UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

Design & Sizing of a Testing Bench for a Turbocompressor

BACHELOR FINAL THESIS

Document:

Appendix

Author:

David Hernandez Gomez

Director/Co-director:

Jordi Ventosa Molina
Àngel Comas Amengual

Degree:

Bachelor in Aerospace Vehicle Engineering

Examination session:

Spring

Contents

Listings

List of Figures

List of Tables

Appendix A

Flowmeter

A.1 Matlab codes

```
clear all
clc
close all

%% DATA INPUT %%

% Fisical data
air_density = 1.225;                                                    % [kg/m^3]
D = 59.25 * 10^-3;                                                      % [m]
visosoty = 1.802 * 10^-5;                                               % [kg/ms]

% diferent constant flow values
% the mass airflow is between 0.03779 & 0.22679 kg/s
m = 5;
q_m = linspace(0.03779, 0.22679, m);                                    % [kg/s]

% different values of beta
n = 100;
beta = linspace(0.15, 0.75, n);

% Pressure accuracy
min_press_increm = 250;                                                 % [Pa]


%% FUNCTION CALL

[increm_P_corner, reynolds_D] = flowmeter_corner_tapp(air_density, D, visosoty, q_m, beta, m, n);
```

Listing A.1: Function call for the calculation of differential pressure values for given values of $\dot{q}_m$ as a function of β.

```
function [increm_P_corner, reynolds_D] = flowmeter_corner_tapp(air_density, D, visosoty, q_m, beta, m, n)

% Tapping data
% As it has corner tapping
L_1 = 0;
```

```
L_2 = 0;


%% Vectros definition
reynolds_D = zeros(1, m);
aux_A = zeros(m, n);
aux_c_1 = zeros(m, n);
aux_c_2 = zeros(m, n);
aux_c_3 = zeros(m, n);
aux_c_4 = zeros(m, n);
aux_c_5 = zeros(m, n);
M_2 = zeros(m, n);
aux_m = zeros(m, n);
coef_c = zeros(m, n);

increm_P_corner = zeros(m, n);
coef_e = zeros(m, n);
coef_e_aux = zeros(m, n);
error_e = zeros(m, n);
P_2 = zeros(m, n);


%% CALCULATIONS

for j = 1:m

    reynolds_D(1, j) = (4 * q_m(1, j)) / (pi * visosoty * D);

    for i = 1:n
        aux_A(j, i) = ((19000 * beta(1, i)) / reynolds_D(1, j))^0.8;

        aux_c_1(j, i) = 0.5961 + 0.0261 * beta(1, i)^2 - 0.216 * beta(1, i)^8;
        aux_c_2(j, i) = 0.000521 * ((10^6*beta(1, i)) / reynolds_D(1, j))^0.7;
        aux_c_3(j, i) = (0.0188 + 0.0063 * aux_A(j, i)) * beta(1, i)^3.5 * (10^6 / reynolds_D(1, j))^0.3;
        aux_c_4(j, i) = (0.043 + 0.08*exp(-10*L_1) - 0.123*exp(-7*L_1)) * (1 - 0.11*aux_A(j, i)) * (beta(1, i)^4 / (1-beta(1, i)^4));
        aux_c_5(j, i) = 0.011 * (0.75 - beta(1, i)) * (2.8 - (D*10^3)/25.4);
        M_2(j, i) = (2 * L_2) / (1 - beta(1, i));
        aux_m(j, i) = -0.031*(M_2(j, i) - 0.8*M_2(j, i)^1.1) * beta(1, i)^1.3;

        coef_c (j, i) = aux_c_1(j, i) + aux_c_2(j, i) + aux_c_3(j, i) +aux_c_4(j, i) + aux_c_5(j, i) + aux_m(j, i);

        coef_e_aux(j, i) = 1;
        error_e(j, i) = 100;
        while(error_e(j, i) > 10^-4)
            increm_P_corner(j, i) = ((q_m(1, j) * sqrt(1-beta(1, i)^4) * 4) / (coef_c(j, i) * coef_e_aux(j, i) * pi * beta(1, i)^2 * D^2))^2 * 1/(2*air_density);
            if increm_P_corner(j, i) >= 101325
                error_e(j, i) = 10^-5;
            else
                P_2(j, i) = 101325 - increm_P_corner(j, i);

                coef_e(j, i) = 1 - (0.351+0.256*beta(1, i)^4+0.93*beta(1, i)^8) * (1 - (P_2(j, i)/101325)^(1/1.4));
                error_e(j, i) = abs(coef_e(j, i) - coef_e_aux(j, i));
                coef_e_aux(j, i) = coef_e(j, i);
            end
```

```
        end
    end

end



%% PLOTS
set(groot,'defaultAxesTickLabelInterpreter','latex');
set(groot,'defaulttextinterpreter','latex');
set(groot,'defaultLegendInterpreter','latex');

%%%%%
fig1 = figure(1);
hold on
set(gca, 'FontSize', 13)
plot(beta, increm_P_corner, LineWidth=2)
set(gca, 'YScale', 'log')
xlabel('$\beta$', 'FontSize', 15)
ylabel('$\Delta{P}$ [Pa]', 'FontSize', 15)
legend('$\dot{q}_m$ = $16.7\%$', '$\dot{q}_m$ = $37.5\%$', '$\dot{q}_m$ = $58.3\%$', '$\dot{q}_m$ = $79
    .1\%$', '$\dot{q}_m$ = $100\%$', 'Location','best');
ylim([0 101325])
grid on
grid minor
box on;
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig1, 'units', 'points');
pos = get(fig1, 'position');
set(fig1, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig1, 'plots/DeltaP for as a function of beta for corner tapp, epsilon', '-dpdf', '-r0', '-bestfit'
    );

%%%%%

fig4 = figure(4);
hold on
set(gca, 'FontSize', 13)
plot(beta, coef_c, LineWidth=2)
xlabel('$\beta$', 'FontSize', 15)
ylabel('Discharge coefficient, $C_d$', 'FontSize', 15)
xlim([0.15 0.75])
legend('$\dot{q}_m$ = $16.7\%$', '$\dot{q}_m$ = $37.5\%$', '$\dot{q}_m$ = $58.3\%$', '$\dot{q}_m$ = $79
    .1\%$', '$\dot{q}_m$ = $100\%$', 'Location','best');
grid on
grid minor
box on;
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig4, 'units', 'points');
pos = get(fig4, 'position');
set(fig4, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig4, 'plots/coef_c as a function of beta', '-dpdf', '-r0', '-bestfit');
```

```
%%%%%
coef_e_1 = coef_e(1, :);
coef_e_2 = coef_e(2, :);
coef_e_3 = coef_e(3, :);
coef_e_4 = coef_e(4, :);
coef_e_5 = coef_e(5, :);
fig5 = figure(5);
hold on
set(gca, 'FontSize', 13)
plot(beta(19:end), coef_e_1(19:end), LineWidth=2)
plot(beta(41:end), coef_e_2(41:end), LineWidth=2)
plot(beta(56:end), coef_e_3(56:end), LineWidth=2)
plot(beta(69:end), coef_e_4(69:end), LineWidth=2)
plot(beta(80:end), coef_e_5(80:end), LineWidth=2)
xlabel('$\beta$', 'FontSize', 15)
ylabel('Expansibility facotr, $\varepsilon$', 'FontSize', 15)
xlim([0.15 0.75])
legend('$\dot{q}_m$ = $16.7\%$', '$\dot{q}_m$ = $37.5\%$', '$\dot{q}_m$ = $58.3\%$', '$\dot{q}_m$ = $79
    .1\%$', '$\dot{q}_m$ = $100\%$', 'Location','best');
grid on
grid minor
box on;
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig5, 'units', 'points');
pos = get(fig5, 'position');
set(fig5, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig5, 'plots/coef_e as a function of beta', '-dpdf', '-r0', '-bestfit');
```

