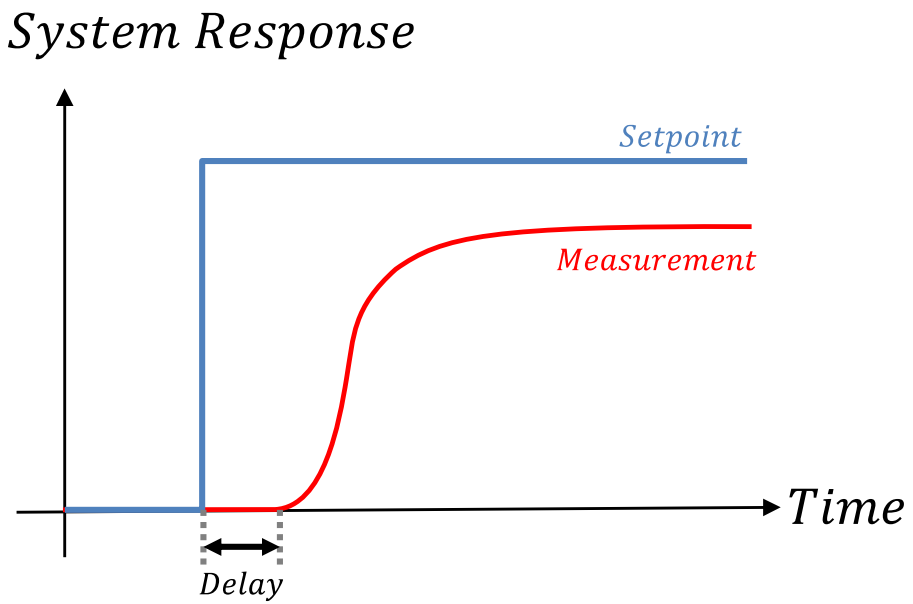
System Dynamics – Lags/delays

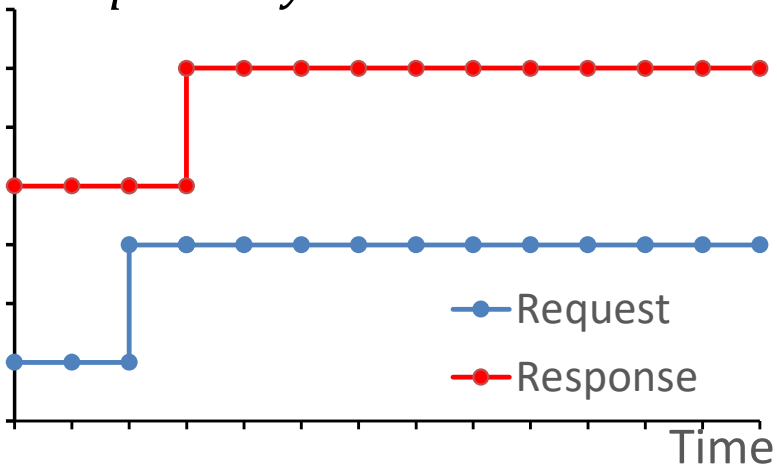
Many systems do not respond immediately to control input. It's common that they take some time before starting to change.

- E.G. When a turned off motor is supplied with a certain amount of energy, it will take certain time to beat its own inertia in order to start moving.

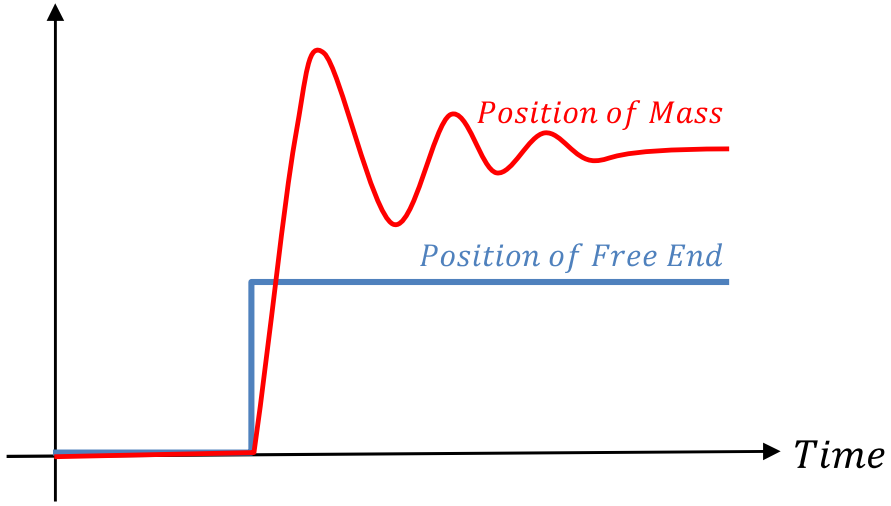


Exercise: there is a lag/delay in the following systems?

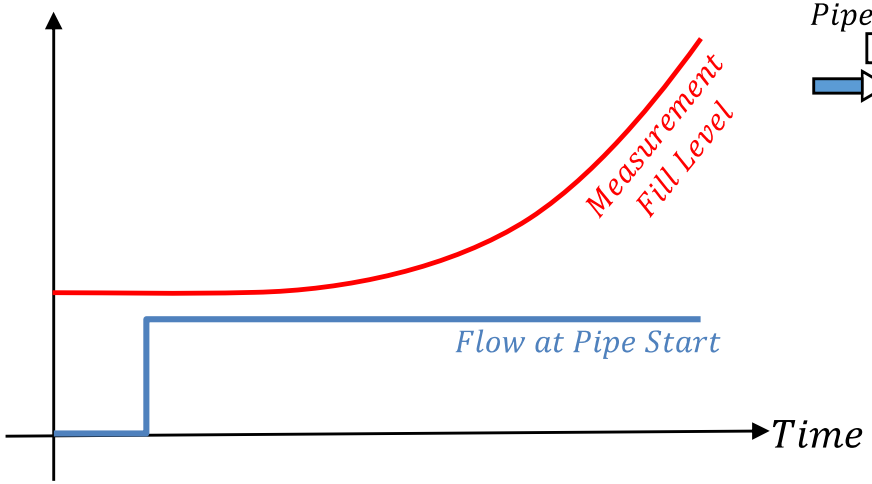
Dynamic response of a computer system

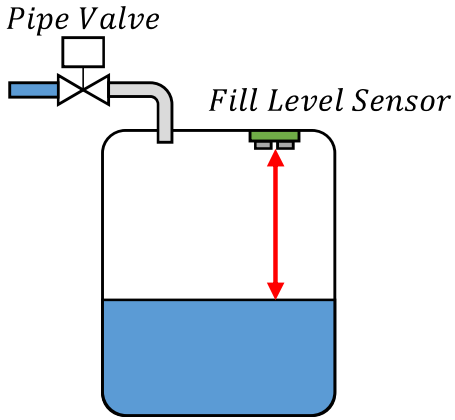


Mass – Spring – Damper response



Pipe Feeding into Tank response





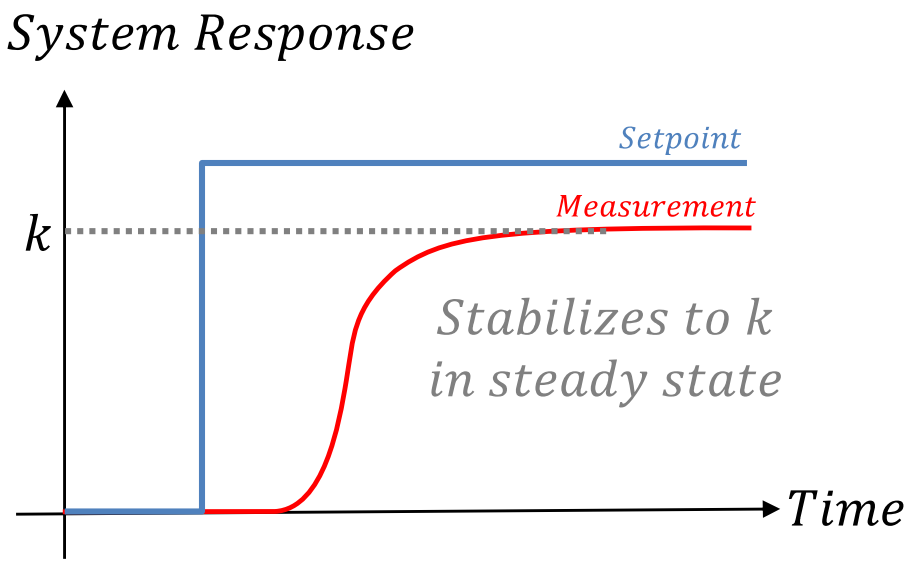
System Dynamics – Stability

- The stability is determined by the response of the system to inputs or disturbances. A system which remains in a constant state unless affected by an external action and which returns to a constant state when the external action is removed can be considered to be stable. Unstable systems don't "stabilize" to a constant value, instead they keep increasing.
- To calculate if a system is stable, it's required to find at what value the system stabilizes in steady state (Y_{ss}) after applying a unit-step input using the following formula:

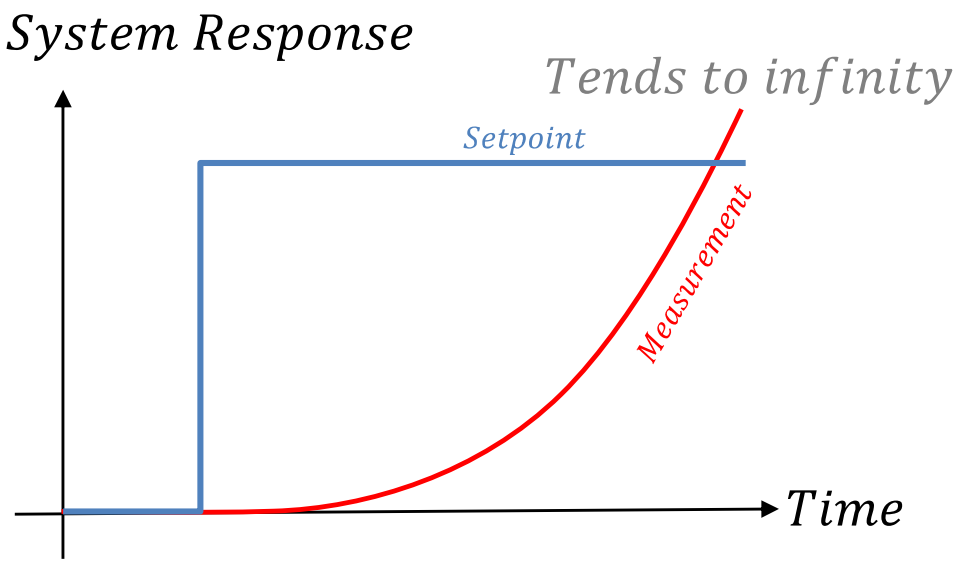
$$Y_{SS} = \lim_{t \to \infty} OUTPUT(t)$$

  - If this limit tends to a constant, the system is stable. If it tends to infinity, the system is unstable.

Stable System

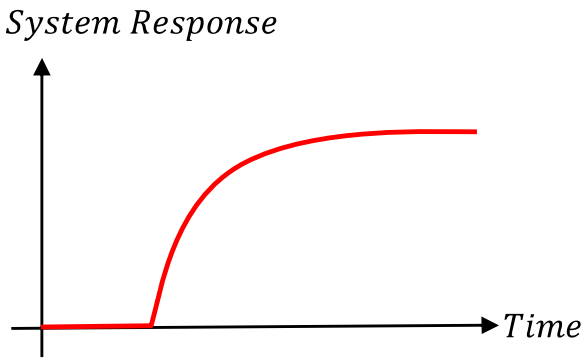


Unstable System

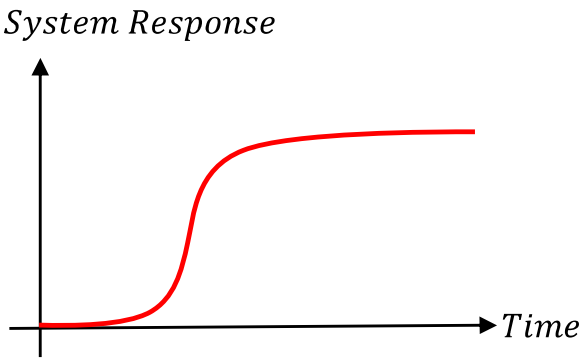


System Dynamics – Order

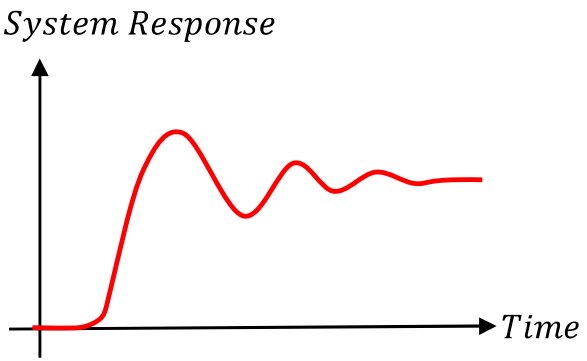
A system can have different type of behaviors. This is determined by the differential equations that model the system and affects how it responds to an input during the transient state. Depending on the order of this differential equations, the system may respond to an external step input as follows:



First Order



Second Order (Overdamped – Critically Damped)



Second Order (Underdamped)

