Ziegler-Nichols for PID Estimation

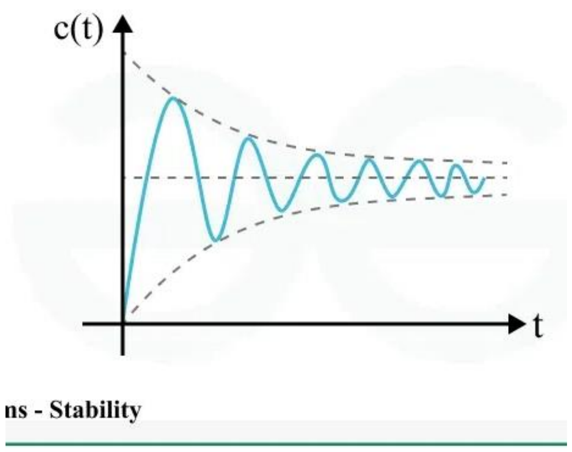
The Ziegler Nichols Method (1)

- Ziegler-Nichols is a **heuristic method** used to **tune** the parameters of a **PID** (Proportional-Integral-Derivative) controller, which is widely used in industrial control systems to maintain the desired output despite disturbances. The method proposes specific rules to set the values of the proportional gain (P), integral gain (I), and derivative gain (D) based on the response of the system to a specific test.

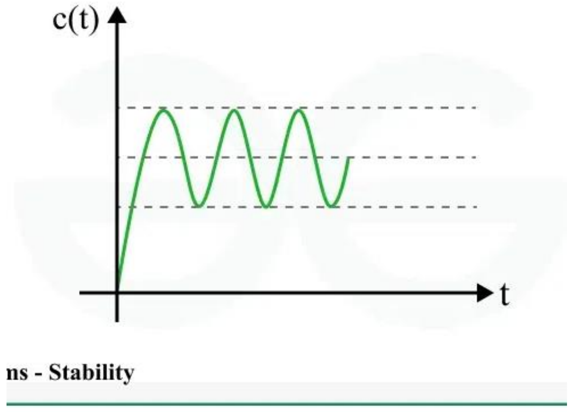
- For applying Ziegler-Nichols the **input** to the system needs to be a **step function** (regulation problem)

The Ziegler Nichols Method (2)

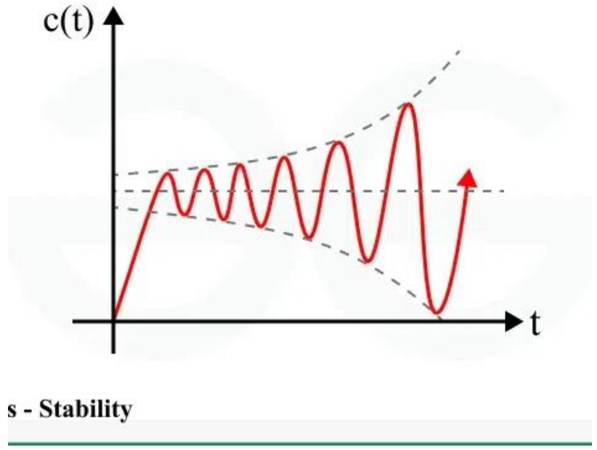
- The integral and derivative gains are set to zero, and the proportional gain is increased from a low value until the output of the system begins to **oscillate steadily**. This gain is called the **ultimate gain** (Ku), and the oscillation period at this gain is called the **ultimate period** (Tu).



Damped Oscillations



Sustained Oscillation



Unstable Oscillation

The Ziegler Nichols Method (3)

- Once Ku and Tu are known is possible to compute the Gains according to this table

Ziegler–Nichols method[1]

Control Type	K_p	T_i	T_d	K_i	K_d
P	$0.5K_u$	–	–	–	–
PI	$0.45K_u$	$0.8\overline{3}T_u$	–	$0.54K_u/T_u$	–
PD	$0.8K_u$	–	$0.125T_u$	–	$0.10K_uT_u$
classic PID[2]	$0.6K_u$	$0.5T_u$	$0.125T_u$	$1.2K_u/T_u$	$0.075K_uT_u$
Pessen Integral Rule[2]	$0.7K_u$	$0.4T_u$	$0.15T_u$	$1.75K_u/T_u$	$0.105K_uT_u$
some overshoot[2]	$0.3\overline{3}K_u$	$0.50T_u$	$0.3\overline{3}T_u$	$0.6\overline{6}K_u/T_u$	$0.1\overline{1}K_uT_u$
no overshoot[2]	$0.20K_u$	$0.50T_u$	$0.3\overline{3}T_u$	$0.40K_u/T_u$	$0.06\overline{6}K_uT_u$

