Le respirateur MakAir

Étude d’un système asservi

Diack Abdou-Aziz
Numéro d’inscription: 48001

Présentation du projet MakAir

Répond à une demande urgente en proposant un système:

- à faible coût
- transportable
- projet en accès libre

Réuni plus de 250 bénévoles dont l'essentiel sont:

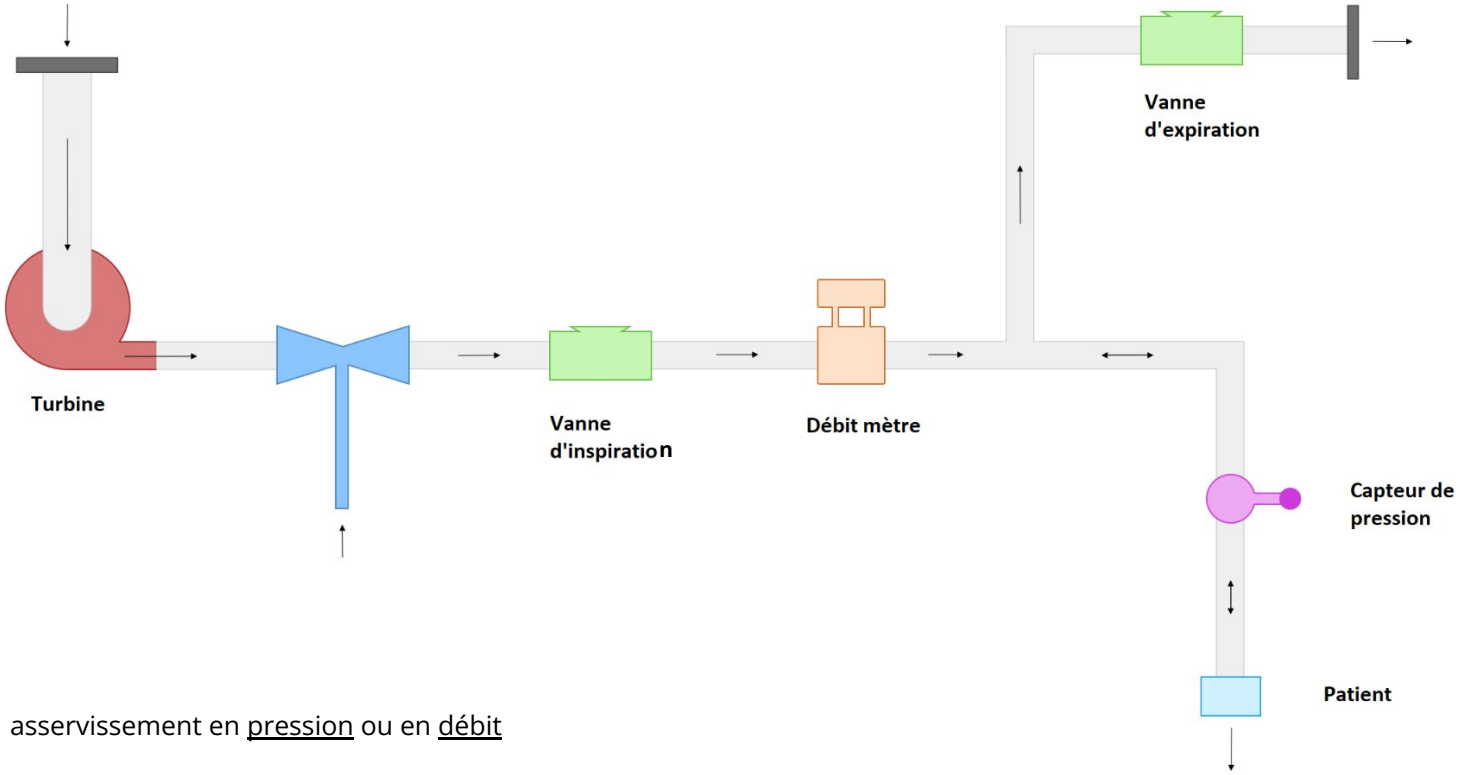
- des professionnels de santé
- des chercheurs
- des ingénieurs

MakAir

Contact: Gabriel Moneyron, ingénieur du projet

Fonctionnement du système

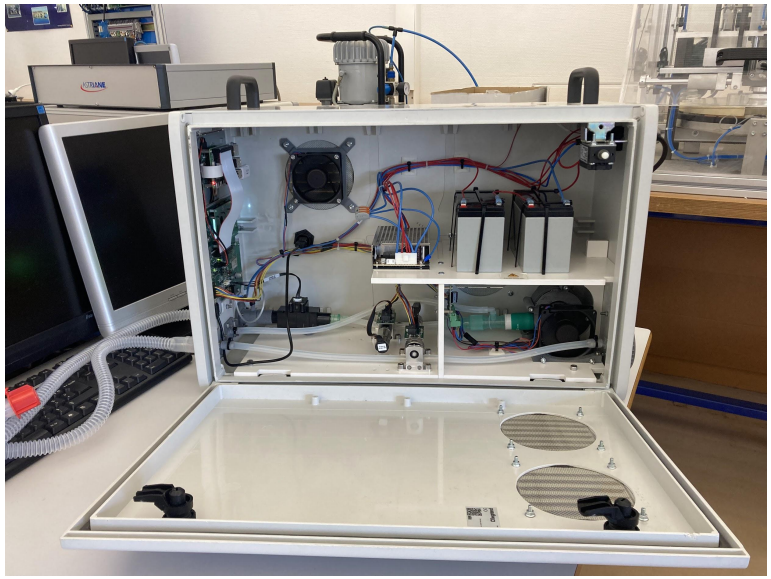
Schéma structurel du circuit pneumatique



asservissement en pression ou en débit

source: GitHub - makers-for-life/makair

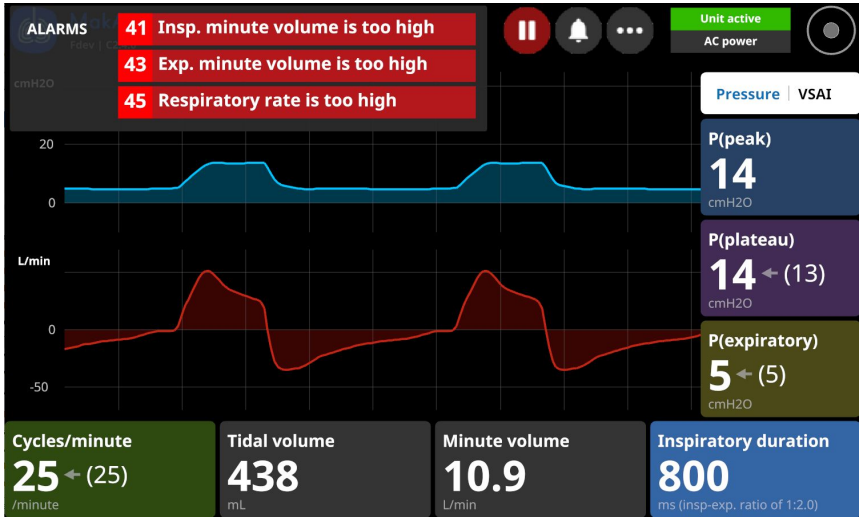
Intérieur du Makair



Ecran de contrôle du Makair

Pression dans les poumons →

débit →



Comment modéliser le système asservi MakAir?