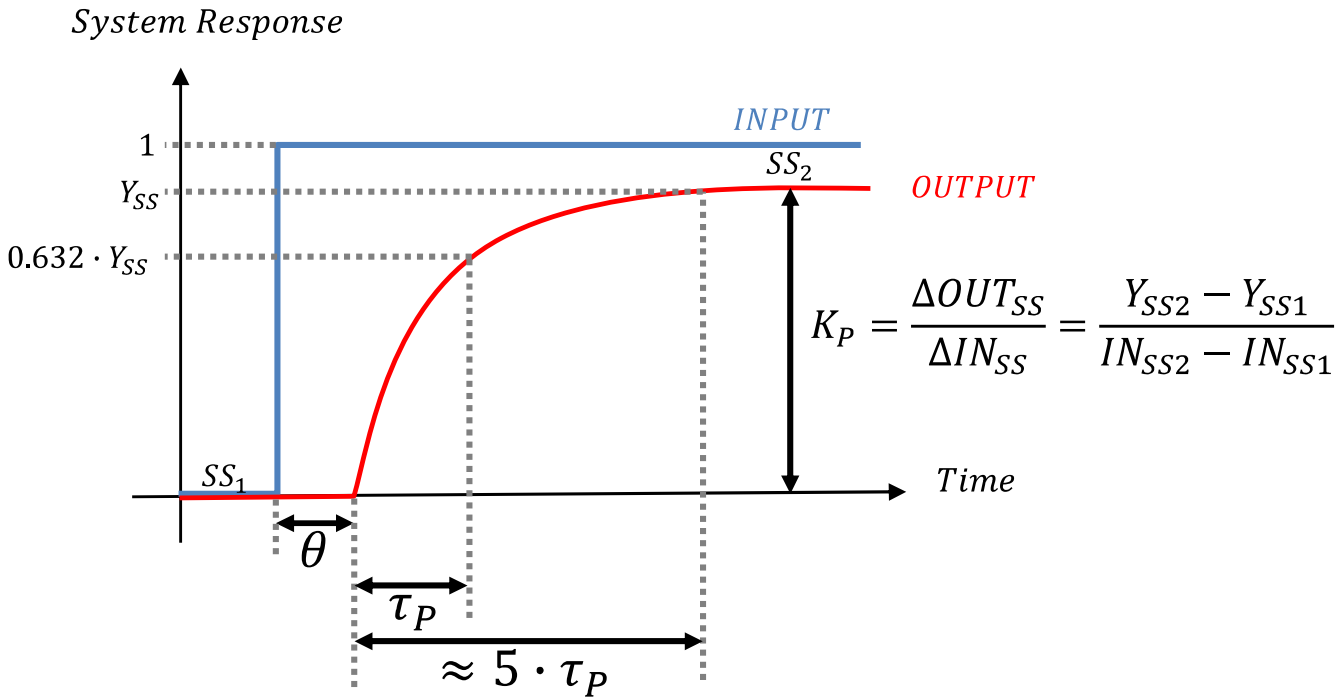
System Dynamics – First Order Plus Dead Time Model (FOPDT)

It's an empirical description of many stable dynamic systems using a first order transfer function:

$$FOPTD = \frac{OUTPUT(s)}{INPUT(s)} = \frac{K_P \cdot e^{-\theta s}}{\tau_P \cdot s + 1}$$

Where:

- K_P: Process gain, represented as the change in the $OUTPUT$ induced by a unit change in the $INPUT$. It can be calculated measuring the steady state output value.
- τ_P: Process time constant. Is the amount of time needed for the $OUTPUT$ to reach the 63.2% of the way to the steady state vale (Y_{SS}). This value affects the speed of the system response, usually the system stabilizes at $5 \cdot \tau_P$.
- θ: Process delay. Time required for the system to start variations after an applied external input



System Dynamics – Second Order Plus Dead Time Model (FOPDT)

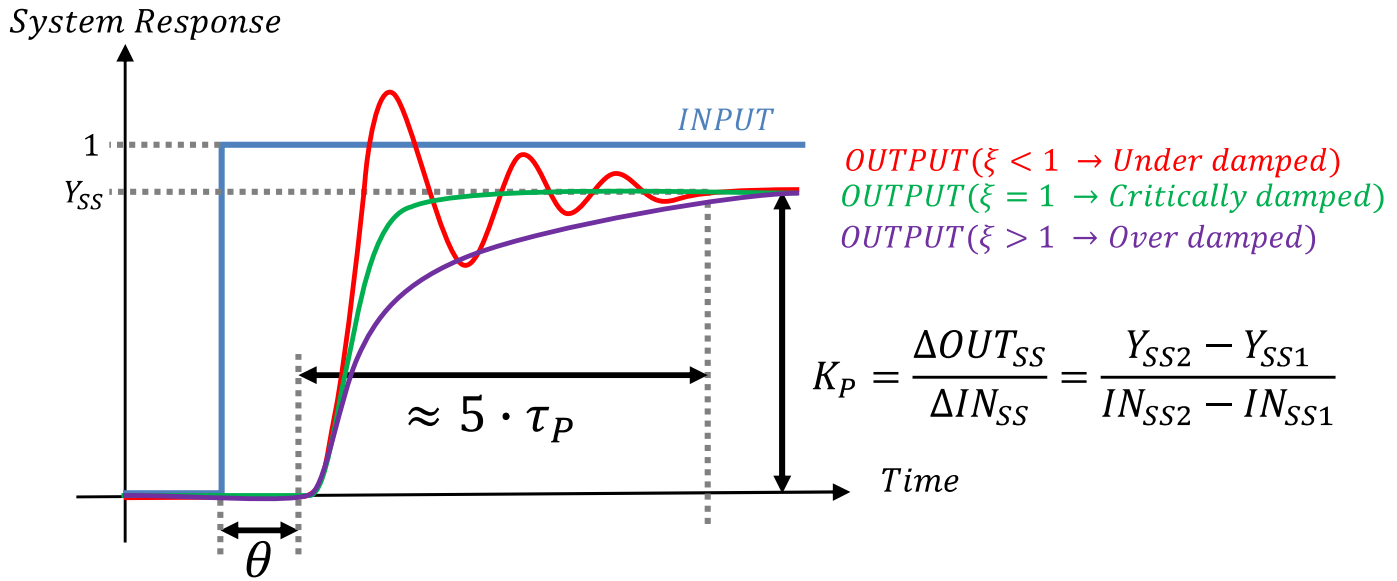
It's an empirical description of stable dynamic systems using a second order transfer function:

$$SOPTD = \frac{OUTPUT(s)}{INPUT(s)} = \frac{K_P \cdot e^{-\theta s}}{\tau_P^2 \cdot s^2 + 2 \cdot \xi \cdot \tau_P \cdot s + 1}$$

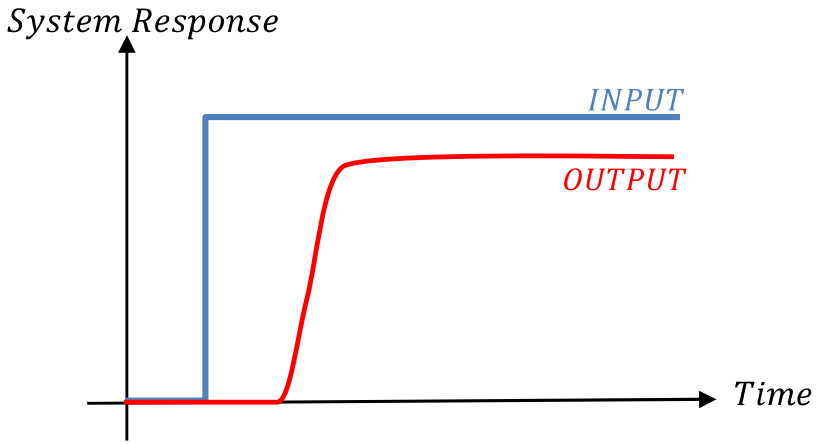
Where:

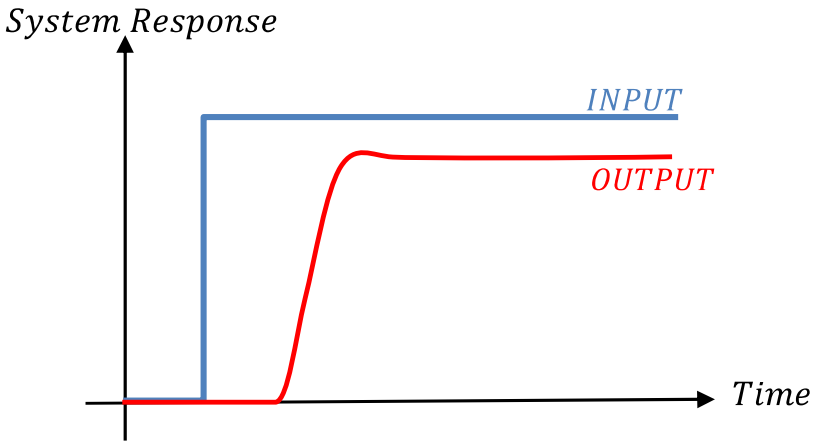
- K_P: Process gain, represented as the change in the $OUTPUT$ induced by a unit change in the $INPUT$. It can be calculated measuring the steady state output value.
- ξ: Damping factor.
  - If this value is greater than one (> 1), the system will be over damped.
  - If it's equal to one ($= 1$), the system will be critically damped.
  - If it's less than one (< 1), the system will be under damped.
- τ_P: Process time constant. Usually the system stabilizes at $5 \cdot \tau_P$.
- θ: Process delay. The amount of time required for the system to start variations after an applied external input.

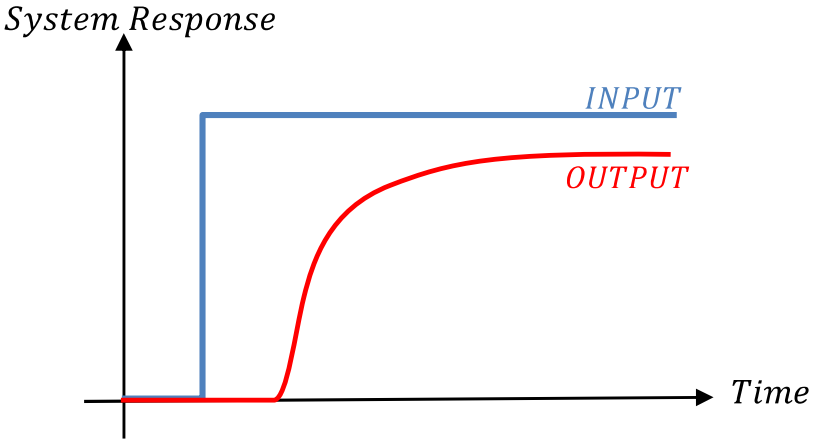
Note: A second order system can be approached to a first order system but the model will lose accuracy.

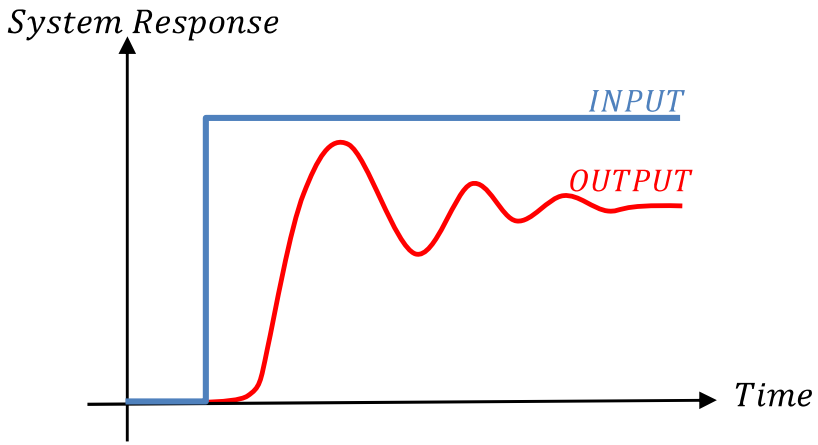


Exercise: identify the order of the following systems









But, how to identify the mathematical model that represents the behavior of a real system?

- **The process reaction curve** (Marlin, 2000): Consists in manually finding the First Order Plus Dead Time (FOPDT) parameters from a system response plot. This method is simple to perform but can be inaccurate because it depends in the user eye and is only restricted to FOPDT model.
- **Statistical lineal regression methods** (Marlin, 2000): Consist in finding a process model through parameter estimation using linear regression methods and real time process data with an injected input. This method requires to have the dataset of the real system with a sample time enough to capture correctly the transitions of the system and the delay after input. It uses the linear least squares algorithm and it can only be used to fit FOPDT model.
- **Statistical non linear regression methods** (Pintelon et al, 2012; Box and Jenkins, 1976)**:** Generally used for second or higher order system dynamics response. They require nonlinear optimization methods and the dataset from an injected setpoint to the system. This method can provide greater accuracy to fit a system dynamics dataset response into a model but it requires higher computation.

E. Marlin, T. (2000). Process Control: Designing Processes and Control Systems for Dynamic Performance. New York: McGraw-Hill (Second, pp. 176–206). New York, NY, USA: McGraw-Hill, Inc. Retrieved from http://pc-textbook.mcmaster.ca/

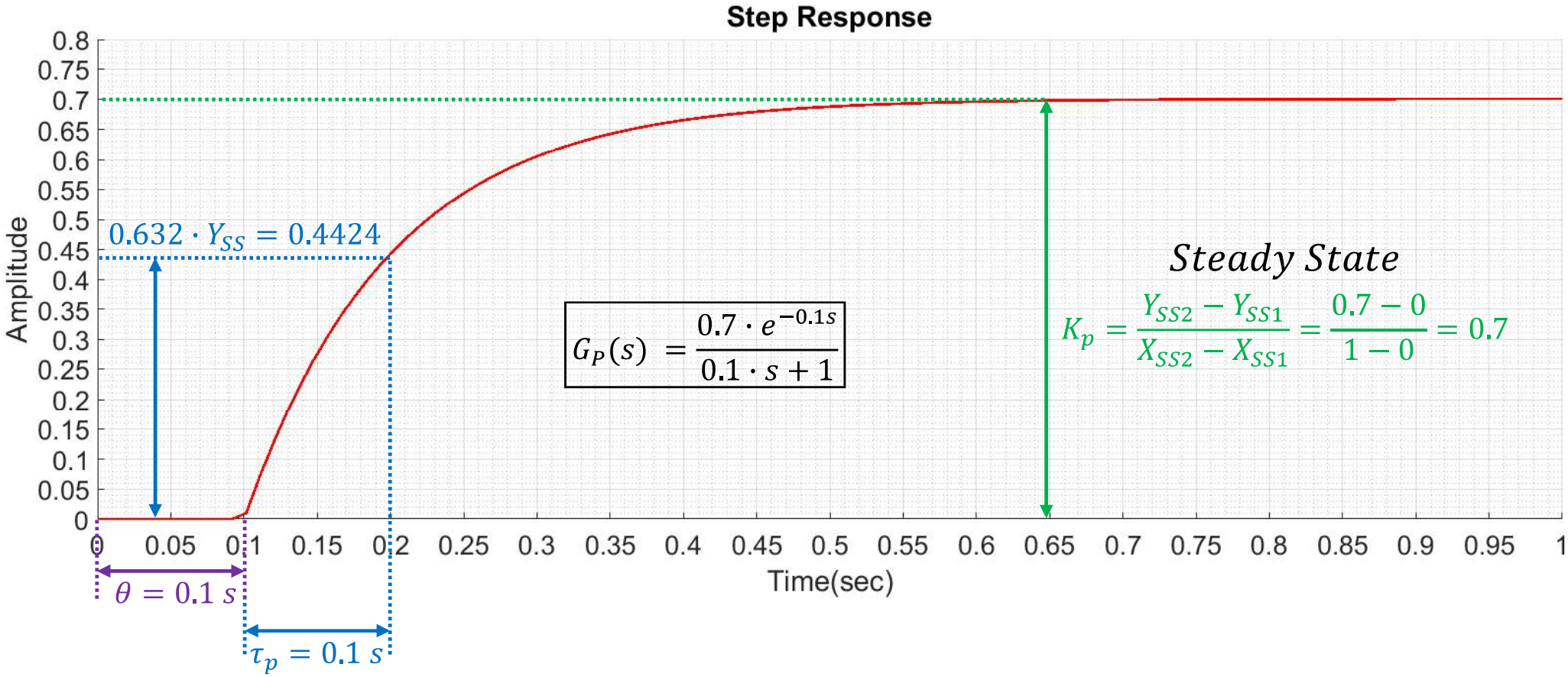
Pintelon, R., & Schoukens, J. (2012). System Identification: A Frequency Domain Approach, Second Edition. *System Identification: A Frequency Domain Approach, Second Edition*, *39*(6), 3276–3287. https://doi.org/10.1002/9781118287422