Where Ki and Kd can be computed as:

$$K_i = K_p/T_i \text{ and } K_d = K_pT_d$$

Configuration File Setup

Set Damping Values

```
    "motor_direction": [[]],
    "motor_damping": [1],
    "motor_damping_coeff": [0.5,0.6,0.2,0.1,0.3,0.35,0.8],
    "motor_elastic_torque": [0],
    "motor_elastic_coeff": [0],
    "motor_inertia": [0],
    "motor_inertia_coeff": [0],
    "foot_friction": [2],
    "foot_restitution": [0],
    "enable_feet_joint_force_sensors": [["RL_foot_fixed", "RR_foot_fixed", "FL_foot_fixed", "FR_foot_fixed"]],
    "floating_base_name": ["floating_base"],
    "servo_pos_gains": [400],
    "servo_vel_gains": [3],
    "delay_measure_flag":[1],
    "delay_measure_steps":[30]
},
```

Set Delay Values

Laboratory Objectives (1)

- For this lab you need to estimate the right **Kp and Kd** parameters for **each joints**.
- The PD controller that we are tuning is a part of of the feedback linearization controller that it is already implemented in the code and it is called:

```
# Control command        VModugno, 4 weeks ago • added example with feedback linearizatio
cmd.tau_cmd = feedback_lin_ctrl(dyn_model, q_mes, qd_mes, q_des, qd_des_clip, kp, kd)
```

Laboratory Objectives (1)

- Starting from the first joint set the Kd=[0,0,0,0,0,0,0] and Kp=[0,1000,1000,1000,1000,1000,1000] and command a desired position only for the first joint and keep the desired velocity equal zero for all the joints.
- For streamline the testing part we provide to you a function to test one testing Kp for the current joint

```
#episode_duration is specified in seconds
def simulate_with_given_pid_values(sim_, kp, joints_id, regulation_displacement=0.1, episode_duration=10, plot=False):
```

Laboratory Objectives (2)

- Using this function plot the resulting motion for the desired joint and by **visual inspection** or **automatically** identify which Kp produces a sustained oscillation of the current joint
- Then once you have found the right Kp you can compute Ku and Tu

```
Ku = cur_kp
Tu = 1 / dominant_frequency
```

- The **dominant frequency** is the frequency at which the system's output oscillates most prominently when it is subjected to an input. This frequency can be identified in the **frequency spectrum** of the output signal as **the peak with the highest amplitude**

Laboratory Objectives (3)

- For finding the Dominant frequency you can use:

```
    You, 3 weeks ago • ziegler nichols a
def perform_frequency_analysis(data, dt):
    n = len(data)
```

- This function employs a Fast Furier Transform to find the frequency spectrum of a time signal

Laboratory Objectives (4)

Tasks:

1)Find the Ku and Tu for each joints

2)Design a strategy to final motion of the joint under inspection to check if reached a sustained oscillation

3)Try to design an automatic routine to establish if the joint has reached sustained oscillation

Laboratory Objectives (5)

4) Use the table to compute the Kp and Kd

5) why this method can can be dangerous?

6) for which system

You would not use this

Tuning method?

Ziegler–Nichols method[1]

Control Type	K_p	T_i	T_d	K_i	K_d
P	$0.5K_u$	–	–	–	–
PI	$0.45K_u$	$0.8\overline{3}T_u$	–	$0.54K_u/T_u$	–
PD	$0.8K_u$	–	$0.125T_u$	–	$0.10K_uT_u$
classic PID[2]	$0.6K_u$	$0.5T_u$	$0.125T_u$	$1.2K_u/T_u$	$0.075K_uT_u$
Pessen Integral Rule[2]	$0.7K_u$	$0.4T_u$	$0.15T_u$	$1.75K_u/T_u$	$0.105K_uT_u$
some overshoot[2]	$0.3\overline{3}K_u$	$0.50T_u$	$0.3\overline{3}T_u$	$0.6\overline{6}K_u/T_u$	$0.1\overline{1}K_uT_u$
no overshoot[2]	$0.20K_u$	$0.50T_u$	$0.3\overline{3}T_u$	$0.40K_u/T_u$	$0.06\overline{6}K_uT_u$

You can find the code in this repo: https://github.com/VModugno/lab_sessions_COMP0211_PUBLIC