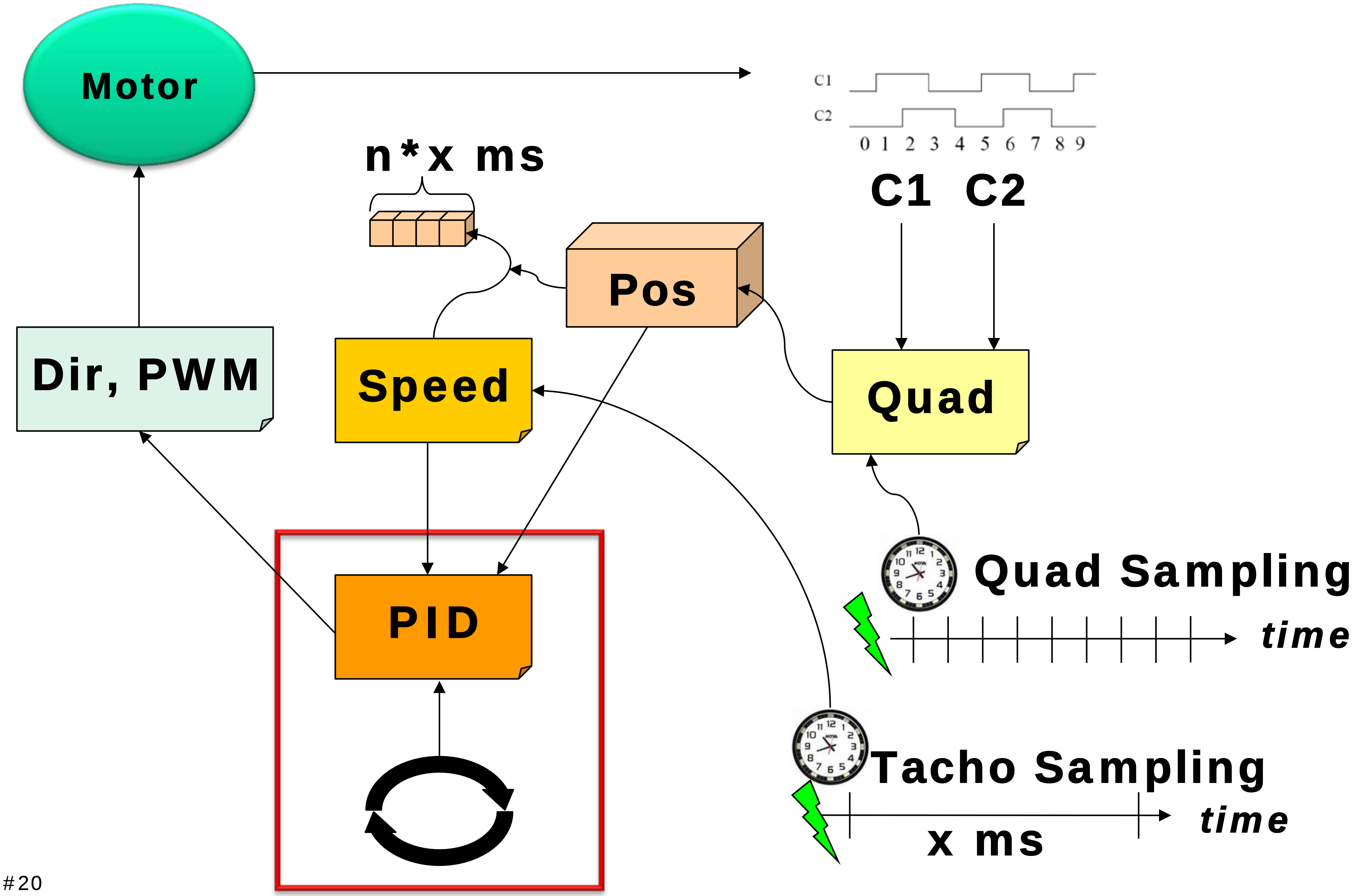


Source: PID Tuning Classical

	K_c	T_I	T_D
P	$K_U/2$		
PI	$K_U/2.2$	$P_U/1.2$	
PID	$K_U/1.7$	$P_U/2$	$P_U/8$

Lucerne University of
Applied Sciences and Arts
HOCHSCHULE
LUZERN
Technik & Architektur

System Architecture & Timing



Lucerne University of
Applied Sciences and Arts
HOCHSCHULE
LUZERN
Technik & Architektur