Box, G. E. P., Jenkins, G. M., & Reinsel, G. C. (1994). Time Series Analysis: Forecasting & Control. Book. https://doi.org/10.1016/j.ijforecast.2004.02.001

Identifying the mathematical model of the behavior of a system through geometric first order plus dead time approximation

This method consists in identifying the parameters from a system response plot, finding all the parameters for a First Order Plus Dead Time model. The steps to use this method are:

1. Allow the process to reach steady state.
2. Introduce a single step change in the input variable.
3. Collect input and output response data until the process again reaches steady state
4. Perform the graphical process reaction curve calculations:
   - I. Identify the steady step in the plot and calculate the process gain K_p, finding the Y axis values (Y_{SS}) for the Output signal and the Setpoint (X_{SS}) at the beginning and when the input signal was injected.
   - II. Measure the delay θ, which is the distance from the beginning of the Setpoint injection to when the Output signal starts to increase.
   - III. Calculate the 63.2% of the Output in steady state and find its corresponding value in time axis. Subtract the delay from this time value and this will be the process time constant τ_p.
   - IV. Replace the calculated parameters into the First Order Plus Dead Time system model. $FOPTD = \frac{OUTPUT(s)}{INPUT(s)} = \frac{K_P \cdot e^{-\theta s}}{\tau_P \cdot s + 1}$

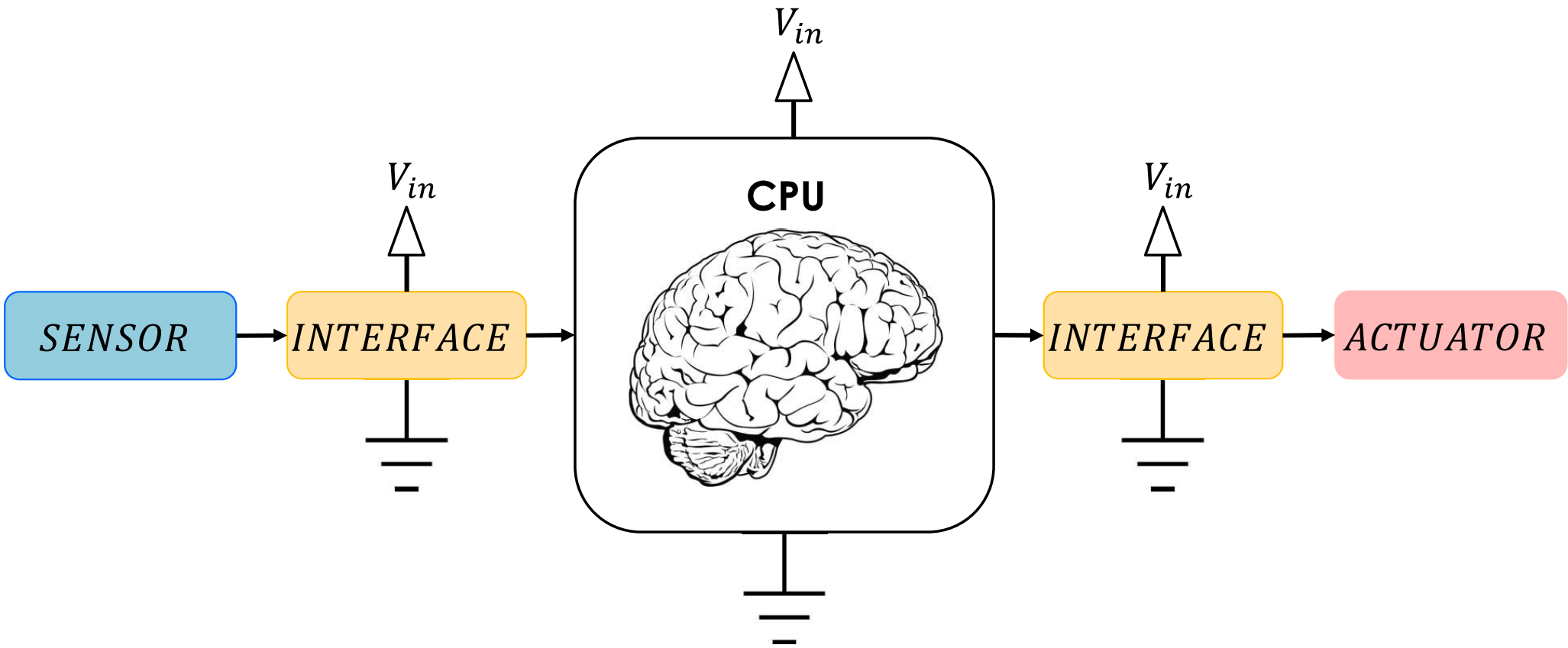


Identifying the mathematical model of the behavior of a system through the statistical regression method (using Matlab)

- Process for acquiring a mathematical model from a real system response.
- This model is usually a transfer function in Laplace Domain.
- Commonly used as the first step to control a system.
- Consists in 3 steps:

Process Implementation → Data Acquisition → Model Fitting

1. Connect all the actuator, sensors and interfaces required to the selected *Data acquisition* hardware.



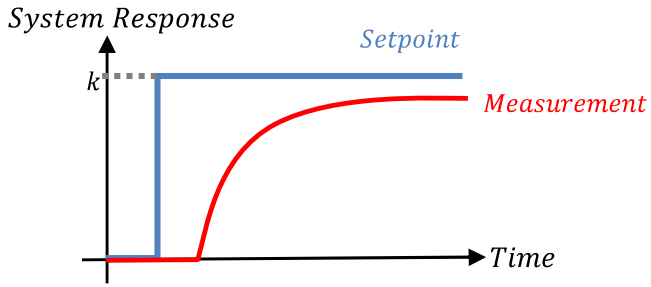
2. Implement a software that is able to inject a Setpoint at a desired time and starts reports the Setpoint, Measurement and Time in a file (e.g. Arduino Datalogger).

```
void datalog() {
  if (millis() - tinis >= TSAM) {
    stime = stime + TSAM / 1000.0;
    //SD Logging
    String datalog = ""; //Define datalog as string for storing data in SD as text
    datalog += String(Setpoint, 4);
    datalog += ",";
    datalog += String(Measurement, 4);
    datalog += ",";
    datalog += String(stime, 4);
    File dataFile = SD.open("datalog.txt", FILE_WRITE);
    // if the file is available, write to it:
    if (dataFile) {
      dataFile.println(datalog);
      dataFile.close();
      // print to the serial port too:
      //Serial.println("Success logging data");
    }
    // if the file isn't open, pop up an error:
    else {
      //Serial.println("error opening datalog.txt");
    }
    tinis = millis(); //Reset tini
  }
}
```

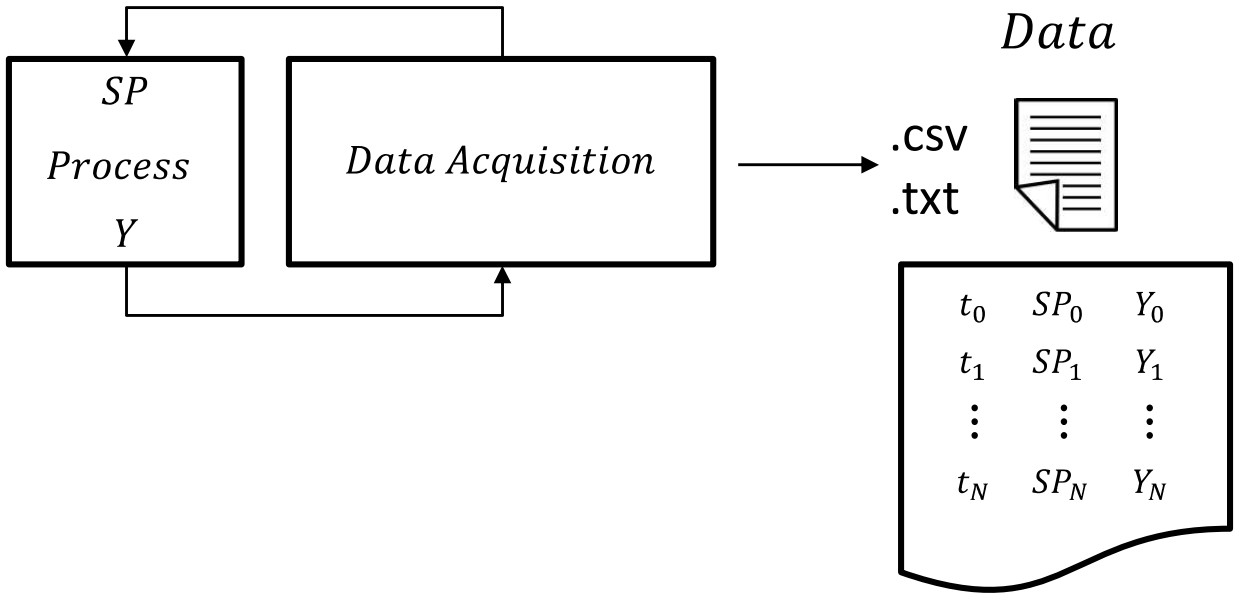
```
void loop() {
  Measurement = smooth(SENSOR, total, readings, readIndex, NUMREADS); //Acquire measurement and smooth it through RMA
  if (debounce(SW,ledState,buttonState,lastButtonState,tinid)==true) {
    Setpoint = SPF; //Assign the desired setpoint
  }
  else {
    Setpoint = 0; //Clear the desired setpoint
  }
  datalog();  //Datalog variables
  printvars();  //Print vars for debug
  analogWrite(ACTUATOR, Setpoint*255.0/1023.0);  //Inject the setpoint to the output
}
```

https://github.com/tidusdavid/arduino-datalogger

4. Inject a Set-point (SP) to the system by running the data acquisition software connected to hardware system.



5. Report since the setpoint injection a datalog until the system reaches stability or steady state (only applies for stable systems) in a .txt file (storing Setpoint, Measurement and Time).



Process Implementation → Data Acquisition → **Model Fitting**

6. Use a system identification software to obtain a transfer function model with high correlation to the real system, using the previous dataset acquired. E.g., Control Station or Matlab.

- Note: It's required that the Setpoint and the Measurement start initially at "0" amplitude. If there is an offset in the Setpoint or the Measurement, remove it until both signals start at "0" amplitude. Also both signals must be positive (apply abs to them if there are negative values).

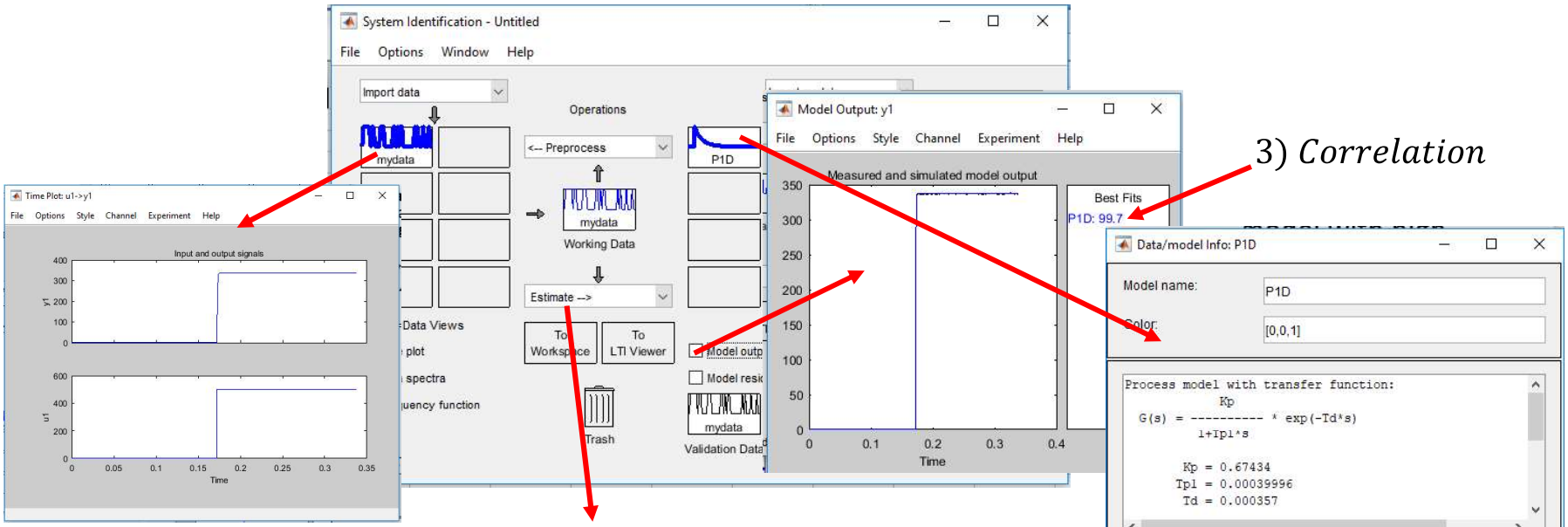
0) *Import Data*

System Identification Software

Matlab® System identification App

First Order with Dead Time Model

$$G_P(s) = \frac{Y(s)}{SP(s)} = \frac{K \cdot e^{-\theta \cdot s}}{\tau_P \cdot s + 1}$$