Listing A.2: Computation of differential pressure values for given values of q_m as a function of β.

```
clear all
clc
close all

%% DATA INPUT

air_density = 1.225;                                                    % [kg/m^3]
D = 59.25 * 10^-3;                                                      % [m]
visosoty = 1.802 * 10^-5;                                               % [kg/ms]

% Tappings
L_1 = 0;
L_2 = 0;

% diferent constant flow values
% the mass airflow is between 0.03779 & 0.22679 kg/s
n = 100;
aux_1234 = (0.03779 / 0.22679) * 100;
aux_12354 = (0.22679 / 0.22679) * 100;
q_m_por = linspace(aux_1234, aux_12354, n);                             % [kg/s]
q_m = linspace(0.03779, 0.22679, n);

% Beta defined, only one value
m = 2;
beta = [0.48 0.68];
```

```matlab

% Pressure error
min_press_increm = 250 + 3.815;                                            % [Pa]

%% VECTOR DEFINITION

reynolds_D = zeros(1, n);
aux_A = zeros(m, n);
aux_c_1 = zeros(m, n);
aux_c_2 = zeros(m, n);
aux_c_3 = zeros(m, n);
aux_c_4 = zeros(m, n);
aux_c_5 = zeros(m, n);
M_2 = zeros(m, n);
aux_m = zeros(m, n);
coef_c = zeros(m, n);
increm_P_corner = zeros(m, n);
coef_e_aux = zeros(m, n);
error_e = zeros(m, n);
P_2 = zeros(m, n);
coef_e = zeros(m, n);
uncertainty_c = zeros(m, n);
uncertainty_e = zeros(m, n);
uncertainty_d = zeros(m, n);
uncertainty_p = zeros(m, n);
uncertainty_aux = zeros(m, n);
uncertainty = zeros(m, n);
delta_P = zeros(1, n);

%% CALCULATIONS

for j = 1:n
    reynolds_D(1, j) = (4 * q_m(1, j)) / (pi * visosoty * D);
    for i = 1:m
        aux_A(i, j) = ((19000 * beta(1, i)) / reynolds_D(1, j))^0.8;
        aux_c_1(i, j) = 0.5961 + 0.0261 * beta(1, i)^2 - 0.216 * beta(1, i)^8;
        aux_c_2(i, j) = 0.000521 * ((10^6*beta(1, i)) / reynolds_D(1, j))^0.7;
        aux_c_3(i, j) = (0.0188 + 0.0063 * aux_A(i, j)) * beta(1, i)^3.5 * (10^6 / reynolds_D(1, j))^0.3;
        aux_c_4(i, j) = (0.043 + 0.08*exp(-10*L_1) - 0.123*exp(-7*L_1)) * (1 - 0.11*aux_A(i, j)) * (beta(1, i)^4 / (1-beta(1, i)^4));
        aux_c_5(i, j) = 0.011 * (0.75 - beta(1, i)) * (2.8 - (D*10^3)/25.4);
        M_2(i, j) = (2*L_2) / (1-beta(1, i));
        aux_m(i, j) = -0.031*(M_2(i, j) - 0.8*M_2(i, j)^1.1) * beta(1, i)^1.3;
        coef_c (i, j) = aux_c_1(i, j) + aux_c_2(i, j) + aux_c_3(i, j) +aux_c_4(i, j) + aux_c_5(i, j) + aux_m(i, j);
        coef_e_aux(i, j) = 1;
        error_e(i, j) = 100;
        while(error_e(i, j) > 10^-4)
            increm_P_corner(i, j) = ((q_m(1, j) * sqrt(1-beta(1, i)^4) * 4) / (coef_c(i, j) * coef_e_aux(i, j) * pi * beta(1, i)^2 * D^2))^2 * 1/(2*air_density);
            if increm_P_corner(i, j) >= 101325
                error_e(i, j) = 10^-5;
            else
                P_2(i, j) = 101325 - increm_P_corner(i, j);
                coef_e(i, j) = 1 - (0.351+0.256*beta(1, i)^4+0.93*beta(1, i)^8) * (1 - (P_2(i, j)/101325)^(1/1.4));
                error_e(i, j) = abs(coef_e(i, j) - coef_e_aux(i, j));
                coef_e_aux(i, j) = coef_e(i, j);
```

```
            end
        end

        if increm_P_corner(i, j) < 101325
            % uncertainty coef_c
            if beta(1, i) <= 0.6
                uncertainty_c(i, j) = 0.5 + 0.9*(0.75-beta(1, i))*(2.8-(D*10^3)/25.4);
            else
                uncertainty_c(i, j) = (1.667*beta(1, i)-0.5) + 0.9*(0.75-beta(1, i))*(2.8-(D*10^3)/25.4);
            end
            % uncertainty epsilon
            uncertainty_e(i, j) = 3.5 * (increm_P_corner(i, j)/(1.4*101325));
            % uncertainty diameter
            uncertainty_d(i, j) = ((2*beta(1, i)^4) / (1-beta(1, i)^4))^2 * (0.05/59.25)^2;
            % uncertainty pressure
            uncertainty_p(i, j) = 1/4*(min_press_increm/increm_P_corner(i, j))^2;

            uncertainty_aux(i, j) = sqrt(uncertainty_p(i, j) + uncertainty_d(i, j) + (uncertainty_c(i, j)/100)^2 + (uncertainty_e(i, j)/100)^2);

            uncertainty(i, j) = uncertainty_aux(i, j) * 100;

        else
        end
    end
end
for t = 1:n
    if t <=37
        delta_P(1, t) = increm_P_corner(1, t);
    else
        delta_P(1, t) = increm_P_corner(m, t);
    end
end

%% PLOTS
set(groot,'defaultAxesTickLabelInterpreter','latex');
set(groot,'defaulttextinterpreter','latex');
set(groot,'defaultLegendInterpreter','latex');

fig5 = figure(5);
hold on
set(gca, 'FontSize', 13)
plot(q_m_por, increm_P_corner, LineWidth=2)
xlabel('$\dot{q}_m$  [\%]', 'FontSize', 15)
ylabel('$\Delta P$ [Pa]', 'FontSize', 15)
legend('$\beta$ = 0,48', '$\beta$ = 0,68', 'Location','best');
grid on
grid minor
ylim([0 101325])
box on
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig5, 'units', 'points');
pos = get(fig5, 'position');
set(fig5, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig5, 'plots/DeltaP as a function of q_m 048 068', '-dpdf', '-r0', '-bestfit');
```

```
%%%%%
uncertainty_1 = uncertainty(1, :);
fig7 = figure(7);
hold on
set(gca, 'FontSize', 13)
plot(q_m_por(1:50), uncertainty_1(1:50), LineWidth=2)
plot(q_m_por, uncertainty(2, :), LineWidth=2)
xlabel('$\dot{q}_m$  [\%]', 'FontSize', 15)
ylabel('$\dot q_m$ uncertainty [\%]', 'FontSize', 15)
legend('$\beta$ = 0,48', '$\beta$ = 0,68', 'Location','best');
grid on
grid minor
box on
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig7, 'units', 'points');
pos = get(fig7, 'position');
set(fig7, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig7, 'plots/uncertanity_total', '-dpdf', '-r0', '-bestfit');
%%%%%
fig8 = figure(8);
hold on
set(gca, 'FontSize', 13)
plot(q_m_por, delta_P, LineWidth=2)
xlabel('$\dot{q}_m$  [\%]', 'FontSize', 15)
ylabel('$\Delta P$ [Pa]', 'FontSize', 15)
grid on
grid minor
box on
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig8, 'units', 'points');
pos = get(fig8, 'position');
set(fig8, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig8, 'plots/DeltaP as a function of q with both plates', '-dpdf', '-r0', '-bestfit');
%%%%%
uncertainty_e_1 = uncertainty_e(1, :);
fig10 = figure(10);
hold on
set(gca, 'FontSize', 13)
plot(q_m_por(1:50), uncertainty_e_1(1:50), LineWidth=2)
plot(q_m_por, uncertainty_e(2, :), LineWidth=2)
xlabel('$\dot{q}_m$  [\%]', 'FontSize', 15)
ylabel('$\varepsilon$ uncertainty [\%]', 'FontSize', 15)
legend('$\beta$ = 0,48', '$\beta$ = 0,68', 'Location','best');
grid on
grid minor
box on
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig10, 'units', 'points');
pos = get(fig10, 'position');
set(fig10, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
```

```
    'PaperSize',[pos(3), pos(4)]);
print(fig10, 'plots/uncertainty_e', '-dpdf', '-r0', '-bestfit');


uncertainty_p_1 = uncertainty_p(1, :);
fig11 = figure(11);
hold on
set(gca, 'FontSize', 13)
plot(q_m_por(1:50), 100*sqrt(uncertainty_p_1(1:50)), LineWidth=2)
plot(q_m_por, 100*sqrt(uncertainty_p(2, :)), LineWidth=2)
xlabel('$\dot{q}_m$  [\%]', 'FontSize', 15)
ylabel('$\Delta P$ uncertainty [\%]', 'FontSize', 15)
legend('$\beta$ = 0,48', '$\beta$ = 0,68', 'Location','best');
grid on
grid minor
box on
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig11, 'units', 'points');
pos = get(fig11, 'position');
set(fig11, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig11, 'plots/uncertainty_p', '-dpdf', '-r0', '-bestfit');

rel_error_tot = relative_error_press_flowmeter(q_m_por, n, increm_P_corner, m);
 [P_1, P_2] = pressure_calc(increm_P_corner, beta, q_m, n, m, air_density, D, coef_c);
```