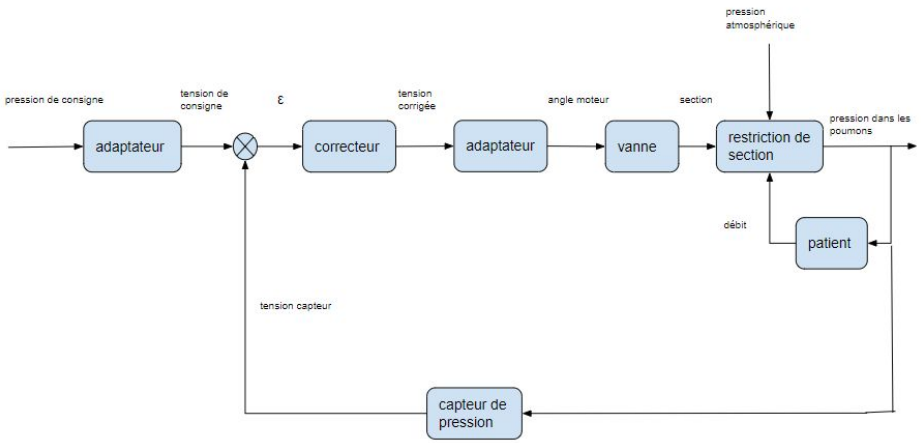
Sommaire

Il est à noter que notre but est d'obtenir des lois linéaires pour pouvoir passer dans le domaine de Laplace

1) La chaîne d'expiration

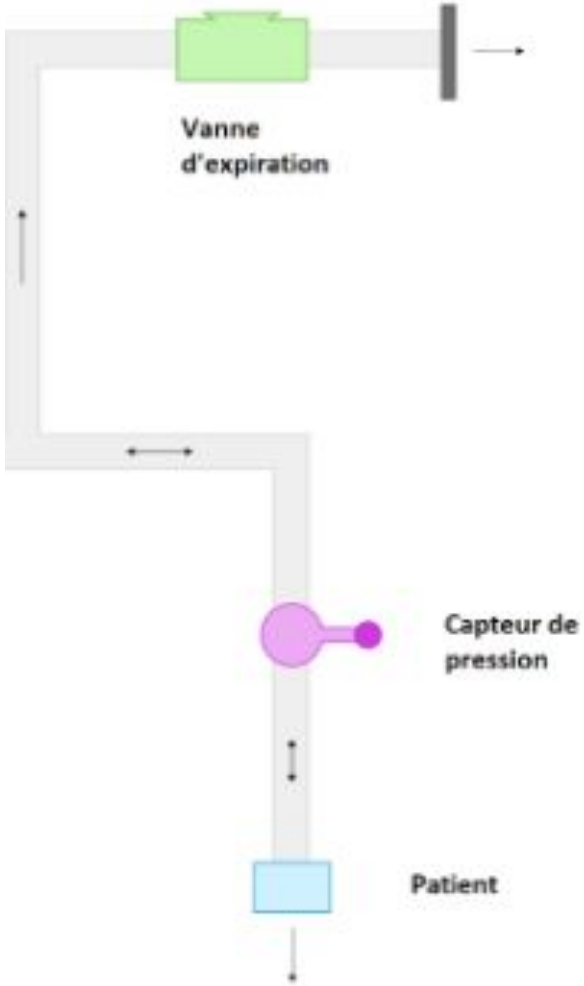
On assimile la chaîne d'expiration au schéma-bloc suivant, on va s'y intéresser bloc par bloc

Schéma de bloc correspondant



- On s'intéresse à l'asservissement en pression
- On cherche les lois entrée-sortie des blocs
- On va chercher analytiquement les coefficients idéaux pour le correcteur

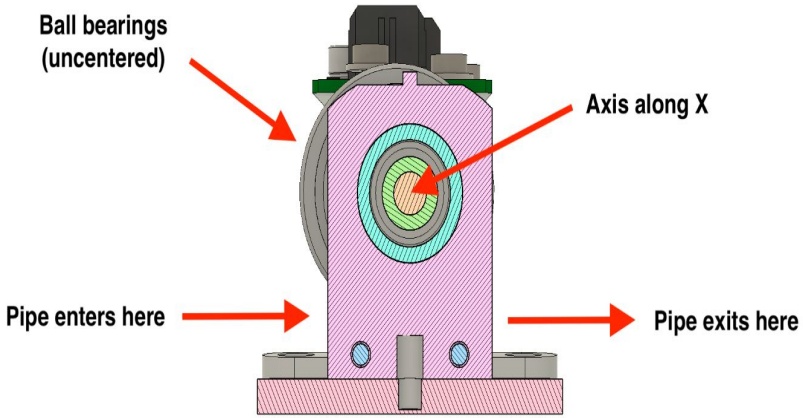
Schéma de la chaîne d'expiration



sources : GitHub - makers-for-life/makair: 🫁 The world's first open-source ventilator tested on human patients. Mass-producible at a low cost (~2000€).

2) La valve de pression

Schéma de la valve de pression



Valve de pression

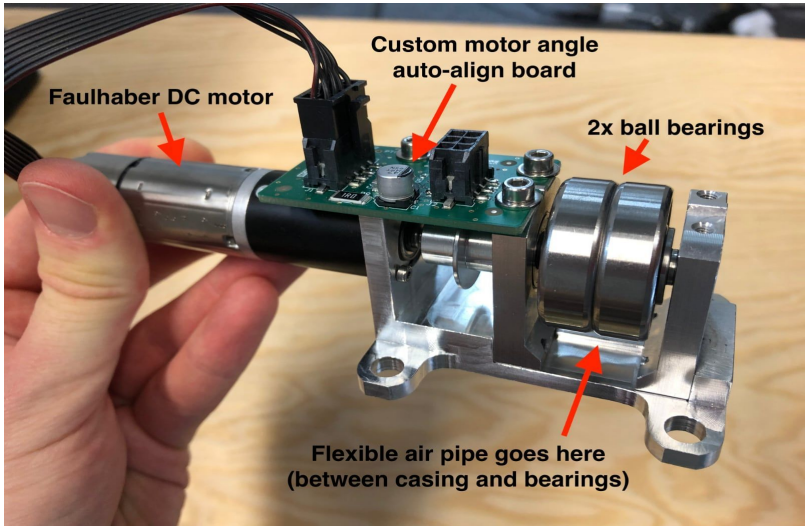
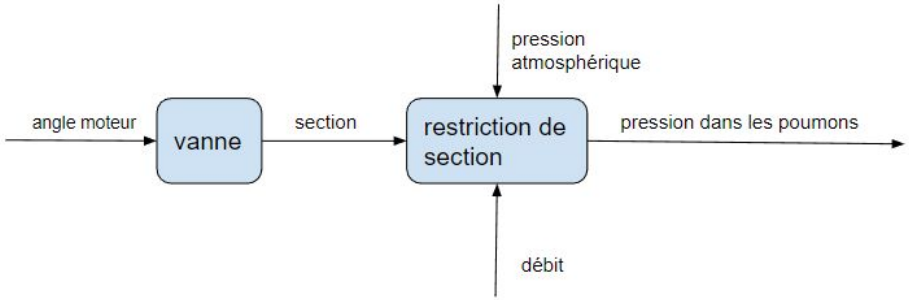