1) *Inputs and Outputs*

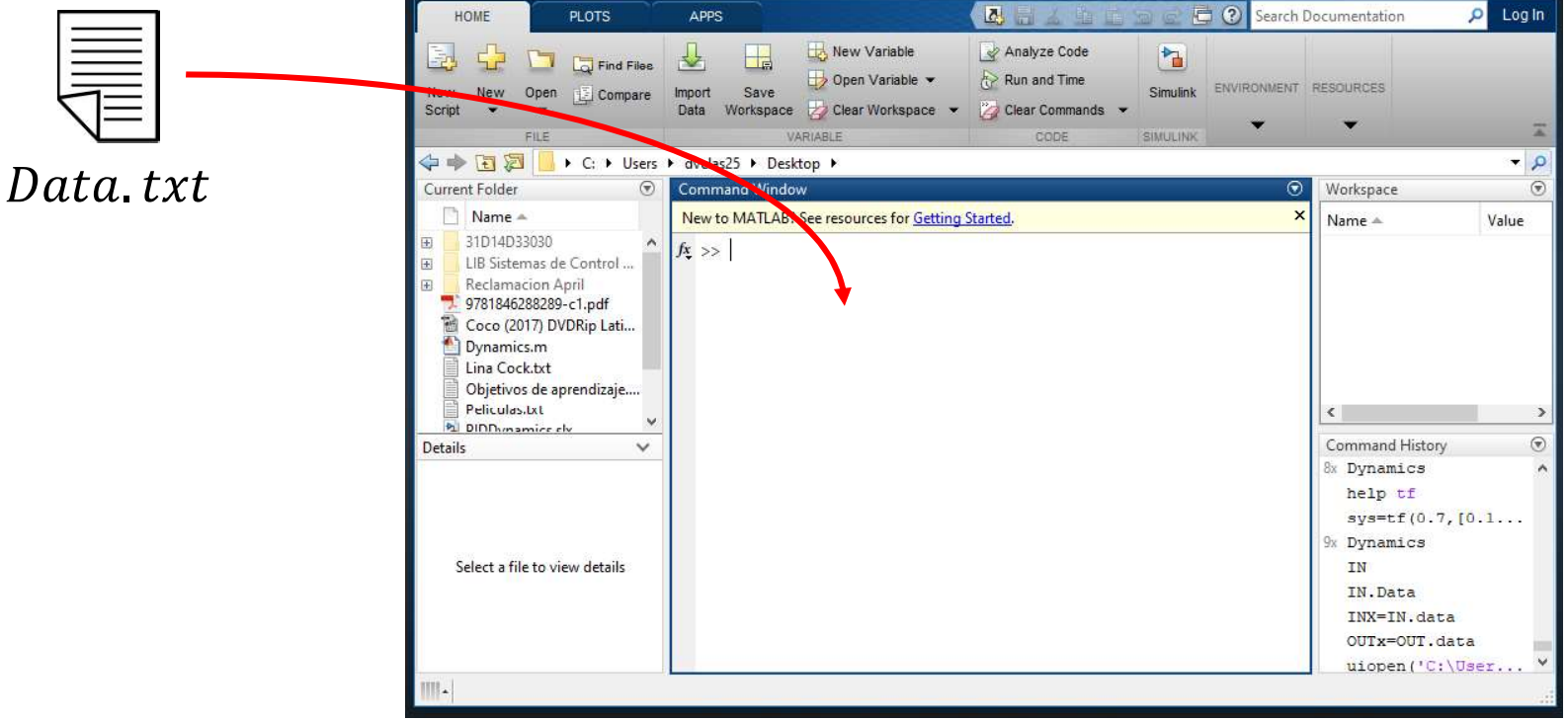
2) *Estimate "Process Models"*

3) *Correlation*

4) *Model Identified*

(For Matlab only)

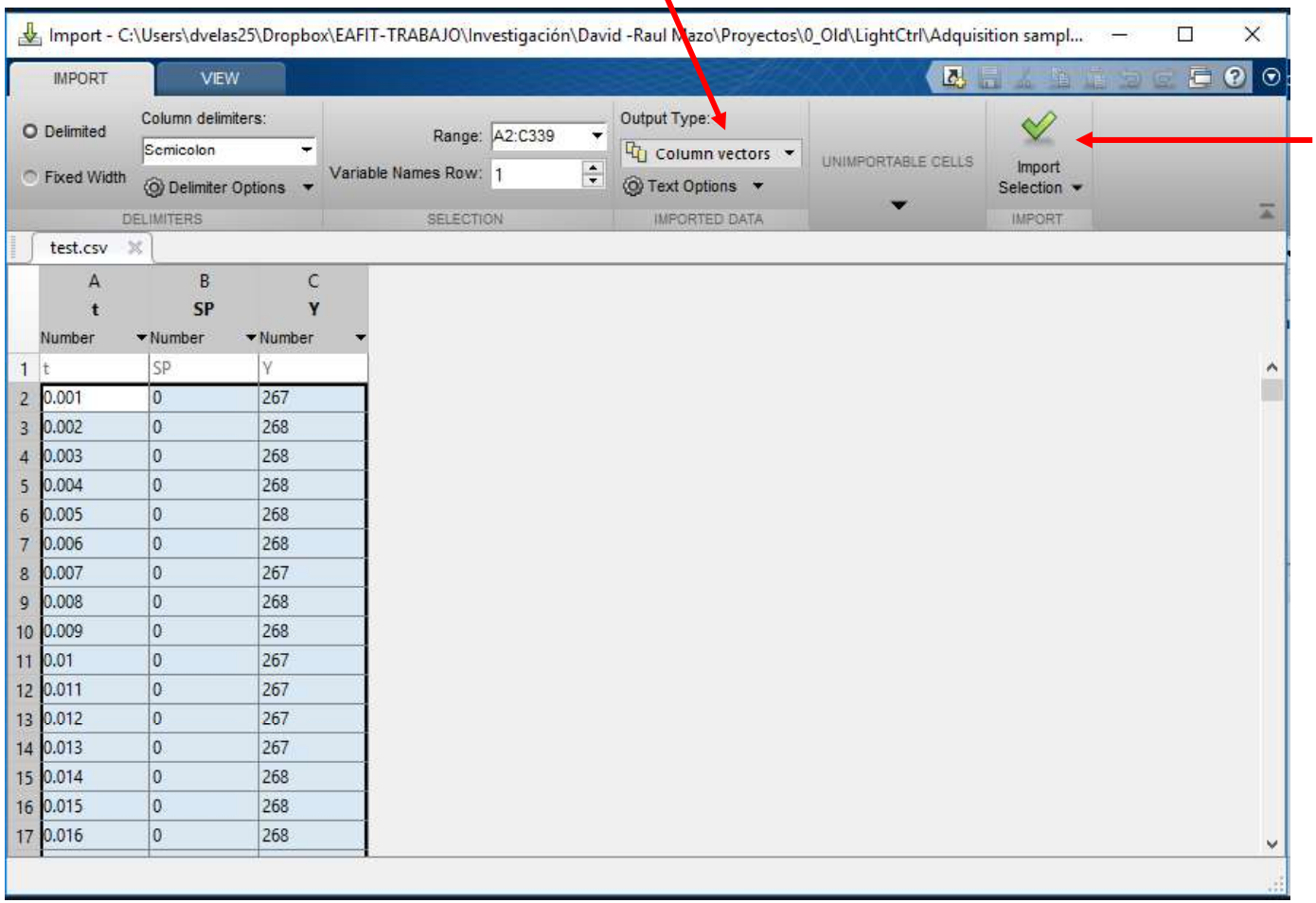
6. 0) Import stored dataset in .txt to Matlab Workspace drag and dropping the file into the Matlab Command Window.



(For Matlab only)

6. 0) Import the data as Column Vectors

I. Select Column Vectors

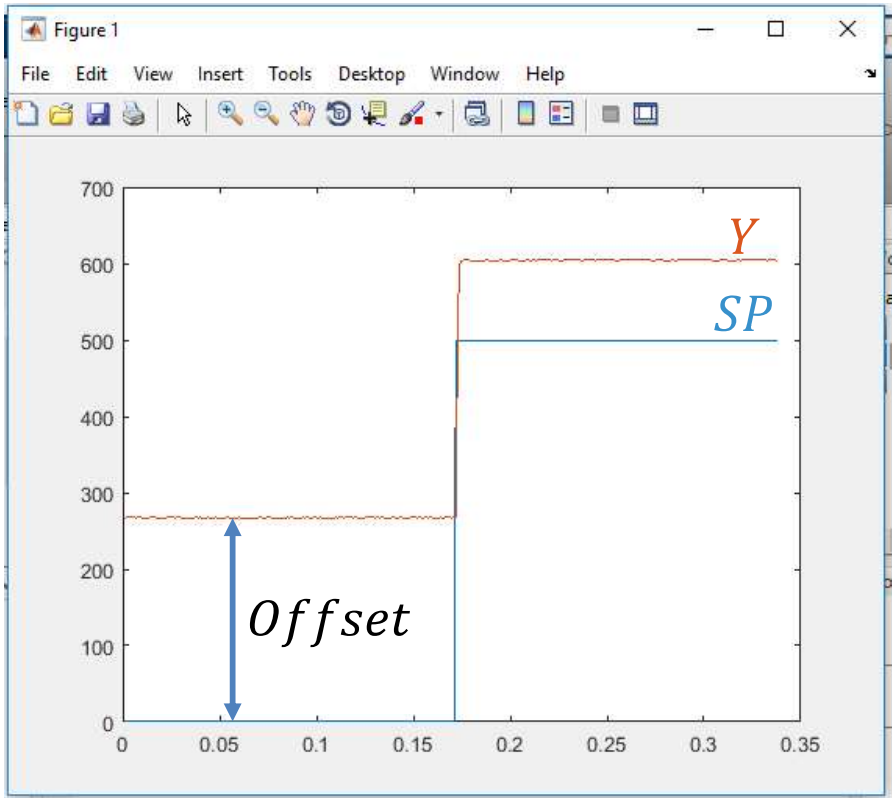


II. Click Import

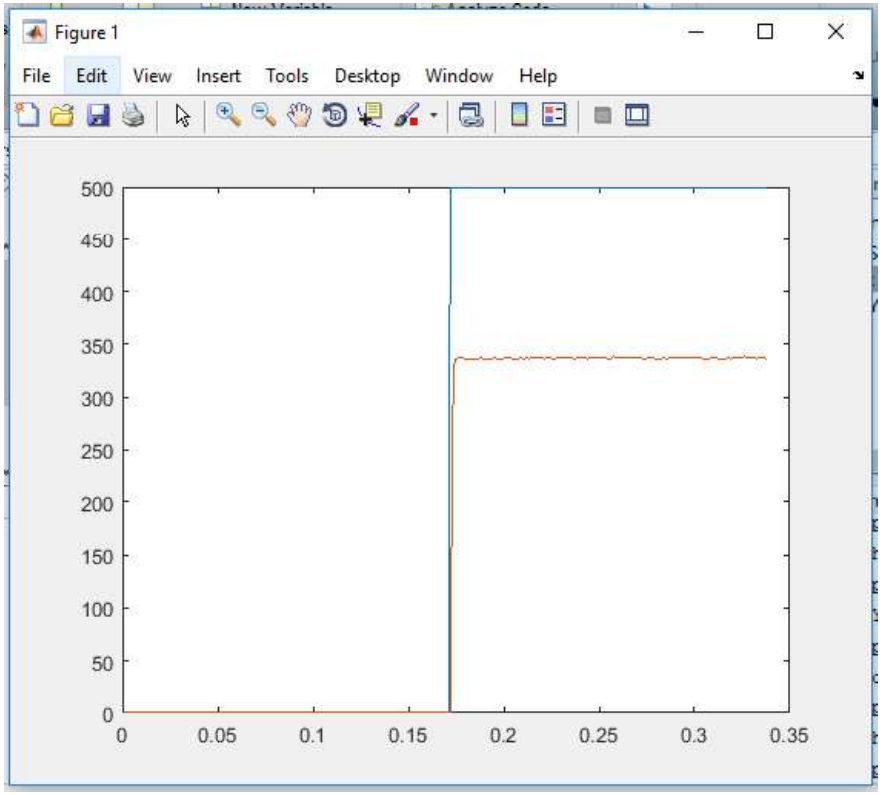
(For Matlab only)

6. 0) Plot Setpoint (SP) and Measurement (Y) versus time to see if data is correct (without offsets and noise) in Matlab.