Listing A.3: Computation of differential pressure values for values of β defined as a function of $\dot{q}_m$.

```
function rel_error_tot = relative_error_press_flowmeter(q_m_por, n, increm_P_corner, m)

%% DATA INPUT
accuracy = 0.5;
max_press = 500;
abs_error_ard = (1000 / 13107) * 0.5 * 100;

%% VECTOR DEFINITION
rel_error_sensor = zeros(m, n);
rel_error_ard = zeros(m, n);
rel_error_tot = zeros(m, n);

%% CALCULATIONS

abs_error_sensor = (accuracy / 100) * max_press * 100;
for j =1:m
    for i = 1:n
        rel_error_sensor(j, i) = (abs_error_sensor/increm_P_corner(j, i)) * 100;
        rel_error_ard(j, i) = (abs_error_ard/increm_P_corner(j, i)) * 100;
        rel_error_tot(j, i) = rel_error_ard(j, i) + rel_error_sensor(j, i);
    end
end

%% PLOTS
set(groot,'defaultAxesTickLabelInterpreter','latex');
set(groot,'defaulttextinterpreter','latex');
set(groot,'defaultLegendInterpreter','latex');

```

```
rel_error_sensor_1 = rel_error_sensor(1,:);
fig31 = figure(31);
hold on
set(gca, 'FontSize', 13)
plot(q_m_por(1:50), rel_error_sensor_1(1:50), LineWidth=2)
plot(q_m_por, rel_error_sensor(2, :), LineWidth=2)
xlabel('$\dot{q}_m$  $[\%]$', 'FontSize', 15)
ylabel('$\delta \Delta P$ $[\%]$', 'FontSize', 15)
legend('$\beta$ = 0,48', '$\beta$ = 0,68', 'Location','best');
grid on
grid minor
box on
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig31, 'units', 'points');
pos = get(fig31, 'position');
set(fig31, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig31, 'plots/rel_error_DeltaP_sens', '-dpdf', '-r0', '-bestfit');
%%%%%
rel_error_ard_1 = rel_error_ard(1,:);
fig32 = figure(32);
hold on
set(gca, 'FontSize', 13)
plot(q_m_por(1:50), rel_error_ard_1(1:50), LineWidth=2)
plot(q_m_por, rel_error_ard(2, :), LineWidth=2)
xlabel('$\dot{q}_m$  $[\%]$', 'FontSize', 15)
ylabel('$\delta \Delta P$ $[\%]$', 'FontSize', 15)
legend('$\beta$ = 0,48', '$\beta$ = 0,68', 'Location','best');
grid on
grid minor
box on
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig32, 'units', 'points');
pos = get(fig32, 'position');
set(fig32, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig32, 'plots/rel_error_DeltaP_ard', '-dpdf', '-r0', '-bestfit');
%%%%%
rel_error_tot_1 = rel_error_tot(1,:);
fig33 = figure(33);
hold on
set(gca, 'FontSize', 13)
plot(q_m_por(1:50), rel_error_tot_1(1:50), LineWidth=2)
plot(q_m_por, rel_error_tot(2, :), LineWidth=2)
xlabel('$\dot{q}_m$  $[\%]$', 'FontSize', 15)
ylabel('$\delta \Delta P$ $[\%]$', 'FontSize', 15)
legend('$\beta$ = 0,48', '$\beta$ = 0,68', 'Location','best');
grid on
grid minor
box on
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig33, 'units', 'points');
```

```
pos = get(fig33, 'position');
set(fig33, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig33, 'plots/rel_error_DeltaP_tot', '-dpdf', '-r0', '-bestfit');
```

Listing A.4: Differential pressure measurement error at the orifice plate.

A.2 Arduino data processing codes

```
#include <AMS.h>            //include the AMS library
#include <LiquidCrystal_I2C.h>       // allows to control I2C displays
#include <Wire.h>                    //allows communication over i2c devices

LiquidCrystal_I2C lcd(0x20, 16, 2);          // Location of the display 0x20 - I2C
    address, 16 x 2 display size

float Pressure;
String DataString;
char PrintData[48];
/*define the sensor's instance with the sensor's family, sensor's I2C address as
    well as
its specified minimum and maximum pressure*/
int sensor_mocel = 5915;
int I2C_address = 0x78; // must be defined with a code to search for I2C address
int press_min = -500;
int press_max = 500;
AMS pressure_sensor(int sensor_model, int I2C_address, int press_min, int
    press_max);

void setup() {
  Serial.begin(9600);
  lcd.init();
  lcd.backlight();                   // Initializes display
}

void loop() {
  if (pressure_sensor.Available() == true) {  //check if the pressure sensor
    responds to the given I2C address
    Pressure = pressure_sensor.readPressure();
      if (isnan(Pressure)) { //check if an error occurred leading to the function
    returning a NaN
        Serial.write("Please check the sensor family name.");
      }
        else {
```

```
        DataString = String(Pressure) + " mbar \n"; //put the data into a string
        DataString.toCharArray(PrintData, 48); //convert string into CharArray
              // Printing to serial
              Serial.print("diff. pressure orifice plate: ");
              Serial.write(PrintData); //write the sensor's data on the serial
    port
              Serial.print(" mbar");
              // Printing to display
              lcd.setCursor(0, 0);
              lcd.print("diff.pres: ");
              lcd.print(PrintData);
              lcd.print("mbar");
      }
    }
    else {
        Serial.write("The sensor did not answer.");
  }
```

Listing A.5: Differential pressure data processing at the orifice plate.

Appendix B

Pressure

B.1 Matlab codes

```
function [P_1, P_2] = pressure_calc(increm_P, beta, q_m, n, m, air_density, D, coef_c)

%% DATA INPUT
P_atm = 101325;                                                              % [Pa]
Area_int = pi * (D/2)^2;

aux_1234 = (0.03779 / 0.22679) * 100;
aux_12354 = (0.22679 / 0.22679) * 100;
q_m_por = linspace(aux_1234, aux_12354, n);

%% VECTOR DEFINITION
velocitat_fluid_1 = zeros(m, n);
P_1 = zeros(m, n);
P_1_por = zeros(m, n);
aux_1 = zeros(m, n);
aux_2 = zeros(m, n);
press_loss = zeros(m, n);
P_2 = zeros(m, n);
P_2_por = zeros(m, n);
P_2_sist = zeros(1, n);
P_2_sist_2 = zeros(1, n);
salto_min = zeros(1, n);

%% CALCULATIONS
for i = 1:m
    for j = 1:n
        velocitat_fluid_1(i, j) = q_m(1, j) / (air_density * Area_int);

        P_1(i, j) = P_atm - (velocitat_fluid_1(i, j)^2 * air_density) / 2;
        P_1_por(i, j) = P_1(i, j) / P_atm;
    end
end

for i = 1:m
    for j = 1:n
        aux_1(i, j) = sqrt(1 - beta(1, i)^4*(1-coef_c(i, j)^2));
```

```
        aux_2(i, j) = coef_c(i, j) * beta(1, i)^2;
        press_loss(i, j) = ((aux_1(i, j) - aux_2(i, j)) / (aux_1(i, j) + aux_2(i, j))) * increm_P(i, j);
        P_2(i, j) = P_1(i, j) - press_loss(i, j);
        P_2_por(i, j) = P_2(i, j) / P_atm;

    end
end

for t = 1:n
    if t <=37
        P_2_sist(1, t) = P_2_por(1, t);
    else
        P_2_sist(1, t) = P_2_por(m, t);
    end
end


% minimum pressure jump to avoid recirculation

for k = 1:n
    if k <= 37
        P_2_sist_2(1, k) = P_2(1, k);
    else
        P_2_sist_2(1, k) = P_2(m, k);
    end
end

for i = 1:n
    salto_min(1, i) = P_atm/P_2_sist_2(1, i);
end

%% PLOTS

set(groot,'defaultAxesTickLabelInterpreter','latex');
set(groot,'defaulttextinterpreter','latex');
set(groot,'defaultLegendInterpreter','latex');

P_2_por_1 = P_2_por(1, :);
fig10 = figure(10);
hold on
set(gca, 'FontSize', 13)
plot(q_m_por(1:50), P_2_por_1(1:50), LineWidth=2)
plot(q_m_por, P_2_por(2, :), LineWidth=2)
yline(1)
xlabel('$\dot{q}_m$  $[\%]$', 'FontSize', 15)
ylabel(['Pressure far downstream $[\% P_{atm}]$'], 'FontSize', 15)
legend('$\beta$ = 0,48', '$\beta$ = 0,68', 'Location','best');
grid on
grid minor
box on
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig10, 'units', 'points');
pos = get(fig10, 'position');
set(fig10, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig10, 'plots/P_2 as a function of airlofw', '-dpdf', '-r0', '-bestfit');
```

```
%%%%%
 fig11 = figure(11);
 hold on
 set(gca, 'FontSize', 13)
 plot(q_m_por, P_2_sist, LineWidth=2)
 xlabel('$\dot{q}_m$  $[\%]$', 'FontSize', 15)
 ylabel('Pressure far downstream $[\% P_{atm}]$', 'FontSize', 15)
 grid on
 grid minor
 box on
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig11, 'units', 'points');
pos = get(fig11, 'position');
set(fig11, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig11, 'plots/P_2 as a function of airlofw both plates', '-dpdf', '-r0', '-bestfit');
%%%%%
fig12 = figure(12);
hold on
set(gca, 'FontSize', 13)
plot(q_m_por, salto_min, LineWidth=2)
xlabel('$\dot{q}_m$  $[\%]$', 'FontSize', 15)
ylabel('$\left(\frac{P_2}{P_1}\right)_{min}$', 'FontSize', 15)
grid on
grid minor
box on
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig12, 'units', 'points');
pos = get(fig12, 'position');
set(fig12, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig12, 'plots/salt de presio minim', '-dpdf', '-r0', '-bestfit');
```