Schéma bloc correspondant



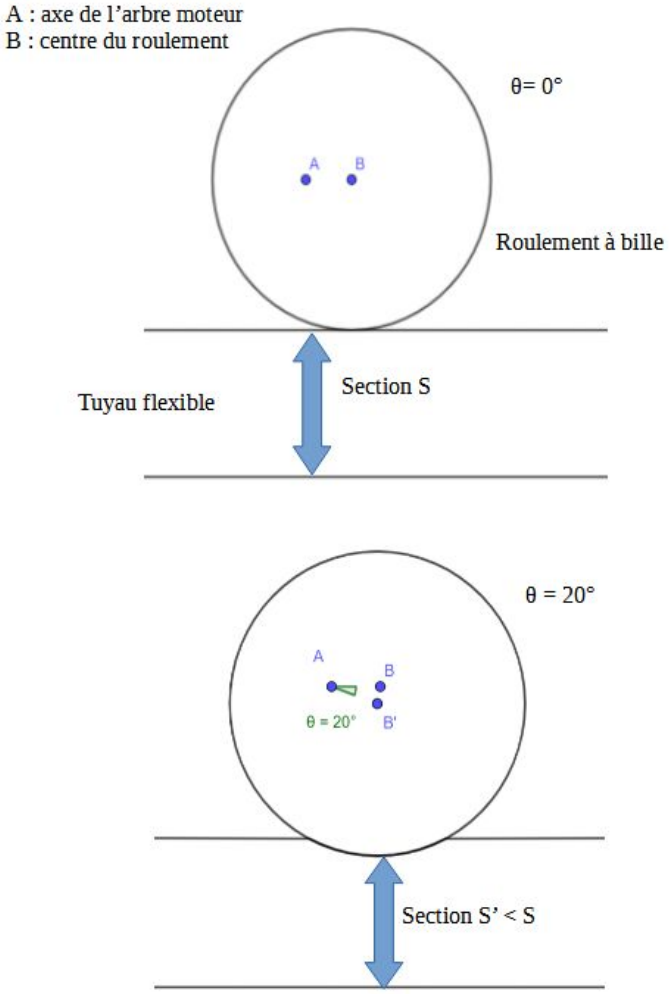
Explication du fonctionnement de la valve utilisée dans le système

sources :GitHub - makers-for-life/makair: 🫁 The world's first open-source ventilator tested on human patients. Mass-producible at a low cost (~2000€).

2-1) Loi géométrique

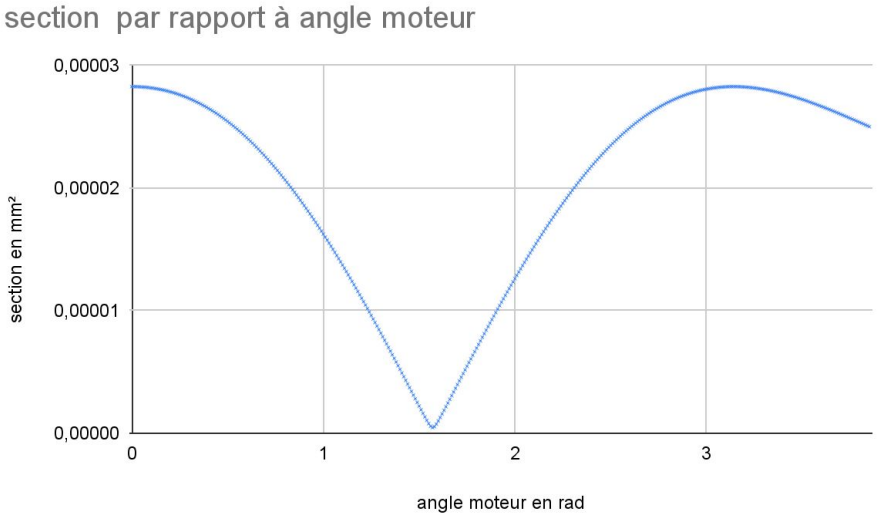
On a supposé que la section du tuyau devenait elliptique quand le roulement à bille appuie dessus, on s'est ensuite servi des lois géométrique dans une ellipse

Schéma de l'action du roulement sur le tuyau



Loi de la section du tuyau

$$S = \pi \times (a - d\sin\theta) \times \sqrt{\left(\frac{p}{\pi\sqrt{2}}\right)^2 - (a - d\sin\theta)^2}$$

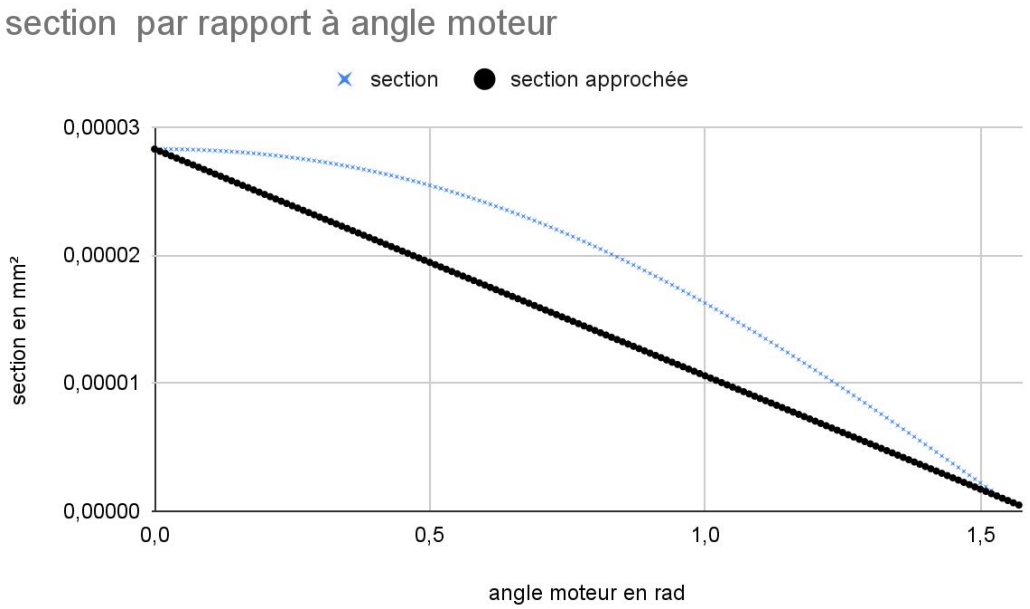


- p périmètre du tuyau (supposé constant)
- a le rayon initial de celui-ci
- théta l'angle entre AB et l'horizontale

2-1) Loi géométrique

On réalise une approximation linéaire de la loi précédente sur la plage angulaire permettant de passer d'un état ouvert à un état fermé

Approximation de notre courbe de section

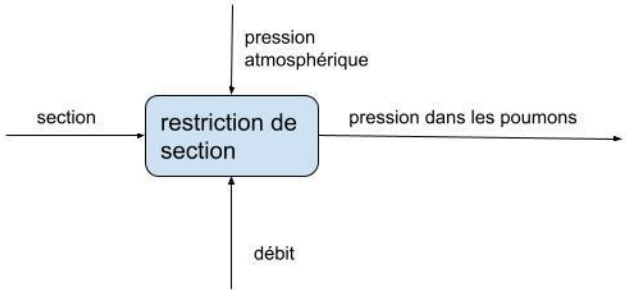


Pour la plage qui nous intéresse, on obtient la loi:

$$s = -1{,}77 \cdot 10^{5}\theta + 2{,}83 \cdot 10^{-5}$$

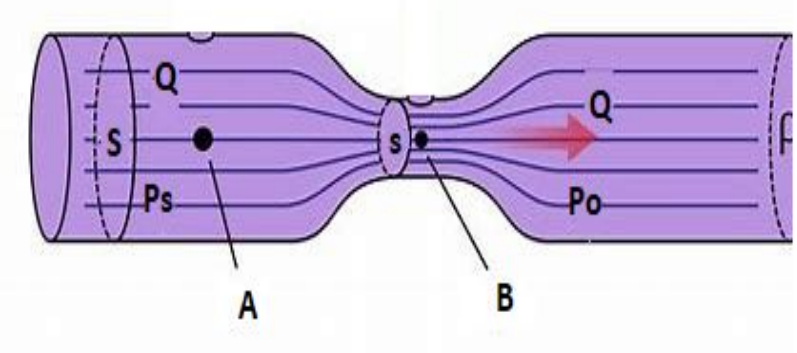
- on pourra passer dans le domaine de Laplace