- Use plot(t,SP) and plot(t,Y)
- Correct the offset removing the mean value of Y before the Setpoint was injected→ Y=Y-Offset;



Preprocess



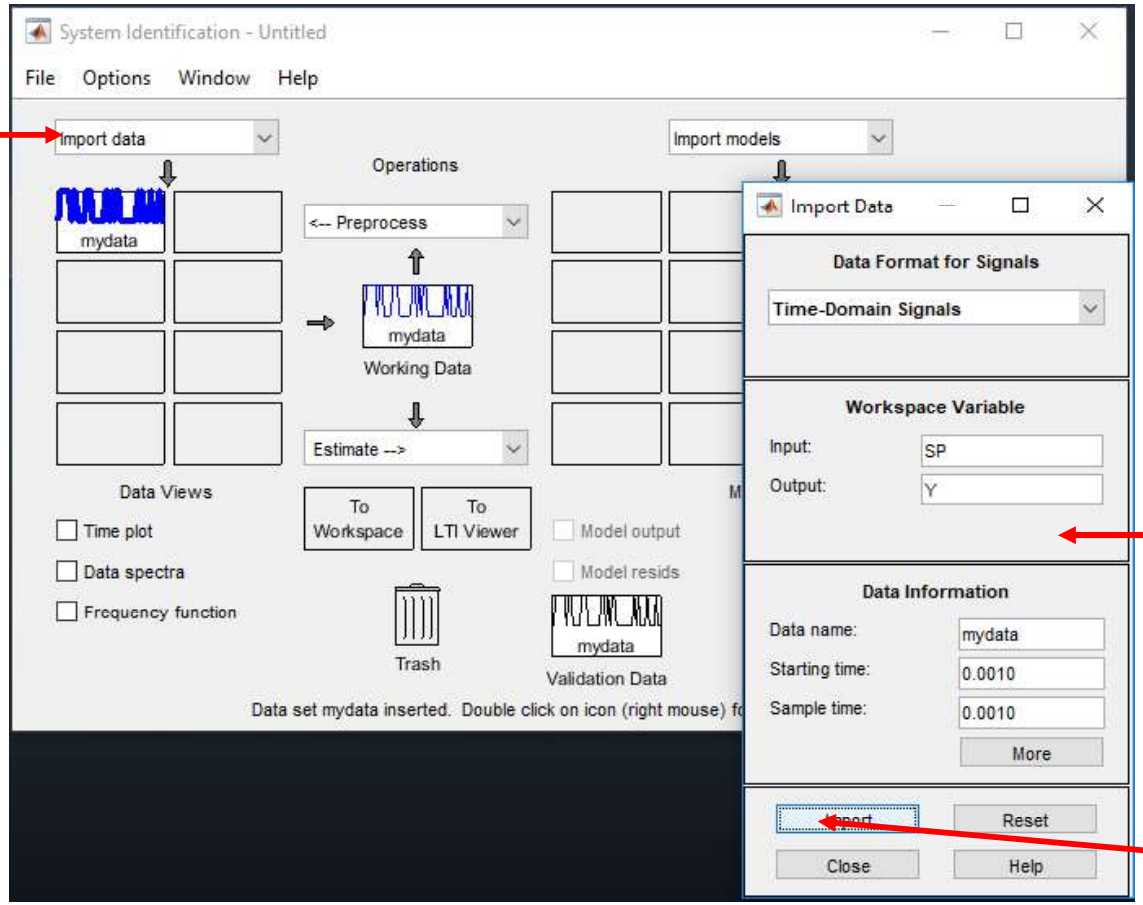
Without Offset

(For Matlab only)

6. 1) Open System Identification App and import SP and Measurements using Import data/Time domain data.

- Specify the sample time and starting time of the signals.

I) Click Time domain data



II) Fill Input Output Starting Time Sample Time

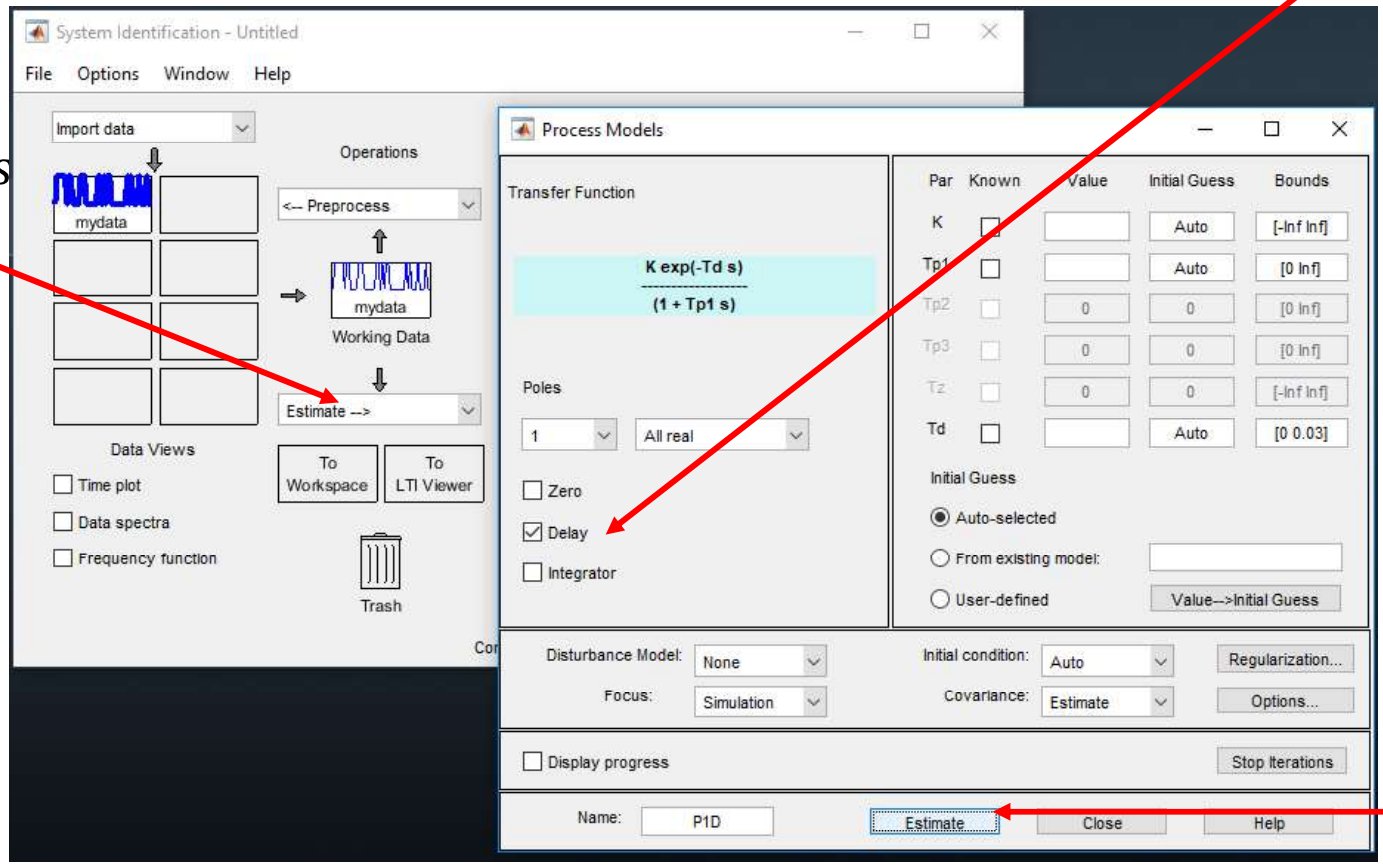
III) Click Import and close dialog

(For Matlab only)

6. 2) Estimate process model using a First Order Plus Dead Time (FOPDT) model.

I)Estimate: Process Models

II)Check delay



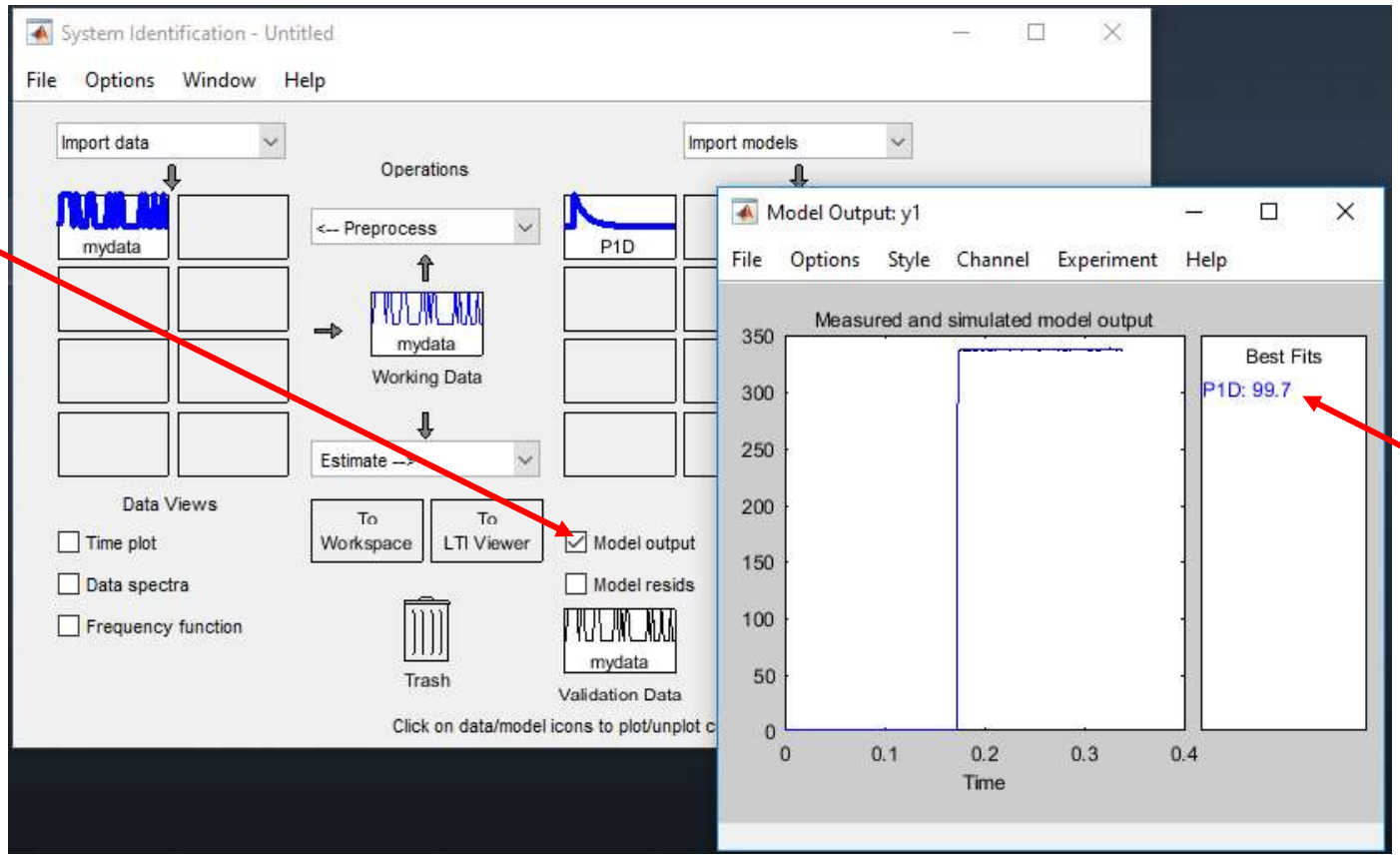
III)Click Estimate and close dialog

(For Matlab only)

6. 3) Check Model output correlation to see if it fits (recommended greater than 80%)

- If the correlation is low, try to use a second order plus dead time model (SOPDT) or filter better your data from noise (acquire new dataset if required).

$I) Click$: $Model\ Output$

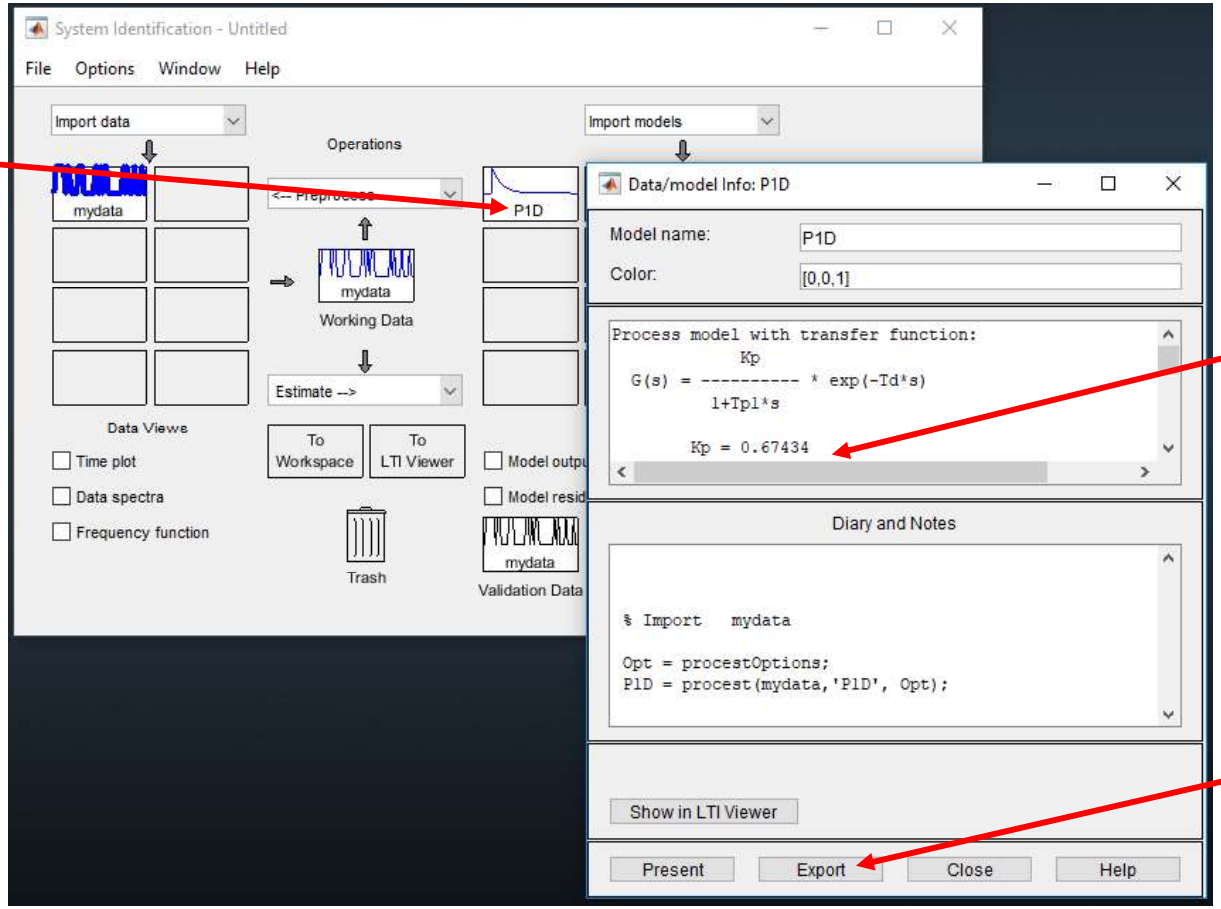


$II) Check\ correlation$

(For Matlab only)

6. 4) Observe the identified model and export it to workspace.