Listing B.1: Calculation of the pressure at the compressor's inlet.

```
clear all
clc
close all

%% DATA INPUT
P_atm = 101325;                                                         % [Pa]
% P_2
precision_transducer = 0.25;                                            % [%]
max_press = 6;                                                          % [bar]
precision_arduino = 0.00366748;                                          % [bar]
% P_1
accuracy = 0.5;                                                         % [%]
max_press_p1 = 500;                                                     %[mbar]
abs_error_ard = (1000 / 13107) * 0.5 * 100;                             % [Pa]
% press. ration
m = 3;
n = 100;
P_1 = [0.90 0.85 0.75];                                                 % [bar]
P_2 = linspace(1, 2.75, n);                                             % [bar]
```

```

%% VECTOR DEFINITION
% P_2
abs_press_outlet = linspace(1, max_press + 1, n);
abs_press_outlet_Pa = abs_press_outlet * 100000;
rel_error_transducer = zeros(1, n);
rel_error_arduino = zeros(1, n);
rel_error_total = zeros(1, n);
% P_1
expected_p1 = linspace(70927, 99298, n);
rel_error_p1_sensor = zeros(1, n);
rel_error_p1_ard = zeros(1, n);
rel_error_total_p1 = zeros(1, n);
% press. ratio
press_ratio = zeros(m, n);
rel_error_pr = zeros(m, n);

%% CALCULATIONS
abs_error_transducer = (precision_transducer/100) * max_press;              % [bar]
abs_error_p1 = (accuracy/100) * max_press_p1 * 100;                         % [Pa]

for i = 1:n
    rel_error_transducer(1, i) = (abs_error_transducer / abs_press_outlet(1, i)) * 100;  % [bar/bar]
    rel_error_arduino(1, i) = (precision_arduino/abs_press_outlet(1, i)) * 100;   % [bar/bar]
    rel_error_total(1, i) = rel_error_arduino(1, i) + rel_error_transducer(1, i);

    rel_error_p1_sensor(1, i) = (abs_error_p1/(P_atm-expected_p1(1, i))) * 100;   % [Pa/Pa]
    rel_error_p1_ard(1, i) = (abs_error_ard/(P_atm-expected_p1(1, i))) * 100;   % [Pa/Pa]
    rel_error_total_p1(1, i) = rel_error_p1_ard(1, i) + rel_error_p1_sensor(1, i);

    for j = 1:m
        press_ratio(j, i) = P_2(1, i) / P_1(1, j);
        rel_error_pr(j, i) = ((abs_error_transducer / (P_2(1, i))) * 100) + ((precision_arduino/P_2(1, i)
    ) * 100) + ((abs_error_p1/(P_atm-(P_atm*P_1(1, j)))) * 100) + (abs_error_ard/(P_atm-(P_atm*P_1(1, j)
    ))) * 100;
    end
end

%% PLOTS

set(groot,'defaultAxesTickLabelInterpreter','latex');
set(groot,'defaulttextinterpreter','latex');
set(groot,'defaultLegendInterpreter','latex');

fig13 = figure(13);
hold on
set(gca, 'Fontsize', 14)
plot(abs_press_outlet_Pa, rel_error_transducer, LineWidth=2)
xlabel('$P_2$  $[Pa]$', 'FontSize', 16)
ylabel('$\delta P_{sensor}$ $[\%]$', 'FontSize', 16)
xlim([100000 300000])
grid on
grid minor
box on
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig13, 'units', 'points');
```

```
pos = get(fig13, 'position');
set(fig13, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig13, 'plots/relative error transductor', '-dpdf', '-r0', '-bestfit');
%%%%%
fig14 = figure(14);
hold on
set(gca, 'Fontsize', 15)
plot(abs_press_outlet_Pa, rel_error_arduino, LineWidth=2)
xlabel('$P_2$  $[Pa]$', 'FontSize', 16)
ylabel('$\delta P_{Arduino}$ $[\%]$', 'FontSize', 16)
xlim([100000 300000])
grid on
grid minor
box on
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig14, 'units', 'points');
pos = get(fig14, 'position');
set(fig14, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig14, 'plots/relative error arduino', '-dpdf', '-r0', '-bestfit');
%%%%%
fig15 = figure(15);
hold on
set(gca, 'Fontsize', 15)
plot(abs_press_outlet_Pa, rel_error_total, LineWidth=2)
xlabel('$P_2$  $[Pa]$', 'FontSize', 16)
ylabel('$\delta P$ $[\%]$', 'FontSize', 16)
xlim([100000 300000])
grid on
grid minor
box on
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig15, 'units', 'points');
pos = get(fig15, 'position');
set(fig15, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig15, 'plots/relative error p_2 total', '-dpdf', '-r0', '-bestfit');
%%%%%
fig16 = figure(16);
hold on
set(gca, 'FontSize', 15)
plot(expected_p1/P_atm, rel_error_p1_sensor, LineWidth=2)
xlabel('$P_1$  $[\% P_{atm}]$', 'FontSize', 16)
ylabel('$\delta P_{sensor}$ $[\%]$', 'FontSize', 16)
grid on
grid minor
box on
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig16, 'units', 'points');
pos = get(fig16, 'position');
set(fig16, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
```

```
    'PaperSize',[pos(3), pos(4)]);
print(fig16, 'plots/relative error p_1 sensor', '-dpdf', '-r0', '-bestfit');
%%%%%
fig17 = figure(17);
hold on
set(gca, 'FontSize', 15)
plot(expected_p1/P_atm, rel_error_p1_ard, LineWidth=2)
xlabel('$P_1$  $[\% P_{atm}]$', 'FontSize', 16)
ylabel('$\delta P_{Arduino}$ $[\%]$', 'FontSize', 16)
grid on
grid minor
box on
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig17, 'units', 'points');
pos = get(fig17, 'position');
set(fig17, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig17, 'plots/relative error p_1 ard', '-dpdf', '-r0', '-bestfit');
%%%%%
fig18 = figure(18);
hold on
set(gca, 'FontSize', 15)
plot(expected_p1/P_atm, rel_error_total_p1, LineWidth=2)
xlabel('$P_1$  $[\% P_{atm}]$', 'FontSize', 16)
ylabel('$\delta P_{sensor}$ $[\%]$', 'FontSize', 16)
grid on
grid minor
box on
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig18, 'units', 'points');
pos = get(fig18, 'position');
set(fig18, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig18, 'plots/relative error p_1 total', '-dpdf', '-r0', '-bestfit');
%%%%%
fig19 = figure(1);
hold on
set(gca, 'FontSize', 15)
plot(press_ratio(1,:), rel_error_pr(1,:), LineWidth=2)
plot(press_ratio(2,:), rel_error_pr(2,:), LineWidth=2)
plot(press_ratio(3,:), rel_error_pr(3,:), LineWidth=2)
xlabel('$\Pi$', 'FontSize', 16)
ylabel('$\delta \Pi$ $[\%]$', 'FontSize', 16)
legend('$P_1= 0,9 \% P_{atm}$', '$P_1 =0,85 \% P_{atm}$', '$P_1 =0,75 \% P_{atm}$', 'Location','best');
xlim([1 2.8])
grid on
grid minor
box on
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig19, 'units', 'points');
pos = get(fig19, 'position');
set(fig19, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
```

```
    'PaperSize',[pos(3), pos(4)]);
print(fig19, 'plots/relative error press_ratio', '-dpdf', '-r0', '-bestfit');
```

Listing B.2: Pressure measurement and pressure ratio error.