2-2)Restriction de section



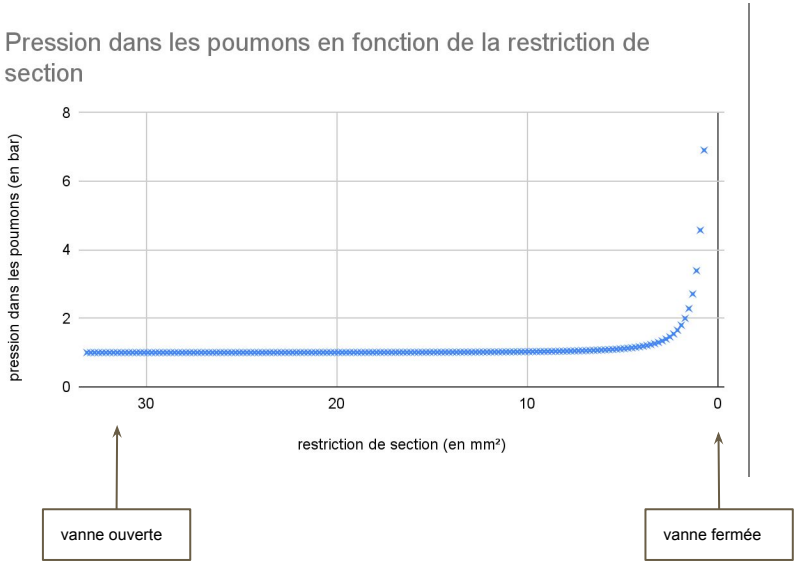
Effet Venturi:

$$Ps = Po + \frac{\mu Q^2}{2}\left(\frac{1}{s^2} - \frac{1}{S^2}\right)$$



Po = pression atmosphérique
Ps = pression dans les poumons
μ = masse volumique de l'air
Q = débit volumique
S = section du tuyau (en A)
s = section restreinte (en B)

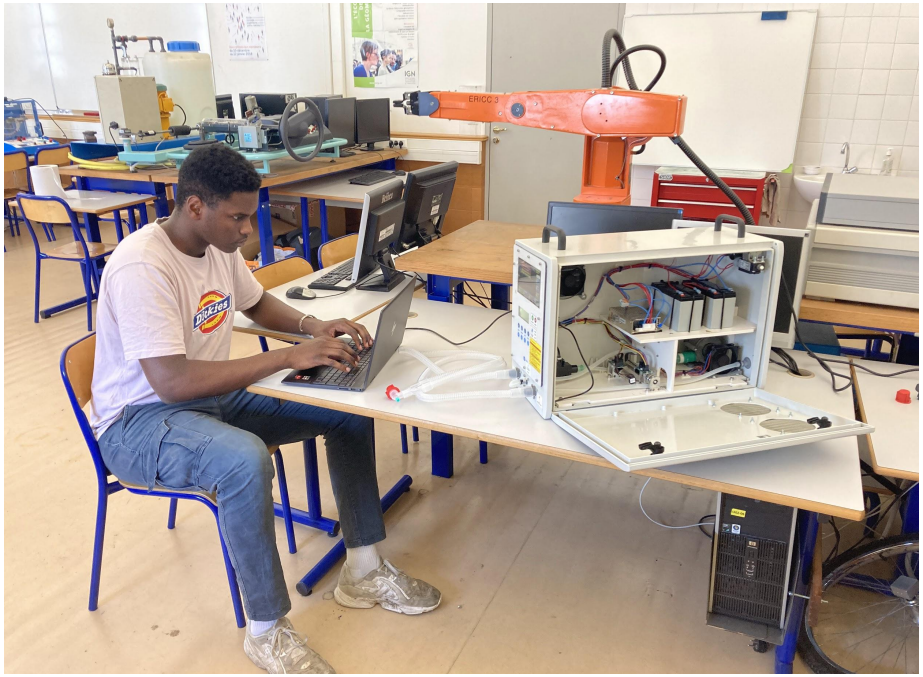
Hypothèse:
- écoulement stationnaire, parfait, incompressible et homogène
- référentiel galiléen
- pas de machine
- poids seule force ext conservation



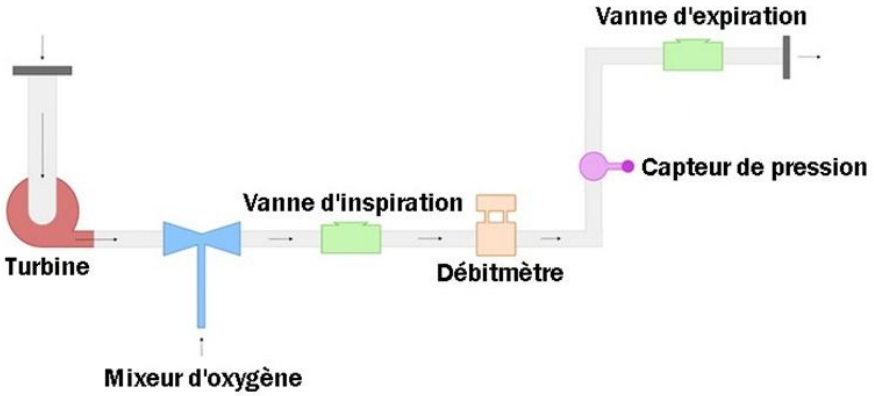
La loi déterminée pour ce bloc n'est pas linéaire et repose sur des hypothèses fortes qu'il convient de vérifier

2-3) Validation expérimentale

Présentation de l'expérience



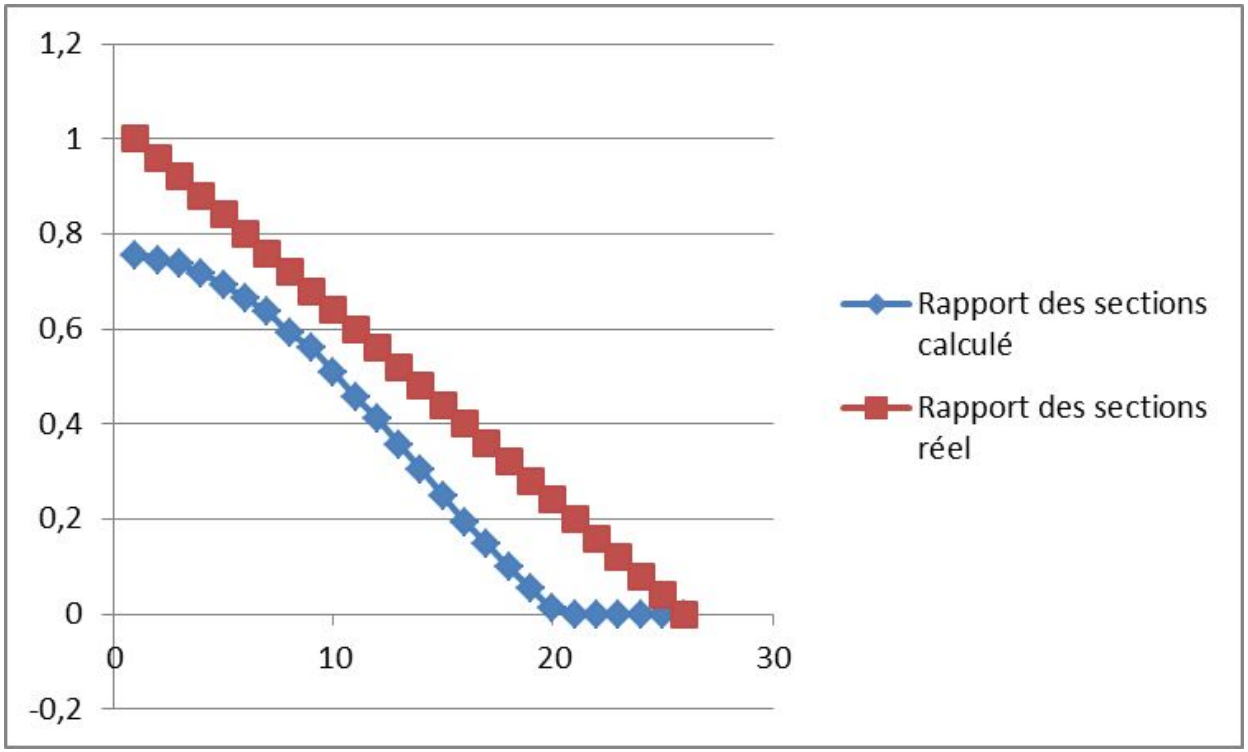
Objectif: Vérifier la loi précédente



On fait varier les différents paramètres intervenants dans la loi précédente sur différents essais