$I)Double\ Click$



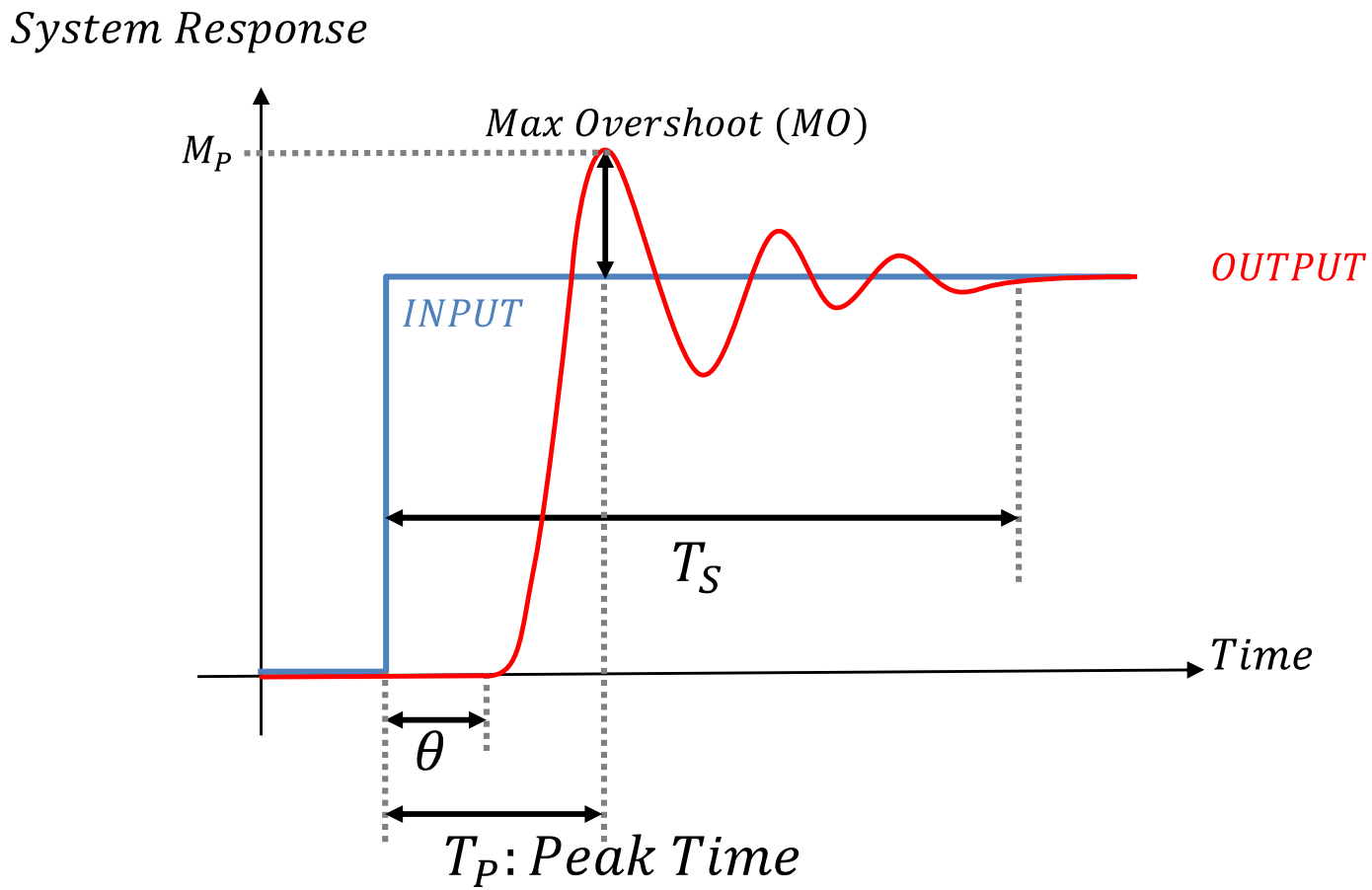
$II)Check\ model$

$$G_P(s) = \frac{K \cdot e^{-\theta \cdot s}}{\tau_P \cdot s + 1}$$

$III)Export\ to\ workspace$

Control

- The purpose of a controller is to calculate a signal (corrective or control action) that is suitable for the system in order to reach a zero error.
- The controller can be designed in order to modify transient and steady states behavior depending on the system response requirements.
- The most important requirements are:
  - Stability: The system must have a constant steady state.
  - Maximum Overshoot (MO): Defined as the maximum percentage of the output signal with respect to the steady state value during transient state.
  - Stablishing time (T_S): The time for the system to stabilize to INPUT desired value.

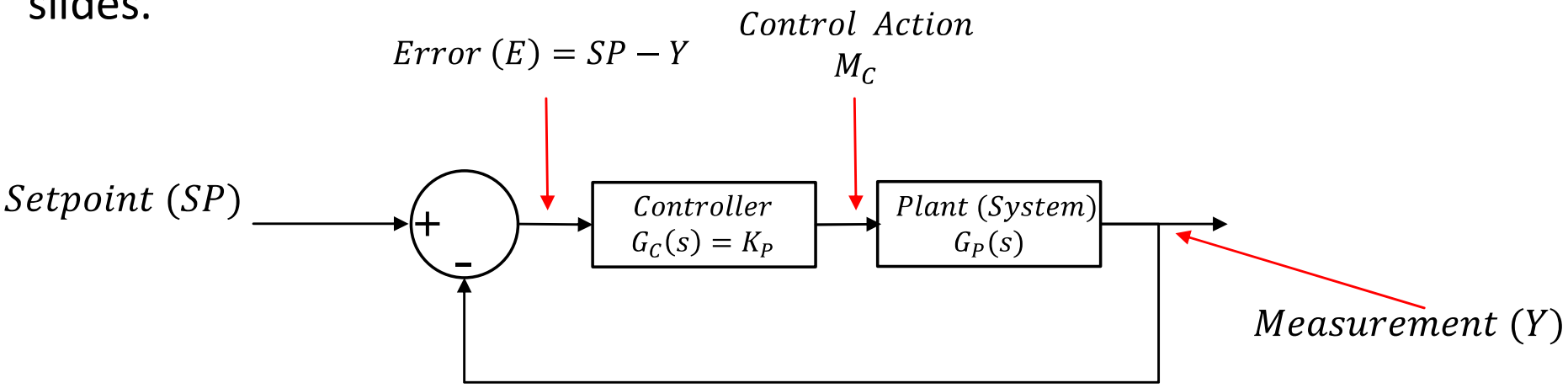


Control variations for different requirements

Controller Type	Application
Finite State Machine (FSM)	It's used to give a set of system states and transitions between them depending on inputs variation and the internal variables. This type of controller gives "autonomy" to systems through the change on system modes or states.
ON/OFF Controller	This controller is mostly used in applications that doesn't require accuracy or fast response. It's commonly used for controlling temperature systems. E.G. Electric stoves with low, medium or high temperatures selector.
PID Controller	The PID controller can provide control action designed for specific process requirements. The response of the controller can be described in terms of the responsiveness of the controller to an error, the degree to which the controller overshoots the setpoint, and the degree of system oscillation. It's commonly used for system that require accuracy, fast response to changes and stability.
Advanced self adaptive controllers	This type of controllers are used to systems that can vary due to context changes or critical fail-proof systems. This controllers can get a model of the system on run-time and adapt or compute new gains to adjust the overall controller.

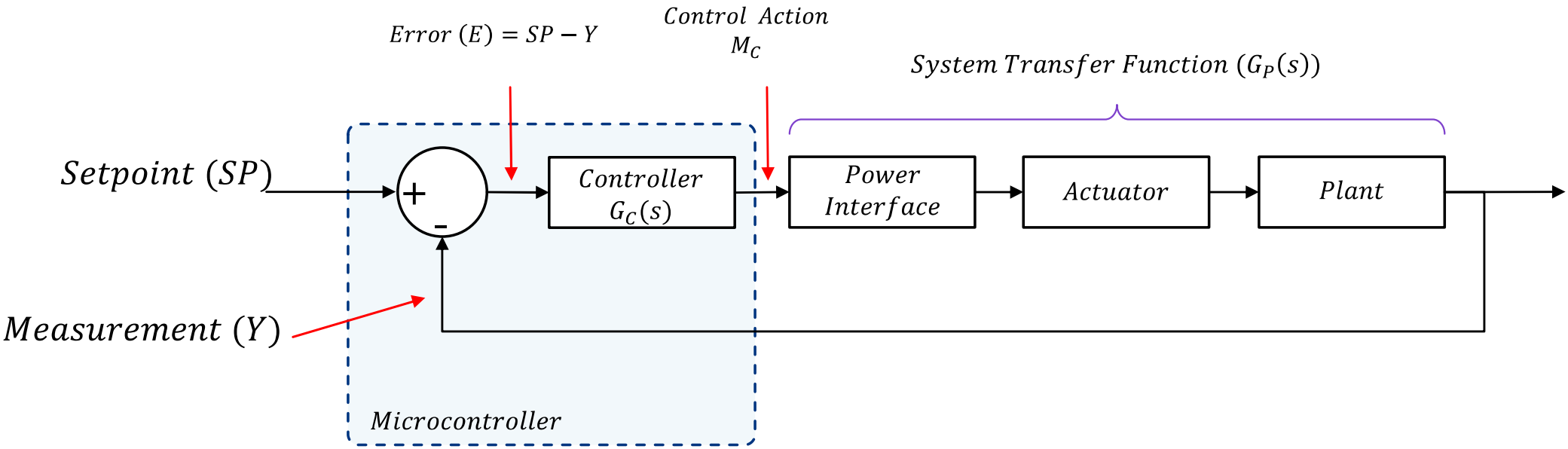
Control Systems Architecture

- A **block diagram** standard loop is a suitable and easy way of **representing all the elements** involving a **controller** and the **system**.
- It includes the **inputs**, **outputs** and **control signals** represented by three elements:
  - **Boxes:** Representation of system, such as controllers, filters, interfaces, actuators and sensors.
  - **Arrows:** Used for showing the flow and direction of control signals.
  - **Circles:** Indicate comparisons of sums or differentiation.
- The following image shows a simplified representation of a control systems architecture block diagram, which will be detailed in the next slides.

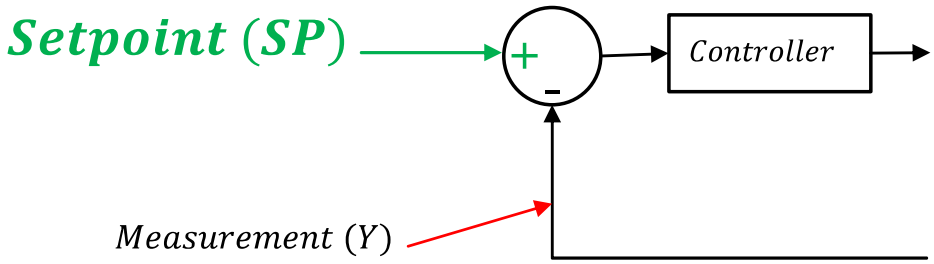


Control Systems Architecture Detailed

- The following block diagram shows an expanded and more detailed version of the control system architecture, including:
  - Actuators with their Power Interfaces.
  - Sensors with their respective interfaces required for conditioning its signal.

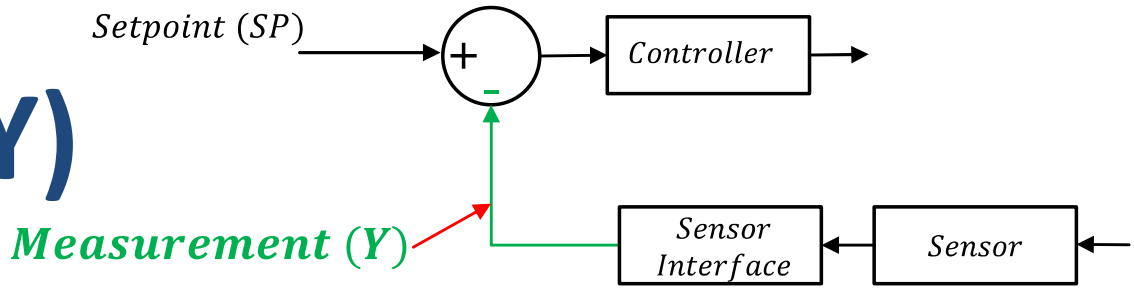


Setpoint (SP)



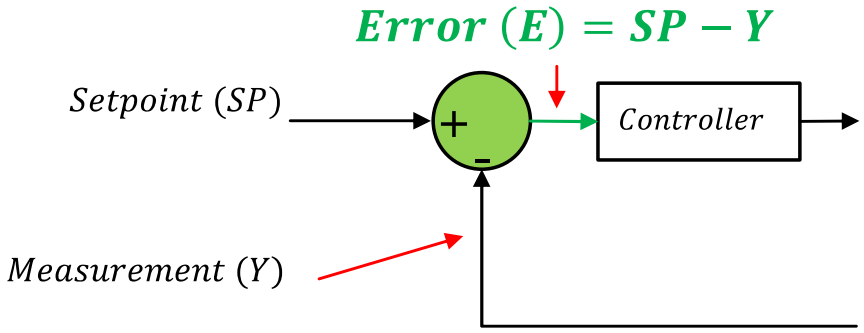
- Also called **Desired Value**, **Set Value** (SV), **Target Value** or **Adjusted Parameter**.
- This value is the **goal** set for the closed loop controller, that is, to what value of the **physical variable** you want to **reach**.
  - For example: in a temperature controller for boiling water, your desired value (Setpoint) for making the water to boil is **100°C**. Then the controller will try to reach this goal.
- The **purpose** of **feedback systems** is to **reach the Setpoint** and **maintain it over the time** even if there are **disturbances** in the context or the environment.

Measurement (Y)



- Also called **Process Variable (PV)** or **tracked metric**.
- Is the **current measured value** of a particular part of a process that is being monitored or controlled.
- In control theory, this variable **is compared with the setpoint** in order to **calculate** the actual **error** of the controlled **system**.
- This variable is the **feedback** obtained **from the system** response to control signals.
  - For example: In the temperature controller for boiling water, the Process Variable is the current or actual temperature of the system, which can be measured using a thermometer. We require this variable in order to verify that the system reached the setpoint correctly (previously set on 100°C).
- This variable is obtained through the **sensors** of the system.