PID Configuration

- Max speed command
- Limiting speed to max value
- Forward (fw) line following configuration

```
pid                             ; Group of PID commands
  help|status                   ; Shows PID help or status
  speed (L|R) (p|d|i|w) <v; Sets P, D, I or      factor 100 (1.0 is 100) no floatingpoint
                                  anti-windup position value
  speed (L|R) speed <value; Maximum speed % value
  pos (L|R) (p|d|i|w) <val; Sets P, D, I or anti-windup
                                  position value
  pos speed <value>             ; Maximum speed % value
  fw (p|i|d|w) <value>          ; Sets P, I, D or anti-Windup
                                  line value   line following
  fw speed <value>              ; Maximum speed % value
```

PID Interface

```
void PID_Speed(int32_t currSpeed, int32_t setSpeed, bool isLeft);

void PID_Pos(int32_t currPos, int32_t setPos, bool isLeft);

void PID_Line(uint16_t currLine, uint16_t setLine);
```

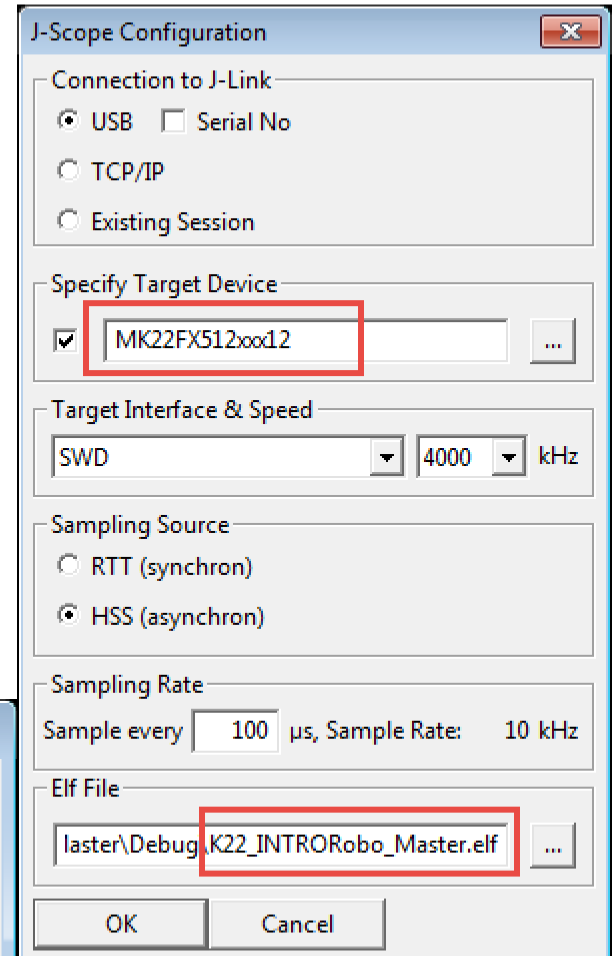
ignore at the moment

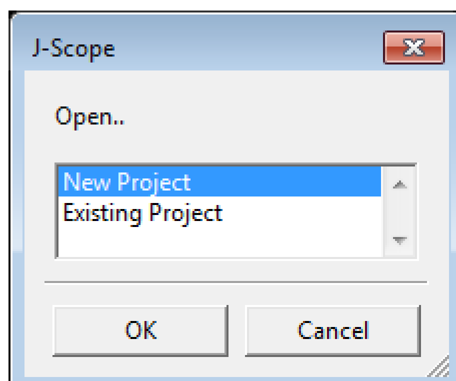
```
/*! \brief Driver re-init and reset */
void PID_Start(void);
```