2-3) Validation expérimentale

Comparaison des rapports des sections théoriques et mesurés



→ On compare sur 26 essais le rapport des sections théorique et le rapport des sections prédit par notre loi à partir des différentes mesures

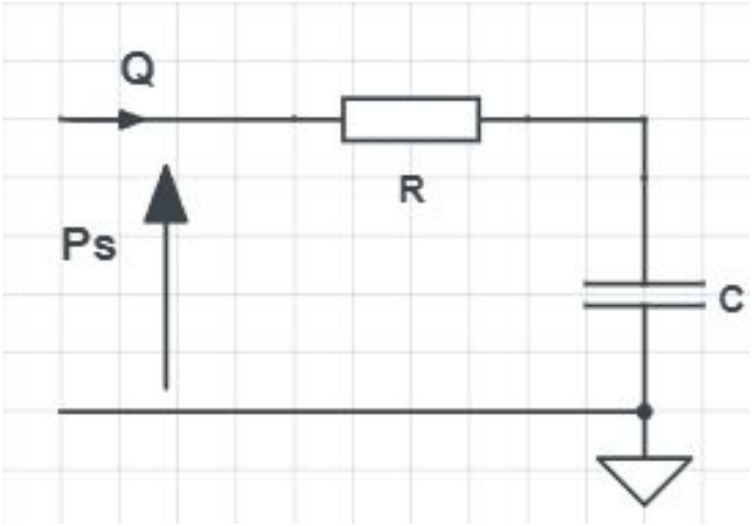
→ On constate des écarts dus aux approximations faites

L'évolution générale reste néanmoins celle attendue, on considère nos hypothèses validées

3) Le patient

Analogie avec un circuit électrique:

$$\frac{dPs}{dt} = \frac{1}{C} Q + R \frac{dQ}{dt}$$



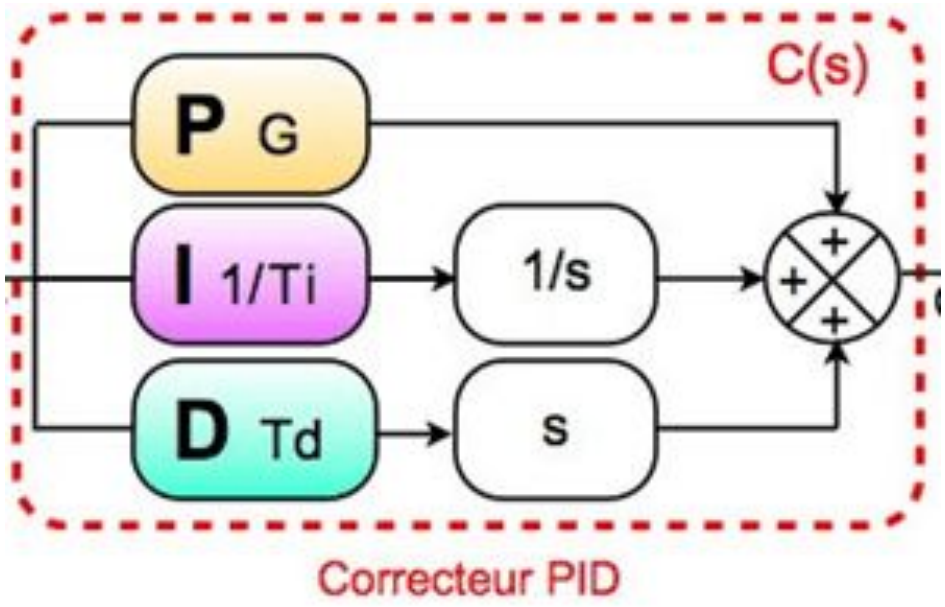
Transformée
de Laplace

$$Q = \frac{Cp}{1+RCp} Ps + \frac{C}{1+RCp} Pi$$

C = capacité hydraulique des poumons
R = résistance hydraulique des tuyaux et de la trachée

Cette partie est issue de la documentation, différente thèse mènent l'étude permettant d'aboutir à ce résultat

4-1) Pourquoi un correcteur PID



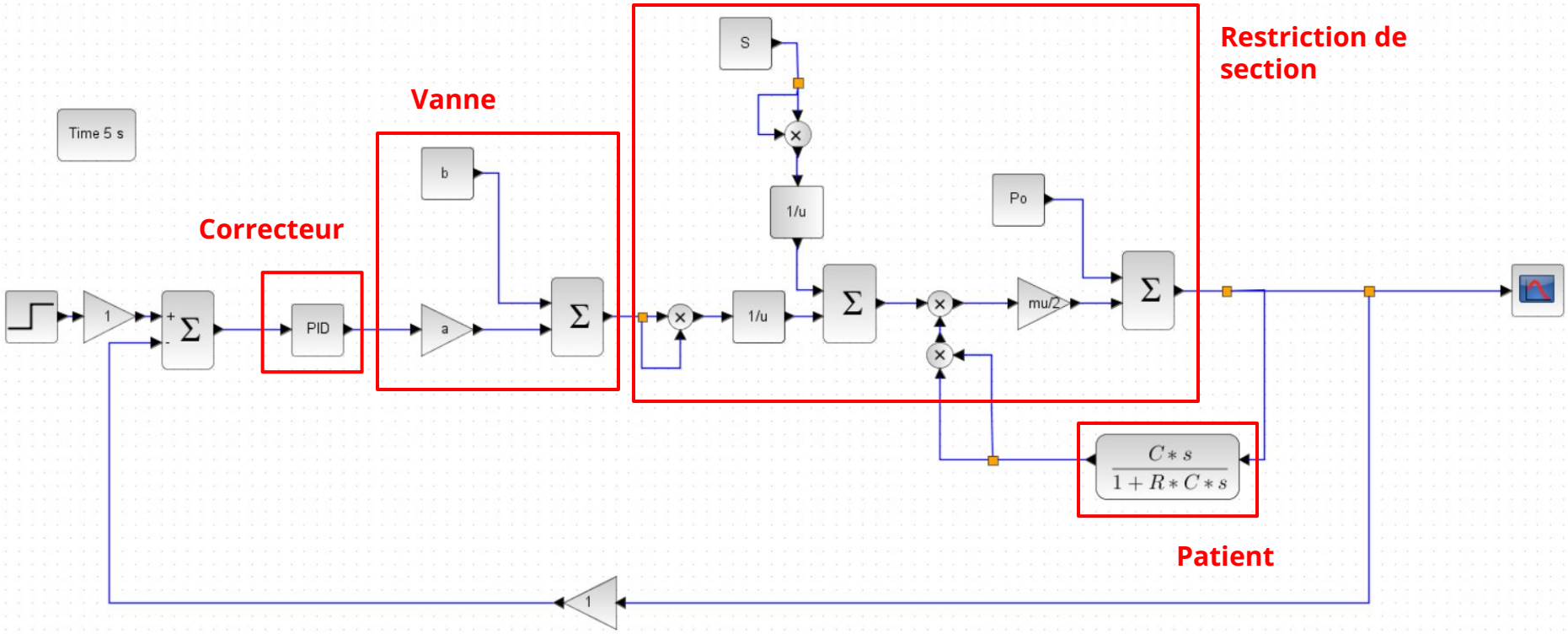
Correcteur PID

<u>Critères à respecter</u>

- Pas de dépassement
- Précision
- Stabilité

Présentation rapide du correcteur utilisé et de la pertinence de son usage dans nombres de systèmes industriels