Error (E)

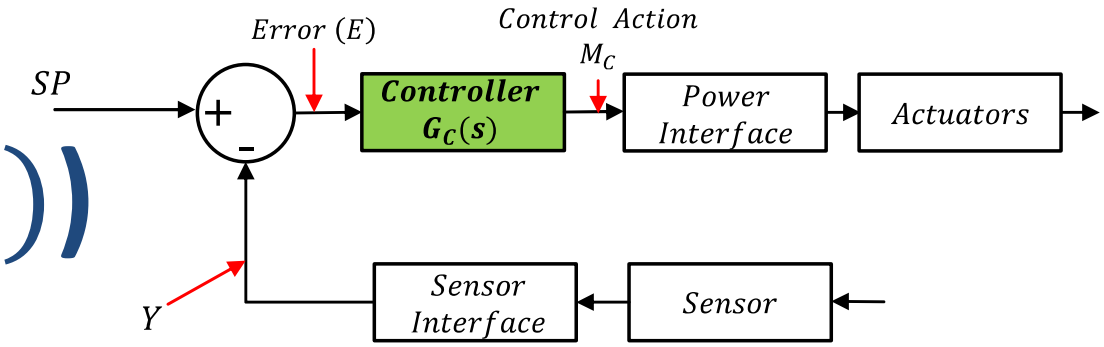


- The error is the difference between the Setpoint and the Measurement:

$$E = SP - Y$$

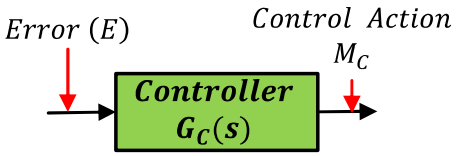
- The job of the controller is to calculate a corrective action based on the actual error:
  - If the **error** is **positive** ($E > 0$), it means that the Setpoint is higher than the Measurement. So the corrective action will need to raise the Y to reach the SP.
  - If the **error** is **negative** ($E < 0$), it means that the Measurement is higher than the Setpoint. In this case, the corrective action needs to lower Y in order to reach SP.
  - If the **error** is approximately **zero** ($E \approx 0$), it means that the Setpoint and Measurement are almost equal, so the controller needs to keep the current corrective action to maintain the system in this state.
- For example: in the temperature controller for boiling water, the Setpoint is 100°C and the Measurement says that the system is at 60°C. The actual error in this system will be $E = 100°C - 60°C \rightarrow E = 40°C$. This means that the controller needs to keep raising the corrective action in order to increase the current temperature.

Controller ($G_C(s)$)



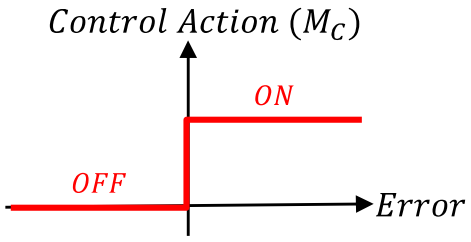
- The control action is calculated using the error.
- The corrective action is reflected on the system through the Power Interface and Actuators.
- Its input is the $Error\ (E)$ and its output is the $Control\ Action\ (M_C)$.

ON/OFF Controller

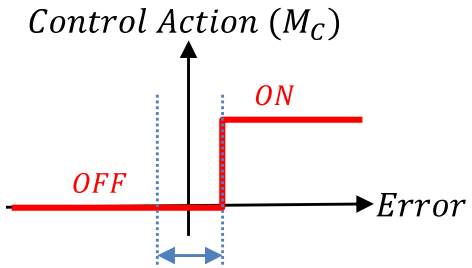


- The most basic controller.
- Consist in a simple ON/OFF switch:
  - If the error is positive ($E > 0$), the corrective action will be to turn fully ON the system plant.
  - If the error is negative ($E > 0$), the corrective action will be to turn fully OFF the system plant.
- The main problem of this controller is that the system never settles down to a steady state, it oscillates constantly and rapidly betwen its two extreme states (Fully ON or Fully OFF) which can damage the actuator.
- To improve this rapid change, a dead zone or an hysteresis can be implemented:
  - Dead Zone: The controller will not send a signal to the plant unless the error exceeeds some threshold value.
  - Hysteresis: The controller maintains the same corrective action while the error switches from positive to negative after a certain threshold value.

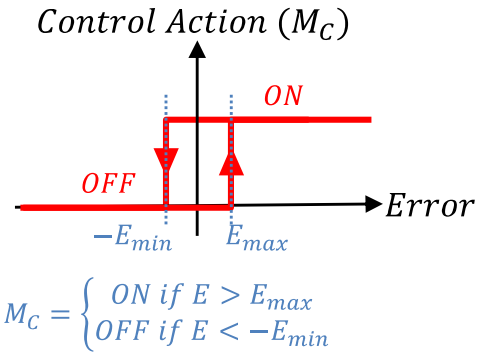
Normal ON/OFF controller



Deadzone ON/OFF controller



Hysteresis ON/OFF controller

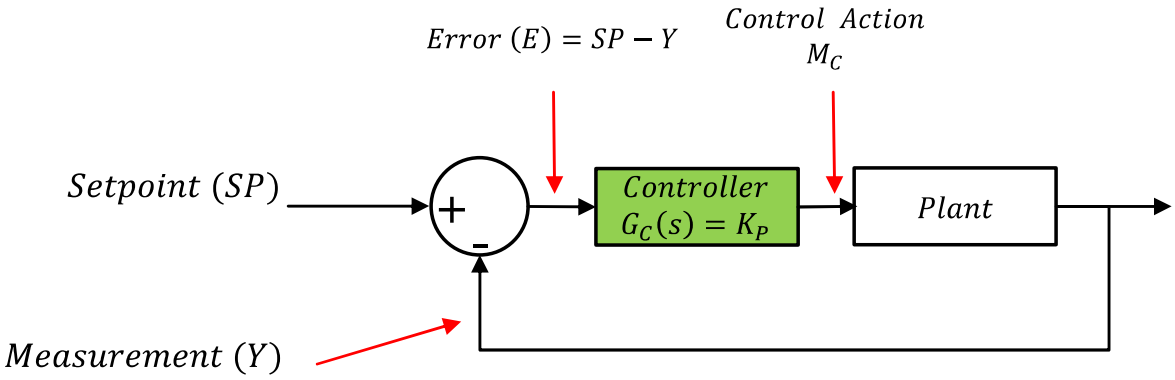


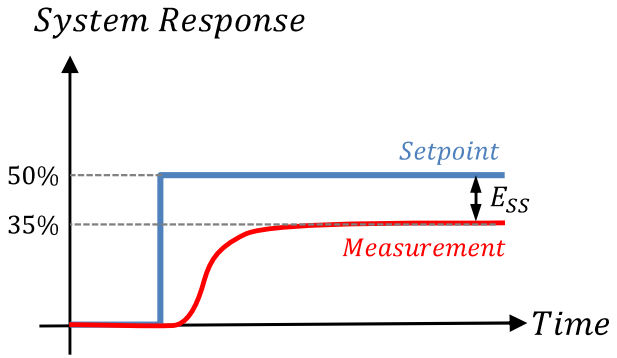
Proportional Controller

- The magnitude of the corrective action depend on the magnitude of the error.
- In other words, the Control Action (M_C) will be directly proportional to error signal.
  - If the error increases, the control action increases.
  - If the error decreases, the control action decreases.

$$m_C(t) = K_P \cdot e(t) \qquad Where\ K_P > 0\ and\ is\ the\ proportional\ constant$$

- They are insufficient to eliminate error in steady state (E_{SS}) entirely:
  - **Proportional droop**: The system output or **measurement** (Y) will always be **less** than the desired **setpoint** value (SP).
  - The E_{SS} can be reduced by increasing the proportional constant (K_P) but doing can make the system **unstable**.



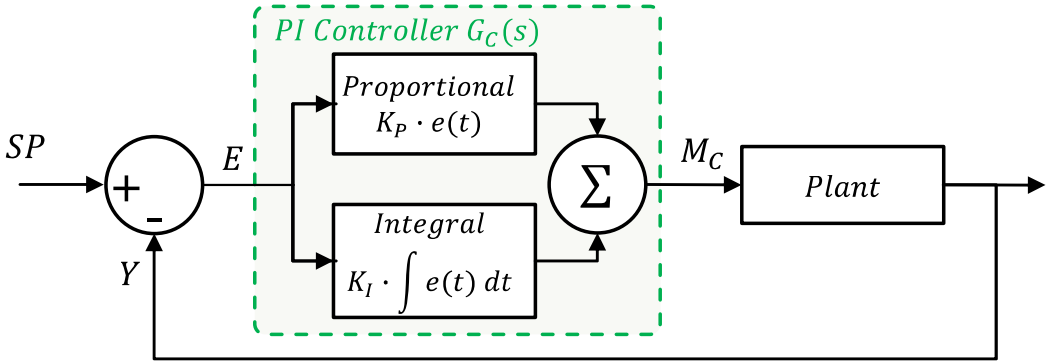


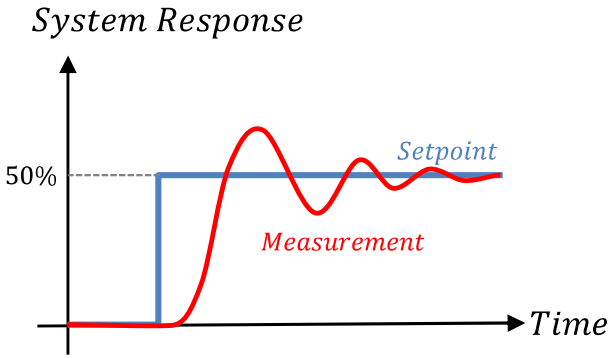
Proportional-Integral Controller (PI)

- The most frequent used controller.
- Adding an Integral part to the controller eliminates the proportional droop amplifying the accumulated error over time and adding it to the controller action.
- The integral part calculates its control action using the accumulated error over a specifyed time, keeping track of the PAST of the system.
- The Control Action (M_C) with proportional and integral control is:

$$m_C(t) = K_P \cdot e(t) + K_I \int_0^T e(t)dt \qquad Where\ K_P > 0\ (Proportional\ constant)\ and\ K_I > 0\ (Integral\ constant)$$

- This controller may introduce oscillations.





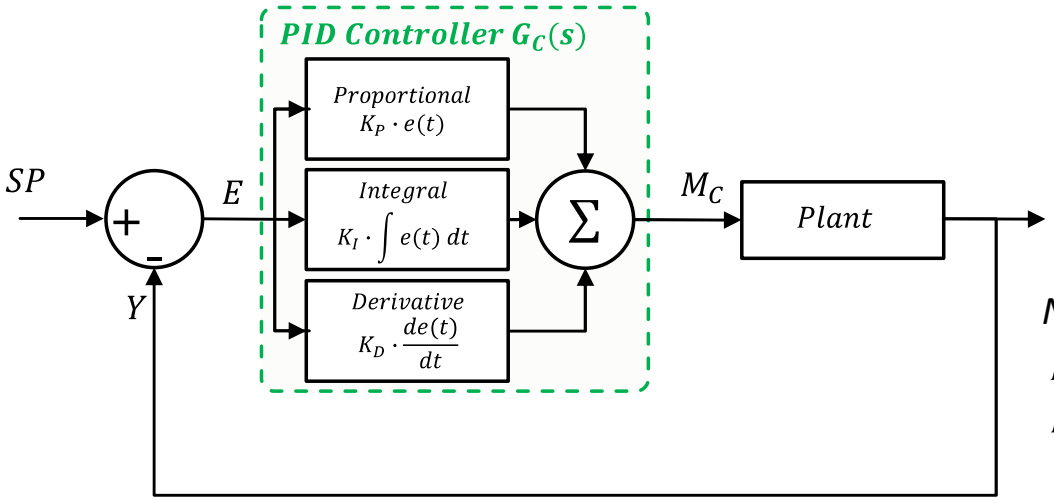
Proportional-Integral-Derivative Controller (PID)

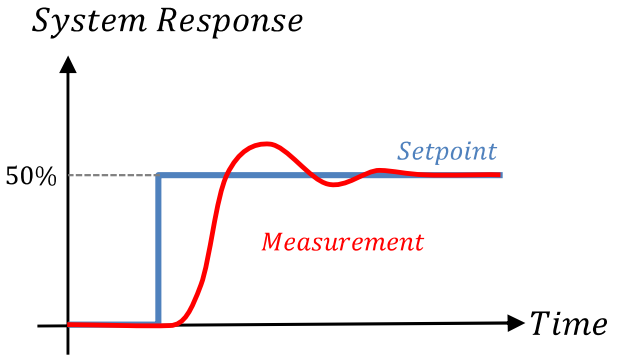
- The most convenient controller for certain applications.
- Adding an Derivative part to the controller counteracts error growth chance in the FUTURE of the system.
- The derivative part calculates its control action using the error change with respect to the time.
  - For discrete-time computer implementation, the derivate is a difference between the actual error and the previous error in a period of time.
- The Control Action (M_C) with proportional, integral and derivative control is:

$$m_C(t) = K_P \cdot e(t) + K_I \int_0^T e(t)dt + K_D \frac{de(t)}{dt}$$

$$Where\ K_P > 0\ (Proportional\ constant), K_I > 0\ (Integral\ constant)\ and\ K_D > 0\ (Derivative\ constant)$$

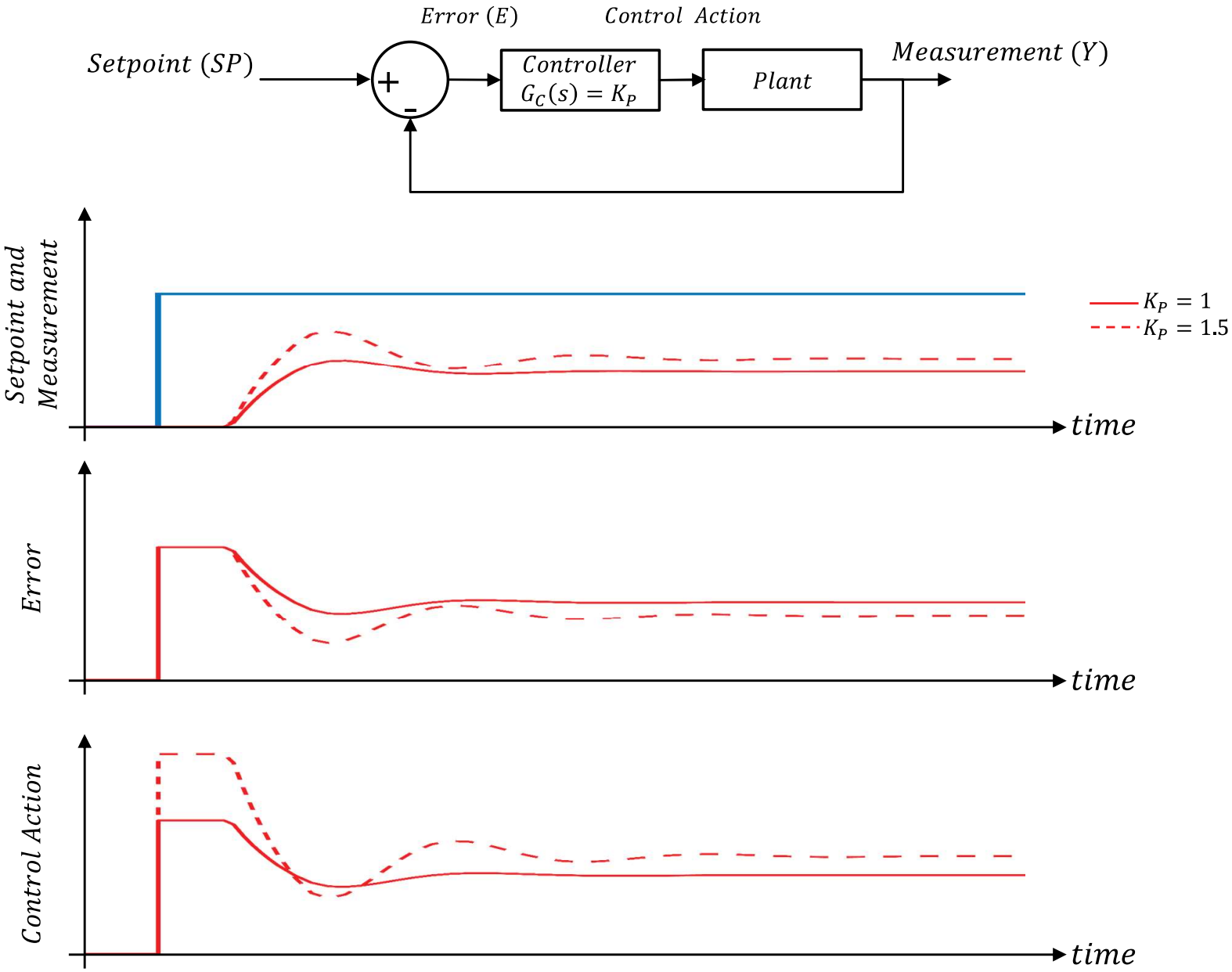
- This controller may be very sensitive to electronic noise and produce unstable behavior.