B.2 Arduino data processing codes

The data processing of the differential pressure between the compressor's inlet pressure and the ambient pressure can be developed the same way as code A.5.

```
#include <LiquidCrystal_I2C.h>  // allows comunications with an I2C display

#include "Wire.h" //allows communication over i2c devices

const int pressureInput = A0; //select the analog input pin for the pressure transducer
const int pressureZero = 205; //analog reading of pressure transducer at 0bar
const int pressureMax = 1023; //analog reading of pressure transducer at 6bar
const int pressuretransducermaxBAR = 6; //psi value of transducer being used
const int baudRate = 9600; //constant integer to set the baud rate for serial monitor
const int sensorreadDelay = 250; //constant integer to set the sensor read delay in milliseconds
const int P_atm = 1;

float pressureValue = 0; //variable to store the value coming from the pressure transducer

LiquidCrystal_I2C lcd(0x20, 16, 2);    // Location of the display 0x20 - I2C address, 16 x 2 display size

void setup() //setup routine, runs once when system turned on or reset
{
  Serial.begin(baudRate); //initializes serial communication at set baud rate bits per second
  lcd.init();
  lcd.backlight();
}

void loop() //loop routine runs over and over again forever
{
  pressureValue = analogRead(pressureInput); //reads value from input pin and assigns to variable
  Serial.print(pressureValue);
```

```
  pressureValue = ((pressureValue-pressureZero)*pressuretransducermaxBAR)/(
    pressureMax-pressureZero) + P_atm; //conversion equation to convert analog
    reading to bar

  // Printing to serial
  Serial.print("press: ");
  Serial.print(pressureValue, 2);
  Serial.println("bar");
  // Printing to display
  lcd.setCursor(0, 0);
  lcd.print("press: ");
  lcd.print(pressureValue, 2);
  lcd.print("bar");
  delay(sensorreadDelay); //delay in milliseconds between read values
}
```

Listing B.3: Compressor's output pressure with transducer data processing.

Appendix C

Tachometer

C.1 Matlab codes

```
clear all
clc
close all

%% DATA INPUT
measure_error = 0.5;                                                   % [us]
min_rpm = 34000;                                                       % [rpm]
max_rpm = 85000;                                                       % [rpm]
max_freq = 3000;                                                       % [Hz}
sensor_acc = 0.01;                                                     % [1%]
n = 100;

%% VECTOR DEFINITION
rpms = linspace(min_rpm, max_rpm, n);
rel_error_rpm = zeros(1, n);
freq = zeros(1, n);
period = zeros(1, n);
rel_error_rpm_sensor = zeros(1, n);
rel_error_rpm_tot = zeros(1, n);

%% CALCULATIONS
T_error = measure_error * 2;
sensor_error = sensor_acc * max_freq;

for i = 1:n
    freq(1, i) =rpms(1, i) / 60;
    period(1, i) = 1 / freq(1, i);
    rel_error_rpm(1, i) = ((T_error*10^-6) / period(1, i)) * 100;
    rel_error_rpm_sensor(1, i) = (sensor_error / freq(1, i)) * 100;
    rel_error_rpm_tot(1, i) = rel_error_rpm_sensor(1, i) + rel_error_rpm(1, i);
end

%% PLOTS
set(groot,'defaultAxesTickLabelInterpreter','latex');
set(groot,'defaulttextinterpreter','latex');
set(groot,'defaultLegendInterpreter','latex');
```

```
fig16 = figure(16);
hold on
set(gca, 'FontSize', 13)
plot(rpms, rel_error_rpm, LineWidth=2)
xlabel('$rpm$', 'FontSize', 15)
ylabel('$\delta rpm$ $[\%]$', 'FontSize', 15)
grid on
grid minor
box on
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig16, 'units', 'points');
pos = get(fig16, 'position');
set(fig16, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig16, 'plots/relative error arduino_rpm', '-dpdf', '-r0', '-bestfit');

fig17 = figure(17);
hold on
set(gca, 'FontSize', 13)
plot(rpms, rel_error_rpm_sensor, LineWidth=2)
xlabel('$rpm$', 'FontSize', 15)
ylabel('$\delta rpm$ $[\%]$', 'FontSize', 15)
grid on
grid minor
box on
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig17, 'units', 'points');
pos = get(fig17, 'position');
set(fig17, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig17, 'plots/relative error sensor_rpm', '-dpdf', '-r0', '-bestfit');

fig18 = figure(18);
hold on
set(gca, 'FontSize', 13)
plot(rpms, rel_error_rpm_tot, LineWidth=2)
xlabel('$rpm$', 'FontSize', 15)
ylabel('$\delta rpm$ $[\%]$', 'FontSize', 15)
grid on
grid minor
box on
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig18, 'units', 'points');
pos = get(fig18, 'position');
set(fig18, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig18, 'plots/relative error tot_rpm', '-dpdf', '-r0', '-bestfit');
```

Listing C.1: Revolutions measurement error.

C.2 Arduino data processing codes

```
#include <LiquidCrystal_I2C.h>

#include "Wire.h" //allows communication over i2c devices

LiquidCrystal_I2C lcd(0x20, 16, 2);      // Location of the display 0x20 - I2C
    address, 16 x 2 display size

unsigned long Htime = 0;       //integer for storing high time

unsigned long Ltime = 0;       //integer for storing low time

double Ttime;          // integer for storing total time of a cycle

double frequency;     //storing frequency
double rpm;

const int baudRate = 9600;
const int sensorreadDelay = 5;

void setup()

{
  pinMode(8,INPUT);     // reads digital value grom input pin 8

  Serial.begin(baudRate);
  lcd.init();
  lcd.backlight();
}

void loop() {
  Htime = pulseIn(8, HIGH);   // measures high time at input pin 8
  Ltime = pulseIn(8, LOW);    // measures low time at input pin 8
  Ttime = (Htime + Ltime);    // total time, period
  frequency = 1000000 / Ttime;
  rpm = frequency * 60;

  // Printing to serial
  Serial.print("rpm: ");
  Serial.print(rpm);
  Serial.print(" rpm");
  // Printing to display
```

```
  lcd.setCursor(0, 0);
  lcd.print(rpm);
  lcd.print(" rpm");
}
```

Listing C.2: Data processing of the revolutions measurement.

Appendix D

Thermometer

D.1 Matlab codes

```
clear all
clc
close all

%% DATA INPUT
T_min = 30;
T_max = 190;
n = 100;
a = -83.151;
b = 685.7;
c = -2078.8;
d = 2210.4;
DeltaV = 0.00244;

%% VECTOR DEFINITION
temp = linspace(T_min, T_max, n);
temp_abs_error = zeros(1, n);
temp_rel_error = zeros(1, n);
V_r = zeros(1, n);
temp_abs_error_ard = zeros(1, n);
temp_rel_error_ard = zeros(1, n);
temp_rel_error_total = zeros(1, n);

%% CALCULATIONS
for i = 1:n
    % sensor
    temp_abs_error(1, i) = 0.3 + 0.005 * temp(1, i);
    temp_rel_error(1, i) = (temp_abs_error(1, i) / temp(1, i)) * 100;
    % arduino
    fun = @(V_r) a*V_r^3 + b*V_r^2 + c*V_r + d - temp(1, i);
    result = fzero(fun, 2);
    V_r(1, i) = result;

    temp_abs_error_ard(1, i) = (3*a*V_r(1, i)^2 + 2*b*V_r(1, i) + c) * DeltaV;
    temp_rel_error_ard(1, i) = (abs(temp_abs_error_ard(1, i)) / temp(1, i)) * 100;

```

```
    temp_rel_error_total(1, i) = temp_rel_error(1, i) + temp_rel_error_ard(1, i);
end

%% PLOTS
set(groot,'defaultAxesTickLabelInterpreter','latex');
set(groot,'defaulttextinterpreter','latex');
set(groot,'defaultLegendInterpreter','latex');

fig20 = figure(20);
hold on
set(gca, 'FontSize', 13)
plot(temp, temp_rel_error, LineWidth=2)
xlabel('T [$^oC$]', 'FontSize', 15)
ylabel('$\delta T$ [$\%$]', 'FontSize', 15)
xlim([30 190])
grid on
grid minor
box on;
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig20, 'units', 'points');
pos = get(fig20, 'position');
set(fig20, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig20, 'plots/relative error temp_sensor', '-dpdf', '-r0', '-bestfit');

fig21 = figure(21);
hold on
set(gca, 'FontSize', 13)
plot(temp, temp_rel_error_ard, LineWidth=2)
xlabel('T [$^oC$]', 'FontSize', 15)
ylabel('$\delta T$ [$\%$]', 'FontSize', 15)
xlim([30 190])
grid on
grid minor
box on;
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig21, 'units', 'points');
pos = get(fig21, 'position');
set(fig21, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig21, 'plots/relative error temp_ard', '-dpdf', '-r0', '-bestfit');

fig22 = figure(22);
hold on
set(gca, 'FontSize', 13)
plot(temp, temp_rel_error_total, LineWidth=2)
xlabel('T [$^oC$]', 'FontSize', 15)
ylabel('$\delta T$ [$\%$]', 'FontSize', 15)
xlim([30 190])
grid on
grid minor
box on;
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;
```

```

set(fig22, 'units', 'points');
pos = get(fig22, 'position');
set(fig22, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig22, 'plots/relative error temp_tot', '-dpdf', '-r0', '-bestfit');
```

Listing D.1: Revolutions measurement error.