4-2) Modélisation numérique

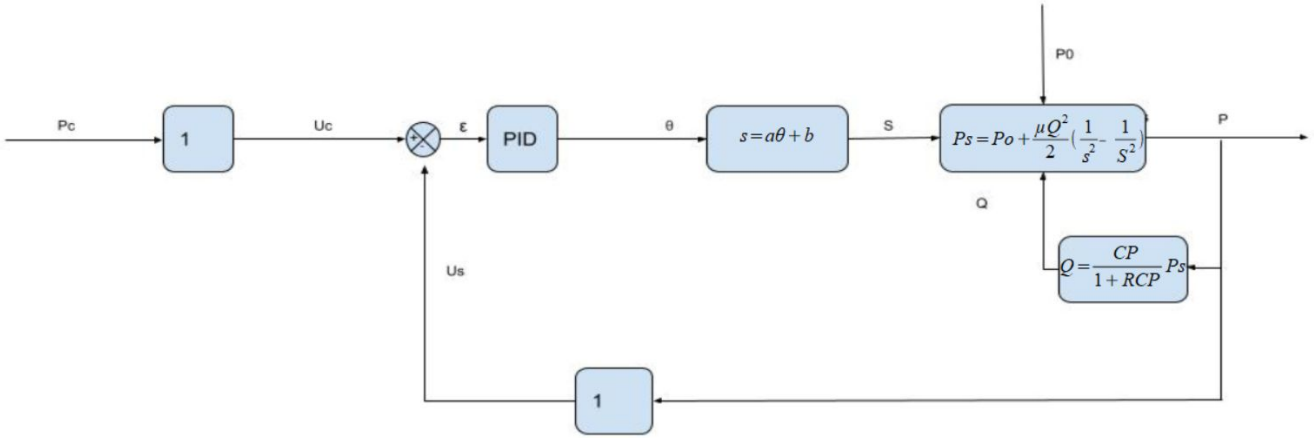


problème: Erreur algébrique et de bouclage → ne trace rien

Une fois nos lois déterminées, on tente de tracer un diagramme de Bode à l'aide de scilab, ce qui n'aboutit pas

5) Conclusion

Modélisation de l'asservissement de l’expiration



Détermination du correcteur

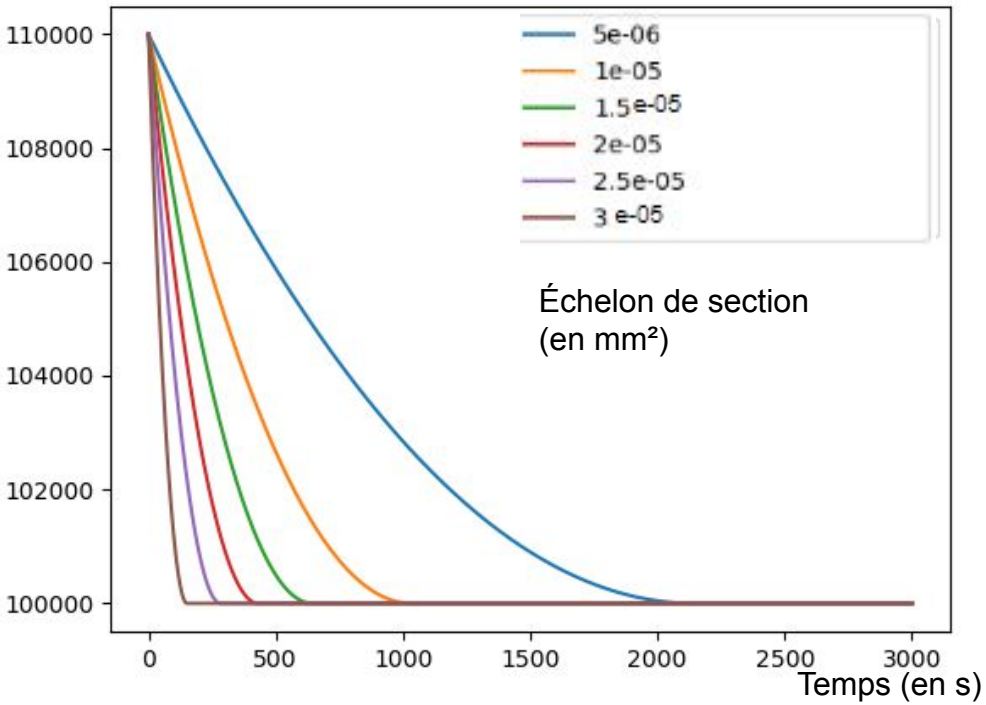
→ Difficultés théoriques pour modéliser le correcteur

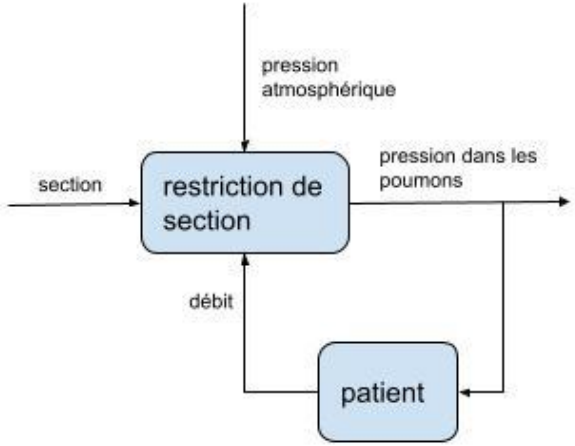
On récapitule les lois trouvées et le schéma bloc final obtenu

5) Conclusion

<u>Evolution de la pression dans les poumons au cour du temps pour un échelon de section</u>

Pression dans les poumons
(en Pa)