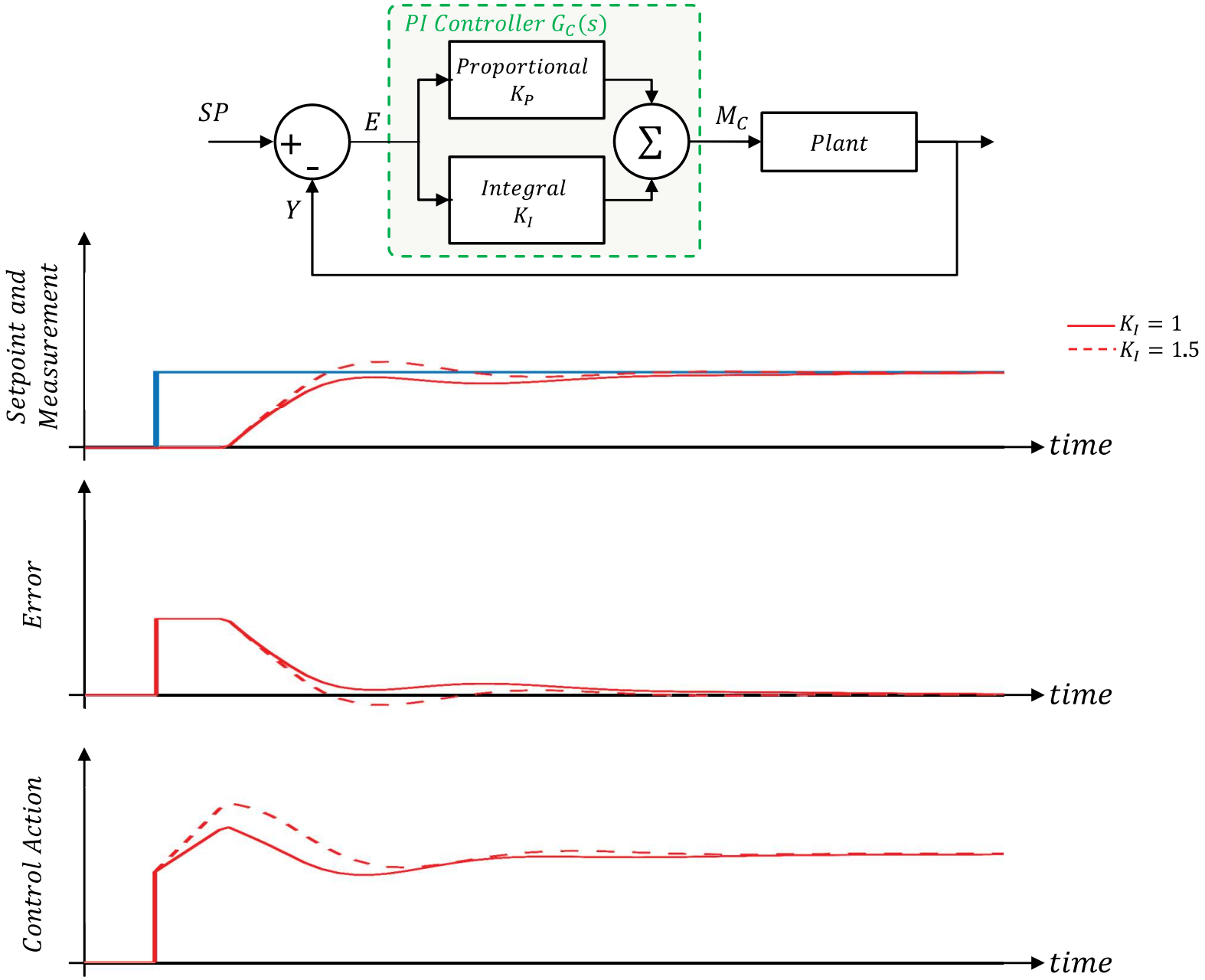
Proportional influence in system dynamics

- Proportional doesn't correct the steady state error.
  - Increasing it reduces the error but increases the oscillations



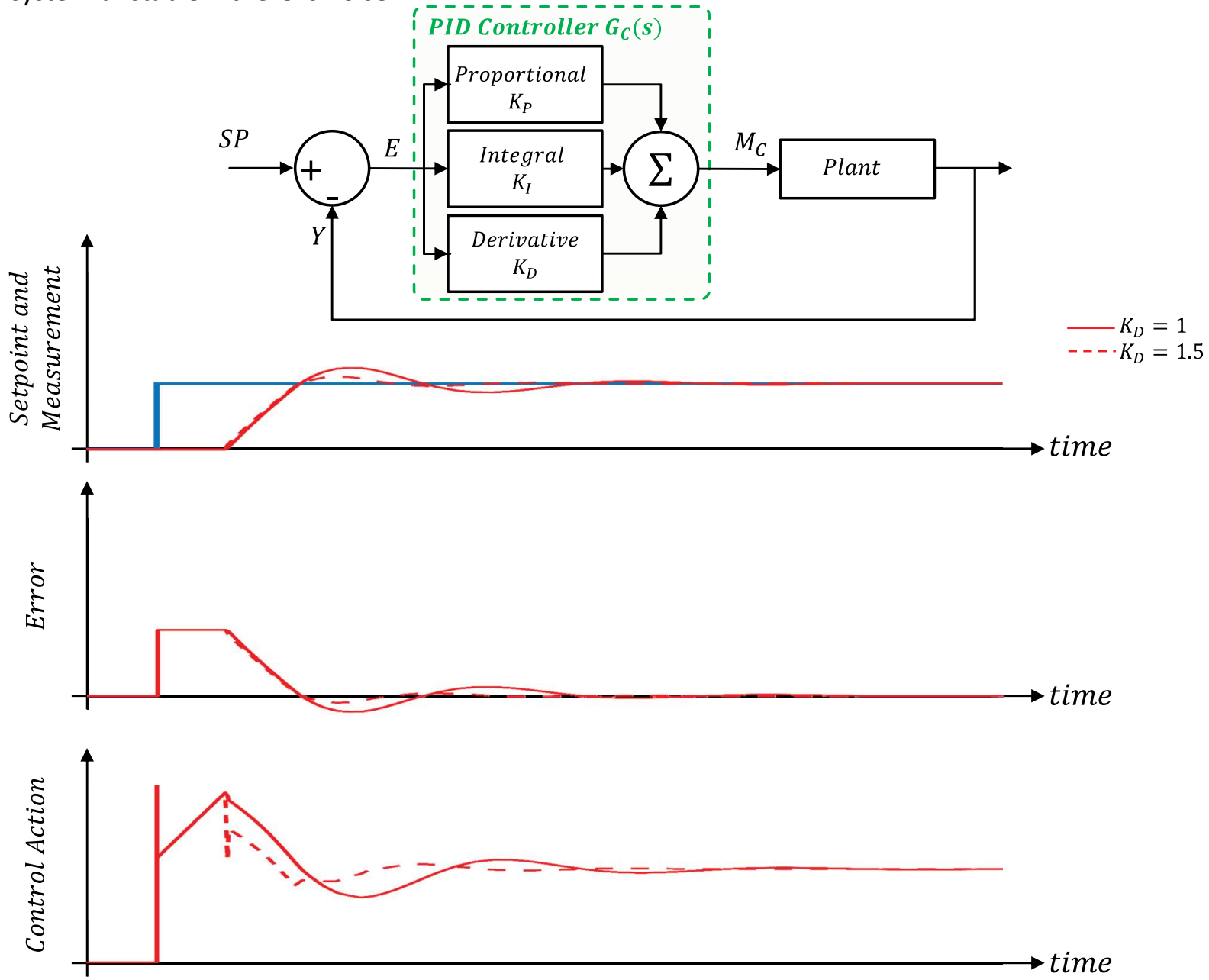
Proportional-Integral influence in system dynamics

- Integral corrects the steady state error.
  - Increasing it reduces stablishing time but after a certain value it will start to increase the stablishing time and oscillations.

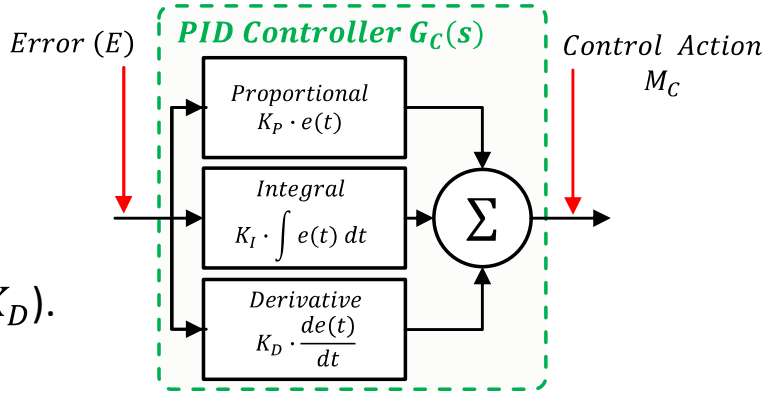


Proportional-Integral-Derivative influence in system dynamics

- Derivative gives speed to the system by anticipating the future.
  - Increasing it reduces stablishing time but makes the system very sensitive to changes, which can make the system unstable if there is noise.

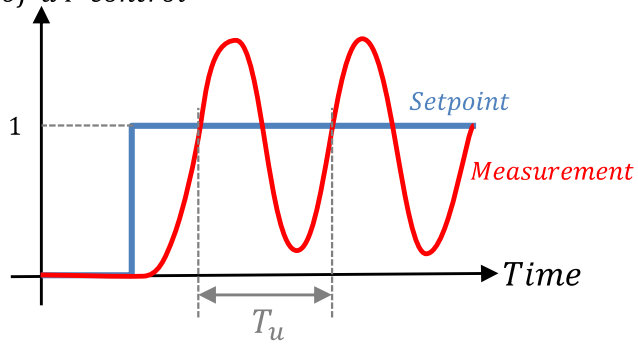


PID Controller Tuning



- Tuning is used for calculating the gain parameters (K_P, K_I *and* K_D).
- The main objectives of tuning are:
  - Guarantying stability.
  - Optimizing behavior of the system depending on the requirements of the situation.
- Tuning each gain parameter can change the system behavior as follows:
  - Increasing K_P: increases speed, decreases stability and enhances noise in controller's action.
  - Increasing K_I: decreases speed, decreases stability, reduces noise in controller's action, eliminates steady-state errors more quickly and increases the tendency to oscillate.
  - Increasing K_D: increases speed, increases stability, strongly enhances noise in controller's action.
- There are heuristic methods for tuning like the Ziegler-Nichols rules. This methods allows to calculate a stable controller but with a fixated performance that usually is acceptable for most of the applications.
- Ziegler Nichols Tuning Method:
  1. Set the integral (K_I) and derivative (K_D) gains to zero (Only proportional controller).
  2. Start increasing the proportional gain (K_P) until it reaches the ultimate gain K_u (this happens when the measurement of the system response, to a unit step SetPoint, has stable consistent oscillations, increasing this gain more than this value will cause the system to be unstable).
  3. Measure the period T_u of the oscillations and set the PID controller gains according to the following table:

System Response to ultimate gain K_u of a P control



Control Type	K_P	T_I	T_D
P	$0.5K_u$		
PI	$0.45K_u$	$T_u/1.2$	
PID	$0.6K_u$	$T_u/2$	$T_u/8$

Note:

$$K_I = \frac{K_P}{T_I}$$

$$K_D = K_P \cdot \tau_D$$

Control Tuning – Step by Step

System identification

1. Implement a first version of the system connecting all the sensors, actuators, interfaces and required processors with corresponding software.
2. Inject a setpoint to the system of a desired value.
3. Report since the setpoint injection a datalog until the system reaches stability or steady state (only applies for stable systems)
4. Use a system identification software using as input the previous generated datalog in order to acquire a mathematical model of the system in transfer function form.
5. Tune a controller using the previous model and acquire the control parameters.
6. Test the tuned controller through simulation and verify its system dynamics performance (Maximum overshoot, stabilization time).
7. Adjust the parameter until the performance meets the desired requirements.
8. Implement a second version of the system adding the control parameters previously tuned and the required control algorithm.
9. Inject a setpoint to the system and evaluate its behavior in terms of the system dynamics performance.
10. Tune again the control parameters in order to improve the system performance in real implementation.