D.2 Arduino data processing codes

```
#include <LiquidCrystal_I2C.h>             // allows to control I2C displays
#include <Wire.h>                          //allows communication over i2c devices

LiquidCrystal_I2C lcd(0x20, 16, 2);          // Location of the display 0x20 - I2C address, 16 x 2 display size

// Temperature //
const int tempInput = A1;               // Analog input pin for the temperature
float tempValue;                      // Variable to store the value coming from the temperature
float temperature;                      // Storing temperature value
float x;                          // Value for developing the equations
const float a = -83.151;                // Cubic term of the equation
const float b = 685.7;                  // Squared term of the equation
const float c = -2078.8;                // Lineal term of the equation
const float d = 2210.4;                 // Constant term of the equation
const int baudRate = 9600;                //constant integer to set the baud rate for serial monitor
const int sensorreadDelay = 250;

void setup()
{
  Serial.begin(baudRate);                 // Initializes serial communication
  lcd.init();
  lcd.backlight();                      // Initializes display
}

void loop() {
  tempValue = analogRead(tempInput);         // Reads value from input pin
  x = (tempValue/1023) * 5;                // Conversion equation to convert the analog value to V
  temperature = a*x^3 + b*x^2 + c*x + d;       // Cubic equation of the relation V - C

```

```
    // Printing to serial
    Serial.print("Temperature: ");
    Serial.print(temperature, 1);
    // Printing to display
    lcd.setCursor(0, 0);
    lcd.print("Temp: ");
    lcd.print(temperature, 1);
    lcd.print(" C");
}
```

Listing D.2: Data processing of the temperature at the compressor outlet.

D.3 PT100 table

° C	0	1	2	3	4	5	6	7	8	9
0	100.00	100.39	100.78	101.17	101.56	101.95	102.34	102.73	103.12	103.51
10	103.90	104.29	104.68	105.07	105.46	105.85	106.24	106.63	107.02	107.40
20	107.79	108.18	108.57	108.96	109.35	109.73	110.12	110.51	110.90	111.28
30	111.67	112.06	112.45	112.83	113.22	113.61	113.99	114.38	114.77	115.15
40	115.54	115.93	116.31	116.70	117.08	117.47	117.85	118.24	118.62	119.01
50	119.40	119.78	120.16	120.55	120.93	121.32	121.70	122.09	122.47	122.86
60	123.24	123.62	124.01	124.39	124.77	125.16	125.54	125.92	126.31	126.69
70	127.07	127.45	127.84	128.22	128.60	128.98	129.37	129.75	130.13	130.51
80	130.89	131.27	131.66	132.04	132.42	132.80	133.18	133.56	133.94	134.32
90	134.70	135.08	135.46	135.84	136.22	136.60	136.98	137.36	137.74	138.12
100	138.50	138.88	139.26	139.64	140.02	140.39	140.77	141.15	141.53	141.91
110	142.29	142.66	143.04	143.42	143.80	144.17	144.55	144.93	145.31	145.68
120	146.06	146.44	146.81	147.19	147.57	147.94	148.32	148.70	149.07	149.45
130	149.82	150.20	150.57	150.95	151.33	151.70	152.08	152.45	152.83	153.20
140	153.58	153.95	154.32	154.70	155.07	155.45	155.82	156.19	156.57	156.94
150	157.31	157.69	158.06	158.43	158.81	159.18	159.55	159.93	160.30	160.67
160	161.04	161.42	161.79	162.16	162.53	162.90	163.27	163.65	164.02	164.39
170	164.76	165.13	165.50	165.87	166.24	166.61	166.98	167.35	167.72	168.09
180	168.46	168.83	169.20	169.57	169.94	170.31	170.68	171.05	171.42	171.19
190	172.16	172.53	172.90	173.26	173.63	174.00	174.37	174.74	175.10	175.47
200	175.84	176.21	176.57	176.94	177.31	177.68	178.04	178.41	178.78	179.14
210	179.51	179.88	180.24	180.61	180.97	181.34	181.71	182.07	182.44	182.80
220	183.17	183.53	183.90	184.26	184.63	184.99	185.36	185.72	186.09	186.45
230	186.82	187.18	187.54	187.91	188.27	188.63	189.00	189.36	189.72	190.09
240	190.45	190.81	191.18	191.54	191.90	192.26	192.63	192.99	193.35	193.71
250	194.07	194.44	194.80	195.16	195.52	195.88	196.24	196.60	196.96	197.33

Table D.1: Resistance value of a PT100 as a function of temperature [1].

Appendix E

Efficiency error

E.1 Matlab codes

```
clear all
clc
close all

%% DATA INPUT
T_1 = 25+273;                                                       % [ K ]
P_atm = 101325;                                                     % [Pa]
m = 5;
press_ratio = [1.5 1.7 2 2.3 2.7];
P_1 = 0.85;                                                         % [bar]
abs_error_transducer = 0.0150;
precision_arduino = 0.00366748;
abs_error_p1 = 250;
abs_error_ard = 3.81475547;
T_min = 30;
T_max = 190;
n = 100;
a = -83.151;
b = 685.7;
c = -2078.8;
d = 2210.4;
DeltaV = 0.00244;

%% VECTOR DEFINITION
P_2 = zeros(1, m);
rel_error_pr = zeros(1, m);
abs_error_pr = zeros(1, m);
aux_1 = zeros(m, n);
aux_1_2 = zeros(m, n);
aux_2 = zeros(m, n);
aux_2_2 = zeros(m, n);
eff_abs_error = zeros(m, n);
eff_rel_error = zeros(m, n);
rendimiento = zeros(m, n);

%% CALCULATIONS
```

```
for i = 1:m
    P_2(1, i) = press_ratio(1, i) * P_1;    % bar
    rel_error_pr(1, i) = ((abs_error_transducer / (P_2(1, i))) * 100) + ((precision_arduino/P_2(1, i)) * 100) + (abs_error_p1/(P_atm-(P_atm*P_1)) * 100) + (abs_error_ard/(P_atm-(P_atm*P_1))) * 100;
    abs_error_pr(1, i) = rel_error_pr(1, i)/100 * press_ratio(1, i);
end

%% VECTOR DEFINITION
temp = linspace(T_min, T_max, n);
temp_abs_error = zeros(1, n);
temp_rel_error = zeros(1, n);
V_r = zeros(1, n);
temp_abs_error_ard = zeros(1, n);
temp_rel_error_ard = zeros(1, n);
temp_rel_error_total = zeros(1, n);
temp_abs_error_tot = zeros(1, n);

% Tenoerature error
for i = 1:n
    % sensor
    temp_abs_error(1, i) = 0.3 + 0.005 * temp(1, i);
    temp_rel_error(1, i) = (temp_abs_error(1, i) / temp(1, i)) * 100;

    % arduino
    fun = @(V_r) a*V_r^3 + b*V_r^2 + c*V_r + d - temp(1, i);
    result = fzero(fun, 2);
    V_r(1, i) = result;

    temp_abs_error_ard(1, i) = (3*a*V_r(1, i)^2 + 2*b*V_r(1, i) + c) * DeltaV;
    temp_rel_error_ard(1, i) = (abs(temp_abs_error_ard(1, i)) / temp(1, i)) * 100;


    temp_rel_error_total(1, i) = temp_rel_error(1, i) + temp_rel_error_ard(1, i);
    temp_abs_error_tot(1, i) = temp_rel_error_total(1, i)/100 * temp(1, i);
end

%% EFFICIENCY ERROR
temp = temp + 273;
for j = 1:m
    for i = 1:n
        % parte de press ratio
        aux_1(j, i) = (T_1*0.4*press_ratio(1, j)^(-5/7)) / ((temp(1, i) - T_1)*1.4);
        aux_1_2(j, i) = aux_1(j, i) * abs_error_pr(1, j);
        % parte temp
        aux_2(j, i) = (T_1*(press_ratio(1, j)^(2/7)-1)) / ((temp(1, i) - T_1)^2);
        aux_2_2(j, i) = aux_2(j, i) * temp_abs_error_tot(1, i);
        % total
        eff_abs_error(j, i) = sqrt(aux_1_2(j, i)^2 + aux_2_2(j, i)^2);
        % rendimiento
        rendimiento(j, i) = ((T_1)/((1/press_ratio(1, j))^(2/7)) - T_1) / (temp(1, i) - T_1);
        % relative error
        eff_rel_error(j, i) = (eff_abs_error(j, i) / rendimiento(j, i)) * 100;
    end
end

%% PLOTS
set(groot,'defaultAxesTickLabelInterpreter','latex');
set(groot,'defaulttextinterpreter','latex');
```

```
set(groot,'defaultLegendInterpreter','latex');

fig22 = figure(22);
hold on
set(gca, 'FontSize', 15)
plot(rendimiento(1,:), eff_rel_error(1,:), LineWidth=2)
plot(rendimiento(2,:), eff_rel_error(2,:), LineWidth=2)
plot(rendimiento(3,:), eff_rel_error(3,:), LineWidth=2)
plot(rendimiento(4,:), eff_rel_error(4,:), LineWidth=2)
plot(rendimiento(5,:), eff_rel_error(5,:), LineWidth=2)
xlim([0.55 0.87])
xlabel('Efficiency, $\eta$', 'FontSize', 16)
ylabel('$\delta \eta$ $[\%]$', 'FontSize', 16)
lgd = legend('$\Pi=1,5$', '$\Pi=1, 7$', '$\Pi = 2$', '$\Pi = 2,3$', '$\Pi=2,7$', 'Location','best');
grid on
grid minor
box on
set(gcf, 'units', 'points', 'position', [50,50,676,420]);
hold off;

set(fig22, 'units', 'points');
pos = get(fig22, 'position');
set(fig22, 'PaperPositionMode', 'Auto', 'PaperUnits', 'points', ...
    'PaperSize',[pos(3), pos(4)]);
print(fig22, 'plots/relative error efficiency', '-dpdf', '-r0', '-bestfit');
```

Listing E.1: Efficiency calculation error.