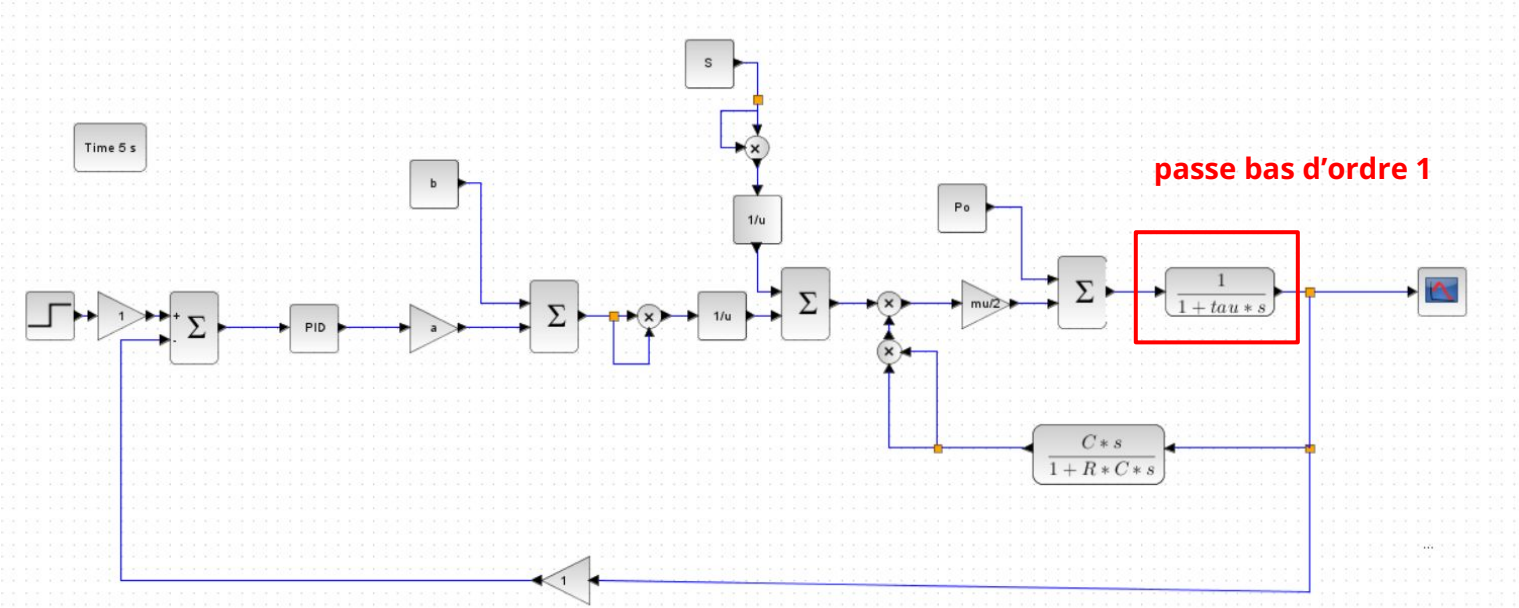




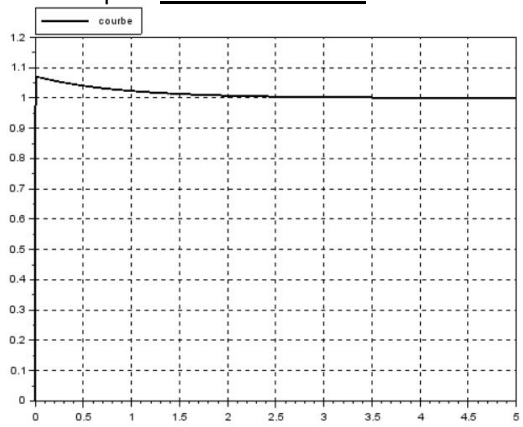
→ Le temps de réponse dépend de la valeur de l'échelon
→ On peut seulement caractériser l'ensemble autour de points de fonctionnement

Piste de recherches: caractériser l'ensemble restriction de section-patient autour de point de fonctionnement, pour obtenir des approximations linéaires autours de ces valeurs de section (graphique obtenu via python)

Solution au problème de bouclage

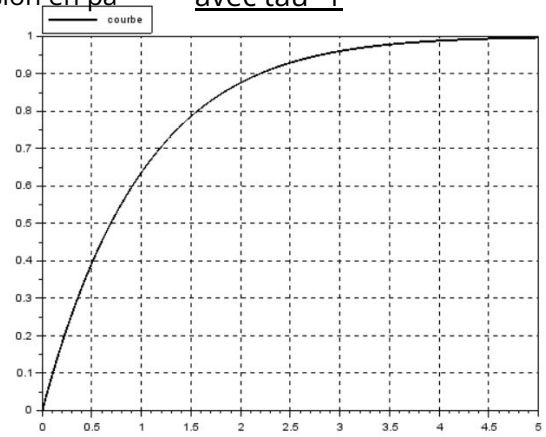


pression en pa avec tau=0.001



temps en s

pression en pa avec tau=1



temps en s

Détermination expérimentale du correcteur

Méthodes de Ziegler et Nichols (empiriques)

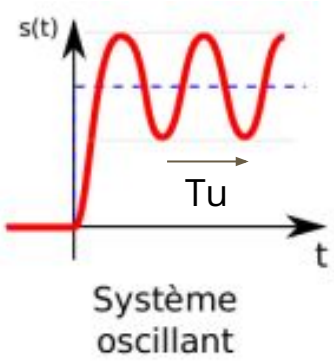
$$C(p) = K(1 + \frac{1}{Ti \times p} + Td \times p)$$

1) Méthode boucle fermée:

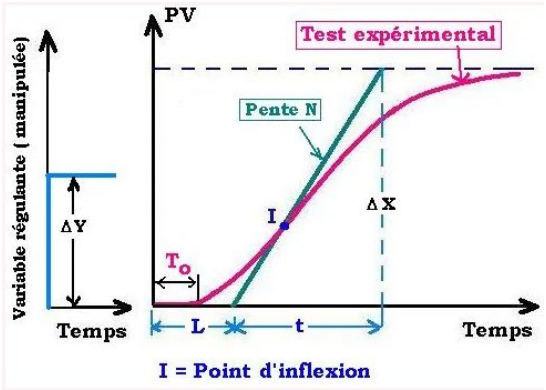
- On trouve le gain du correcteur proportionnel plaçant le système à la limite de sa stabilité noté Ku

-

Type de contrôle	K_p	T_i	T_d
P	$0.5K_u$	-	-
PI	$0.45K_u$	$T_u/1.2$	-
PD	$0.8K_u$	-	$T_u/8$
PID	$0.6K_u$	$T_u/2$	$T_u/8$



2) Méthode boucle ouverte

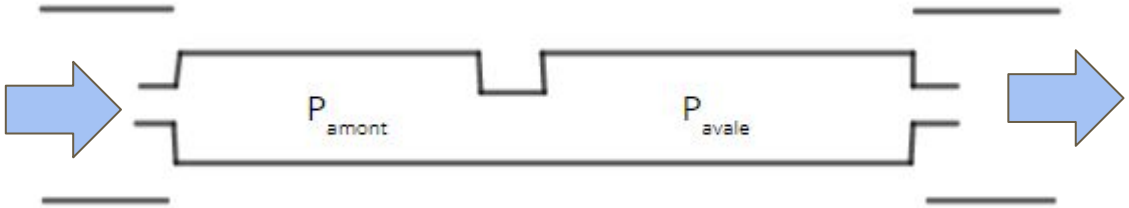


Mode de régulation	Gain G	Ti (rép/min)	Td (min)
P	?Y/NL	*	*
PI	0.9 ?Y/NL	3.33 L	*
PID	1.2 ?Y/NL	2L	0.5 L

sources :https://fr.wikipedia.org/wiki/M%C3%A9thode_de_Ziegler-Nichols, https://scientificsentence.net/Regulation/ZN.html

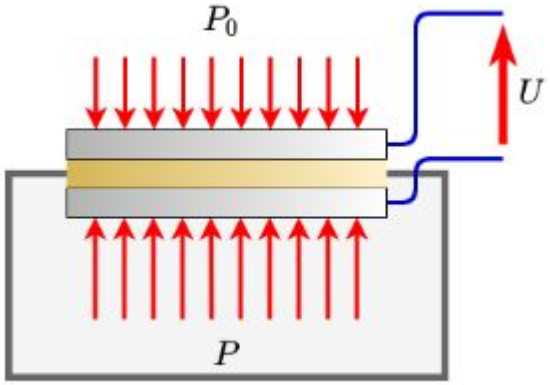
Précision sur les capteurs

Capteur de débit



ΔP *proportionnelle au débit*

Capteur de pression



U *proportionnelle à* ΔF

$$P = \frac{F}{S}$$

sources :https://si.blaisepascal.fr/1t-les-capteurs/

Code Python utilisé pour caractériser la vanne

```
import matplotlib.pyplot as plt

R=800000
S=0.000033
mu=1.2
C=0.00007
Po=100000

def a(s):
    return ((2*R/mu)/(1/(s*s)-1/(S*S)))

def b(s):
    return (2/(mu*(1/(s*s)-1/(S*S))))

def Ps (n,pas,s):
    P=[110000]
    for k in range ( 1, n):
        if P[k-1] > Po :
            P.append(-pas*(1/((1+a(s))*C)*(((abs(Po-P[k-1]))*b(s))**(1/2)))+P[k-1])
        else :
            P.append(pas*(1/((1+a(s))*C)*(((abs(Po-P[k-1]))*b(s))**(1/2)))+P[k-1])
    return(P)

def trace(P,pas,titre):
    X=[k*pas for k in range(len(P))]
    plt.plot(X,P,label=titre)

for k in range(1,6) :
    trace(Ps(15000,0.2,k*0.000005),0.2,k*0.000005)
plt.legend()
plt.show()
```