resets the values, restarting the pid

Segger J-Scope

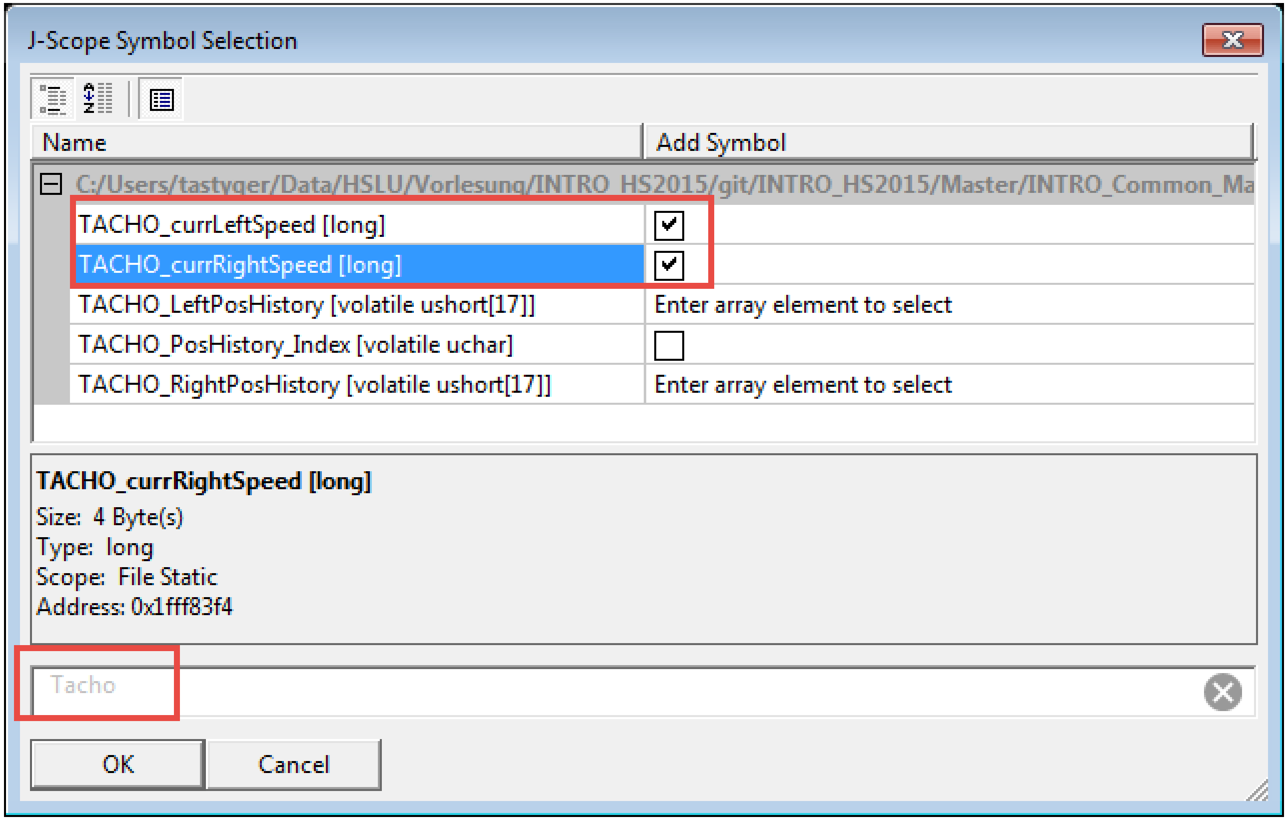
- Segger J-Link
- Windows only ☹
- https://www.segger.com/j-link-j-scope.html
- Needs J-Link software installed!
- <install>\segger\ Jscope.exe
- New or existing project
- Target: MK22FX512xxx12
- Sampling Source: **HSS**
- Specify Elf File