Appendix F

System unification

F.1 Matlab code for obtaining a compressor map point

```
clear all
clc
close all

%% DATA INPUT

air_density = 1.225;
visosoty = 1.802 * 10^-5;
D = 59.25 * 10^-3;
P_atm = 101325;
k = 1.4;

%% VARIABLES
% Orifice plate used
beta = ;
T_o = 298;

% Measurements
DeltaP = ;              % differential pressure at the orifice palte in Pa
P_1 = ;                 % differential pressure at the compressor's inlet in Pa
P_2 = ;                 % differential pressure at the compressor's outlet in bar
temp = ;                % temperature at the compressor's outlet in  C
rpm = ;                 % revolutions measurement in rpm

%% MASS AIRFLOW CALCULATION
[q_m, uncertainty] = mass_airflow_calc(air_density, beta, DeltaP, visosoty, D, P_atm, k)
%% PRESSURE RATIO CALCULATION
[press_ratio, relative_error_press_ratio] = press_ratio_calculation(P_2, P_atm, P_1)
%% EFFICIENCY CALCULATION
[efficiency, relative_error_eff] = efficiency_calculation(temp, T_o, k, press_ratio,
    relative_error_press_ratio)
%% RPM
[rpm, rel_error_rpm_tot] = rpm_calculation(rpm)
```

Listing F.1: Calculations to obtain a compressor map's point for a given measures done.

```
function [q_m, uncertainty] = mass_airflow_calc(air_density, beta, DeltaP, visosoty, D, P_atm, k)
%% DATA
q_m_aux = 0.1;
error = 100;

%% CALCULATION
while error > 10^-5
    reynolds_D = (4*q_m_aux) / (pi*visosoty*D);

    % coef c
    aux_A = ((19000*beta) / reynolds_D)^0.8;
    aux_c_1 = 0.5961 + 0.0261*beta^2 - 0.216*beta^8;
    aux_c_2 = 0.000521*((10^6*beta)/reynolds_D)^0.7;
    aux_c_3 = (0.0188 + 0.0063*aux_A) * beta^3.5*(10^6/reynolds_D)^0.3;
    % aux_c_4 = 0;
    aux_c_5 = 0.011 * (0.75-beta) * (2.8-(D*10^3)/25.4);

    coef_c = aux_c_1 + aux_c_2 + aux_c_3 + aux_c_5;

    % coef e
    P_2 = P_atm - DeltaP;
    coef_e = 1 - (0.351 + 0.256*beta^4 + 0.93*beta^8) * (1 - (P_2/P_atm)^(1/k));


    % mass airflow calculation
    q_m = (coef_c / sqrt(1-beta^4)) * coef_e * pi/4 * beta^2 * D^2 * sqrt(2*DeltaP*air_density);
    error = abs(q_m_aux - q_m);
    q_m_aux = q_m;
end

%% uncertainty
% uncert_d
uncert_d = ((2*beta^4) / (1-beta^4))^2 * (0.05/59.25)^2;
% uncert C_d
if beta < 0.6
    uncert_Cd = 0.5 + 0.9*(0.75-beta)*(2.8-(D*10^3)/25.4);
else
    uncert_Cd = (1.667*beta - 0.5) + 0.9*(0.75-beta)*(2.8-(D*10^3)/25.4);
end
% uncert e
uncert_e = 3.5 * (DeltaP / (k * P_atm));
% uncert_p
min_press_increm = 250 + 3.815;
uncert_p = 1/4*(min_press_increm/DeltaP)^2;

uncertainty_aux = sqrt(uncert_p + uncert_d + (uncert_Cd/100)^2 + (uncert_e/100)^2);
uncertainty = uncertainty_aux * 100;
```

Listing F.2: $\dot{q}_m$ and $\delta\dot{q}_m$ calculations function for a given β and ΔP values.

```
function [press_ratio, relative_error_press_ratio] = press_ratio_calculation(P_2, P_atm, P_1)
%% DATA
abs_error_transducer = (0.25/100) * 6;
precision_arduino = 0.00366748;
abs_error_p1 = (0.5/100) * 500 * 100;
abs_error_ard = (1000 / 13107) * 0.5 * 100;

%% PRESSRE RATIO CALCULATION
```

```
P_2_pa = P_2 * P_atm;
press_ratio = P_2_pa / P_1;

%% RELATIVE ERROR CALULCATION
aux_1 = (abs_error_transducer / P_2) * 100;
aux_2 = (precision_arduino/P_2) * 100;
aux_3 = (abs_error_p1/(P_atm-P_1)) * 100;
aux_4 = (abs_error_ard/(P_atm-P_1)) * 100;
relative_error_press_ratio = aux_4 + aux_3 + aux_2 + aux_1;
```

Listing F.3: Π and $\delta\Pi$ calculations function for a given P_1 and P_2 values.

```
function [efficiency, relative_error_eff] = efficiency_calculation(temp, T_o, k, press_ratio, relative_error_press_ratio)
%% DATA
a = -83.151;
b = 685.7;
c = -2078.8;
d = 2210.4;
DeltaV = 0.00244;

%% EFFICIENCY CALC
T_2_k = temp + 273;
T_s = T_o / ((1/press_ratio)^((k-1)/k));
efficiency = (T_s - T_o) / (T_2_k - T_o);

%% EFFICIENCY RELATIVE ERROR
% temperature error
    % sensor
temp_abs_error = 0.3 + 0.005 * temp;
temp_rel_error = (temp_abs_error / temp) * 100;
    % arduino
fun = @(V_r) a*V_r^3 + b*V_r^2 + c*V_r + d - temp;
result = fzero(fun, 2);
V_r = result;
temp_abs_error_ard = (3*a*V_r^2 + 2*b*V_r + c) * DeltaV;
temp_rel_error_ard = (abs(temp_abs_error_ard) / temp) * 100;

temp_rel_error_total = temp_rel_error + temp_rel_error_ard;
temp_abs_error_tot = temp_rel_error_total/100 * temp;

% pressure ratio absolute error
abs_error_pr = relative_error_press_ratio/100 * press_ratio;

% efficiency error
% press_ratio_part
aux_1 = (T_o*0.4*press_ratio^(-5/7)) / ((T_2_k - T_o)*1.4);
aux_1_2 = aux_1 * abs_error_pr;
% parte temp
aux_2 = (T_o*(press_ratio^(2/7)-1)) / ((T_2_k - T_o)^2);
aux_2_2 = aux_2 * temp_abs_error_tot;
% total
eff_abs_error = sqrt(aux_1_2^2 + aux_2_2^2);
relative_error_eff = (eff_abs_error / efficiency)*100;
```

Listing F.4: η and $\delta\eta$ calculations function for a given Π and T_2 values.

```
function [rpm, rel_error_rpm_tot] = rpm_calculation(rpm)
%% ERROR CALCULATION
measure_error = 0.5;                                                         % [us]
max_freq = 3000;                                                             % [Hz}
sensor_acc = 0.01;
T_error = measure_error * 2;
sensor_error = sensor_acc * max_freq;

frecuency = rpm / 60;
period = 1 / frecuency;

rel_error_rpm = ((T_error*10^-6) / period) * 100;
rel_error_rpm_sensor = (sensor_error / frecuency) * 100;
rel_error_rpm_tot = rel_error_rpm_sensor + rel_error_rpm;
```

Listing F.5: *rpm* and *δrpm* calculations function for a given *rpm* value.