Code Arduino utilisé pour l'expérience

```
/*********************************************************************************
 * @author Makers For Life
 * @copyright Copyright (c) 2020 Makers For Life
 * @file respirator.cpp
 * @brief Entry point of ventilator program
 ********************************************************************************/

#pragma once

#include "../includes/config.h"
#if MODE == MODE_PROD

// INCLUDES ====================================================================

// External
#include "Arduino.h"
#include <HardwareSerial.h>
#include <IWatchdog.h>
#include <LiquidCrystal.h>

// Internal
#include "../includes/battery.h"
#include "../includes/blower.h"
#include "../includes/buzzer.h"
#include "../includes/buzzer_control.h"
#include "../includes/calibration.h"
#include "../includes/cpu_load.h"
#include "../includes/debug.h"
#include "../includes/end_of_line_test.h"
#include "../includes/keyboard.h"
#include "../includes/main_controller.h"
#include "../includes/main_state_machine.h"
#include "../includes/mass_flow_meter.h"
#include "../includes/parameters.h"
#include "../includes/pressure.h"
#include "../includes/pressure_valve.h"
#include "../includes/rpi_watchdog.h"
#include "../includes/screen.h"
#include "../includes/serial_control.h"
#include "../includes/telemetry.h"
```

```
// PROGRAM =============================================================================

HardwareTimer* hardwareTimer1;  // ESC command
HardwareTimer* hardwareTimer3;  // valves command

HardwareSerial Serial6(PIN_TELEMETRY_SERIAL_RX, PIN_TELEMETRY_SERIAL_TX);

void setup(void) {
    // Nothing should be sent to Serial in production, but this will avoid crashing the program if
    // some Serial.print() was forgotten
    Serial.begin(115200);
    DBG_DO(Serial.println("Booting the system...");)

    startScreen();

    initBattery();
    if (isBatteryDeepDischarged()) {
        displayBatteryDeepDischarge();

        // Heartbeat fatal error periodically
        while (true) {
            sendBatteryDeeplyDischargedFatalError(getBatteryLevelX100());
            delay(1000);
        }
    }

    initTelemetry();
    sendBootMessage();

    // Timer for valves
    hardwareTimer3 = new HardwareTimer(TIM3);
    hardwareTimer3->setOverflow(VALVE_PERIOD, MICROSEC_FORMAT);

    // Valves setup
    inspiratoryValve = PressureValve(hardwareTimer3, TIM_CHANNEL_INSPIRATORY_VALVE,
                                     PIN_INSPIRATORY_VALVE, VALVE_OPEN_STATE, VALVE_CLOSED_STATE);
    inspiratoryValve.setup();
    hardwareTimer3->resume();
    expiratoryValve = PressureValve(hardwareTimer3, TIM_CHANNEL_EXPIRATORY_VALVE,
                                    PIN_EXPIRATORY_VALVE, VALVE_OPEN_STATE, VALVE_CLOSED_STATE);
```

```
    expiratoryValve.setup();
    hardwareTimer3->resume();

    // Blower setup
    hardwareTimer1 = new HardwareTimer(TIM1);
    hardwareTimer1->setOverflow(ESC_PPM_PERIOD, MICROSEC_FORMAT);
    blower = Blower(hardwareTimer1, TIM_CHANNEL_ESC_BLOWER, PIN_ESC_BLOWER);
    blower.setup();

    // Init controllers
    mainController = MainController();
    alarmController = AlarmController();

    // Init sensors
    inspiratoryPressureSensor = PressureSensor();
#ifdef MASS_FLOW_METER_ENABLED
    (void)MFM_init();
#endif

    // Setup pins of the microcontroller
    pinMode(PIN_PRESSURE_SENSOR, INPUT);
    pinMode(PIN_BATTERY, INPUT);
    pinMode(PIN_ENABLE_PWR_RASP, OUTPUT);
    pinMode(PIN_LED_START, OUTPUT);
    pinMode(PIN_LED_RED, OUTPUT);
    pinMode(PIN_LED_YELLOW, OUTPUT);
    pinMode(PIN_LED_GREEN, OUTPUT);
    pinMode(PB12, INPUT);

    // Turn on the Raspberry Pi power
    digitalWrite(PIN_ENABLE_PWR_RASP, PWR_RASP_ACTIVE);
#if DEBUG != 0
    rpiWatchdog.disable();
#endif

    // Activate test mode if a service button is pressed
    // The end of line test mode cannot be activated later on.
    // Autotest inputs: the service button on PB12, top right of the board's rear side
    if (HIGH == digitalRead(PB12)) {
```

```
        eolTest.activate();
        displayEndOfLineTestMode();
        while (HIGH == digitalRead(PB12)) {
            continue;
        }
    }

    // Catch potential Watchdog reset
    // cppcheck-suppress misra-c2012-14.4 ; IWatchdog.isReset() returns a boolean
    if (IWatchdog.isReset(true)) {
        // Run a high priority alarm
        BuzzerControl_Init();
        Buzzer_Init();
        Buzzer_High_Prio_Start();
        displayWatchdogError();

        // Heartbeat fatal error periodically
        while (true) {
            sendWatchdogRestartFatalError();
            delay(1000);
        }
    }

    initKeyboard();
    BuzzerControl_Init();
    Buzzer_Init();
    Calibration_Init();

    if (!eolTest.isRunning()) {
        /*mainStateMachine.setupAndStart();

        // Init the watchdog timer. It must be reloaded frequently otherwise MCU resests
        IWatchdog.begin(WATCHDOG_TIMEOUT);
        IWatchdog.reload();*/
    } else {
        // eolTest.setupAndStart();
    }
}
```

```
}

// cppcheck-suppress unusedFunction
void loop(void) {

    inspiratoryValve.open();
    expiratoryValve.close();

    int32_t blower_values[6] = {1800, 1440, 1080, 720, 360, 180};
    int32_t expiratory_angle_values[4] = {62,90,20, 0};
    int32_t inspiratory_angle_values[4] = {62,90,20, 0};

    Serial.println("time \tblower speed (0 - 1800)\t inspiratoryValveOpenning(0 - 125)\texpiratoryValveOpenning(0 - 125)\tinspiratoryFlow(mL / min)\texpiratoryFlow(mL / min)\tpressure(mmH2O
    for (int32_t i = 0; i < 6; i++) {
        blower.runSpeed(blower_values[i]);

        for (int32_t k = 0; k < 4; k++) {
            expiratoryValve.open(expiratory_angle_values[k]);
            for (int32_t l = 0; l < 4; l++) {
                inspiratoryValve.open(inspiratory_angle_values[l]);
                inspiratoryValve.execute();
                expiratoryValve.execute();
                delay(10000);
                int32_t sumFlow = 0;
                int32_t sumFlowexpi = 0;
                int32_t sumPressure = 0;
                for (int32_t j = 0; j < 100; j++) {
                    expiratoryValve.execute();
                    inspiratoryValve.execute();
                    sumFlow += MFM_read_airflow();
                    sumFlowexpi += MFM_expi_read_airflow();
                    sumPressure += inspiratoryPressureSensor.read();
                    delay(10);
                }
                }
                Serial.print(millis());
                Serial.print("\t");
                Serial.print(blower_values[i]);
                Serial.print("\t");
                Serial.print(inspiratory_angle_values[l]);
                Serial.print("\t");
                Serial.print(expiratory_angle_values[k]);
                Serial.print("\t");
                Serial.print(sumFlow / 100);
                Serial.print("\t");
                Serial.print(sumFlowexpi / 100);
                Serial.print("\t");
                Serial.print(sumPressure / 100);
                Serial.println();
            }
        }
    }

    Serial.println("Fini");
    delay(10000000);
}

#endif
```