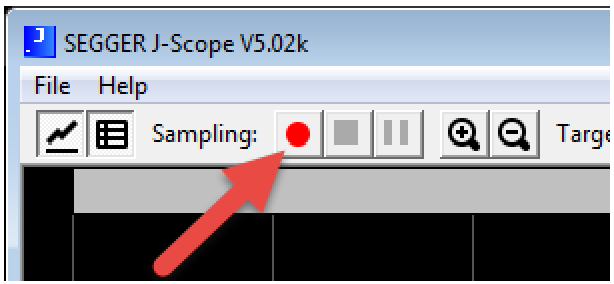
Segger J-Scope Variable Selection

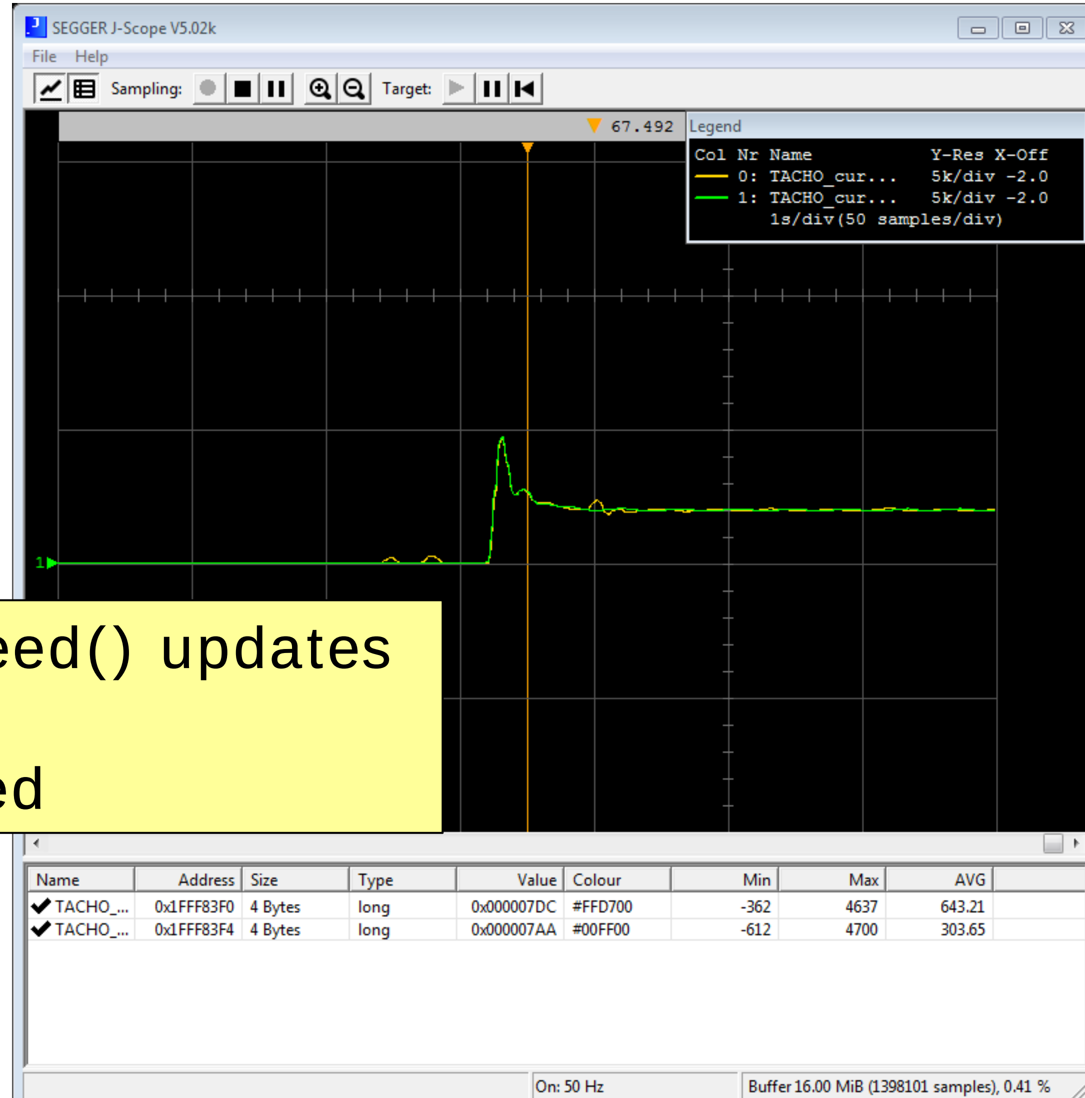
- Filter for variable name
- Enable variable to show in scope



Start Recording

- Use Start button

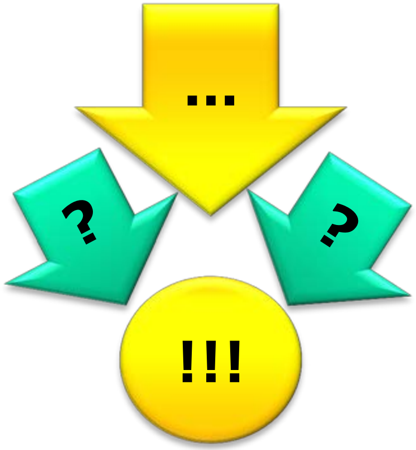




Note: TACHO_CalcSpeed() updates
TACHO_currLeftSpeed
TACHO_currRightSpeed

Summary

- P, I and D step response
- Impact of P, I and D terms
- Combinations
- PID controller
- Implementation in Software/Microcontroller
- Parameters and optimization
  - K_d, K_i, K_p
  - Anti-Wind-Up



Lucerne University of
Applied Sciences and Arts
HOCHSCHULE
LUZERN
Technik & Architektur

Lab: Closed Loop Control

- PID Closed Loop Controller (PID.c/.h)
  - Speed (moving at speed)
  - Turning (move to position)
  - K_p, K_i and K_d effects
- PID Factor turning
- Use Shell & J-Scope
- ➔ Next: Drive and Turn!