F.2 Arduino data processing codes

```
#include <Wire.h>                    //allows communication over i2c devices
#include <LiquidCrystal_I2C.h>             // allows to control I2C displays
#include <AMS.h>          //include the AMS library

LiquidCrystal_I2C lcd1(0x20, 20, 4);        // Location of the display

// Differential pressure orifice plate //
float Pressure_op;
String DataString_op;
char PrintData_op[48];
/*define the sensor's instance with the sensor's family, sensor's I2C address as
    well as
its specified minimum and maximum pressure*/
int sensor_mocel = 5915;
int I2C_address_op = 0x78; // must be defined with a code to search for I2C
    address
int press_min = -500;
int press_max = 500;
AMS pressure_sensor_op(int sensor_model, int I2C_address_op, int press_min, int
    press_max);

// Differential pressure inlet compressor //
float Pressure_ic;
String DataString_ic;
char PrintData_ic[48];
/*define the sensor's instance with the sensor's family, sensor's I2C address as
    well as
```

```
its specified minimum and maximum pressure*/
int sensor_mocel = 5915;
int I2C_address_ic = 0x78; // must be defined with a code to search for I2C
    address
int press_min = -500;
int press_max = 500;
AMS pressure_sensor_ic(int sensor_model, int I2C_address_ic, int press_min, int
    press_max);

// Pressure transducer //
const int pressureInput = A0;              // Analog input pin for the pressure
    transducer
const int pressureMin = 205;               // Analog reading at 0 bar
const int pressureMax = 1023;              // Analog reading at 6 bar
const int pressuretransducermaxBAR = 6;        // Maximum pressure measurable
const int P_atm = 1;
float pressureValue;                     // Variable to store the value coming from
     the pressure transducer

// Frequency //
const int freqInput = 8;               // Digital input pin fot the frequency
    measure
unsigned long Htime = 0;                   // Integer for storing high time
unsigned long Ltime = 0;                     // Integer for storing low time
float Ttime;                           // Integer for storing total time of a cycle
float frequency;                       // Storing frequency
float rpm;                       // Storing rpm value

// Temperature //
const int tempInput = A1;              // Analog input pin for the temperature
float tempValue;                     // Variable to store the value coming from the
    temperature
float temperature;                     // Storing temperature value
float x;                         // Value for developing the equations
const float a = -83.151;               // Cubic term of the equation
const float b = 685.7;                 // Squared term of the equation
const float c = -2078.8;               // Lineal term of the equation
const float d = 2210.4;                // Constant term of the equation

const int baudRate = 9600;               // Constant integer to set the baud rate
    for serial monitor
```

```
const int sensorreadDelay = 100;          // Constant integer to set the sensor read delay in milliseconds

void setup() {
  pinMode(freqInput,INPUT);              // Pin 8, where the frequency is read, is set up as an input pin

  Serial.begin(baudRate);                // Initializes serial communication
  lcd1.init();                     // Initializes display
  lcd1.backlight();
  lcd2.init();                     // Initializes display
  lcd2.backlight();
}

void loop() {
  // Differential pressure orifice plate //
  if (pressure_sensor_op.Available() == true) {  //check if the pressure sensor responds to the given I2C address
    Pressure_op = pressure_sensor_op.readPressure();
      if (isnan(Pressure_op)) { //check if an error occurred leading to the function returning a NaN
        Serial.write("Please check the sensor family name.");
      }
        else {
        DataString_op = String(Pressure_op) + " mbar \n"; //put the data into a string
        DataString_op.toCharArray(PrintData_op, 48); //convert string into CharArray
              // Printing to serial
              Serial.print("diff. pressure orifice plate: ");
              Serial.write(PrintData_op); //write the sensor's data on the serial port
              Serial.print(" mbar");
              // Printing to display
              lcd.setCursor(0, 0);
              lcd.print("diff.pres: ");
              lcd.print(PrintData_op);
              lcd.print("mbar");
      }
    }
    else {
        Serial.write("The sensor did not answer.");
```

```
  }

// Differential pressure inlet compressor //
  if (pressure_sensor_ic.Available() == true) {  //check if the pressure sensor responds to the given I2C address
    Pressure_ic = pressure_sensor_ic.readPressure();
      if (isnan(Pressure_op)) { //check if an error occurred leading to the function returning a NaN
        Serial.write("Please check the sensor family name.");
      }
        else {
        DataString_ic = String(Pressure_ic) + " mbar \n"; //put the data into a string
        DataString_ic.toCharArray(PrintData_ic, 48); //convert string into CharArray
              // Printing to serial
              Serial.print("diff. pressure inlet compressor: ");
              Serial.write(PrintData_ic); //write the sensor's data on the serial port
              Serial.print(" mbar");
              // Printing to display
              lcd.setCursor(0, 1);
              lcd.print("diff.pres: ");
              lcd.print(PrintData_ic);
              lcd.print("mbar");
      }
    }
    else {
        Serial.write("The sensor did not answer.");
  }

  // Pressure //
  pressureValue = analogRead(pressureInput);    // Reads value from input pin
  pressureValue = ((pressureValue-pressureMin)*pressuretransducermaxBAR)/(pressureMax-pressureMin) + P_atm; // Conversion equation to convert the analog value to bar

  // Frequency //
  Htime = pulseIn(freqInput, HIGH);         // Measure high time of the signal
  Ltime = pulseIn(freqInput, LOW);          // Measure low time of the signal
  Ttime = (Htime + Ltime);                  // The period of the signal can be computed as the sum of high and low time
```

```
  frequency = 1000000 / Ttime;            // The frequency is computed from the period
  rpm = frequency * 60;               // Frequency is converted to rpm

  // Temperature //
  tempValue = analogRead(tempInput);        // Reads value from input pin
  x = (tempValue/1023) * 5;  // Conversion equation to convert the analog value to V
  temperature = a*x^3 + b*x^2 + c*x + d;       // Cubic equation of the relation V - C

  // Printing //
  Serial.print("press: ");
  Serial.print(pressureValue, 2);
  Serial.print("bar    ");
  Serial.print("freq: ");
  Serial.print(frequency, 2);
  Serial.print("Hz    ");
  Serial.print("Temperature: ");
  Serial.print(temperature, 2);
  Serial.print(" C     ");
    Serial.print("rpm: ");
  Serial.print(rpm, 2);
  Serial.print(" rpm");
  Serial.print('\n');

  lcd1.setCursor(0, 2);
  lcd1.print("P:");
  lcd1.print(pressureValue, 2);
  lcd1.print("bar ");
  lcd1.setCursor(9,2);
  lcd1.print("T:");
  lcd1.print(temperature, 1);
  lcd1.print("C");
  lcd1.setCursor(0, 3);
  lcd1.print(rpm);
  lcd1.print(" rpm");
  delay(sensorreadDelay); //delay in milliseconds between read values
}
```

Listing F.6: All data processing required united in a single code.

Appendix G

Orifice plate construction and assembly

The first step in the construction of the system is the construction of the orifice plate and the testing of its functionality. In order to test and assemble the orifice plate configuration, the first step is to get build the sized orifice plates according to the system requirements, those that have been defined along the report. For the construction of the plates itself, to obtain them quickly and economically while complying with the necessary requirements and strength, they have been manufactured using 3D printing following the relevant drawings for each of the plates. This printing has been done by *UPC FabLab Terrassa*, where some 3D printers are available. The results obtained are the shown in figure G.1

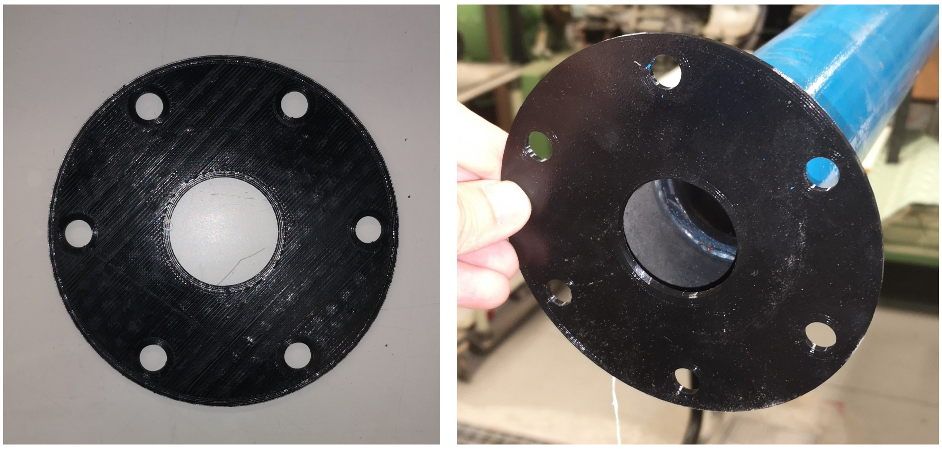

Figure G.1: 3D printed orifice plates

To test the correct functioning of the plates, it has been decided, instead of feeding the turbine of the turbocompressor so that the compressor aspirates air, to feed the orifice plate directly. For this purpose, the outlet of the blower is connected directly to the inlet of the orifice plate pipe. Thus, a certain amount of air can be blown over the orifice plate and the differential pressure generated can be measured, and consequently the mass airflow rate passing through can be calculated. In addition, this measurement of the outgoing mass airflow rate from the blower will serve another purpose. The blower does not indicate the flow rate or velocity of the outgoing air, only its power can be adjusted. Therefore, by directly concentrating the blower output to the orifice plate, not only the correct functioning of the plate itself will be tested, but also the relationship between power and airflow supplied by the blower will be characterised. The assembly made is shown in figures G.2 and G.3.

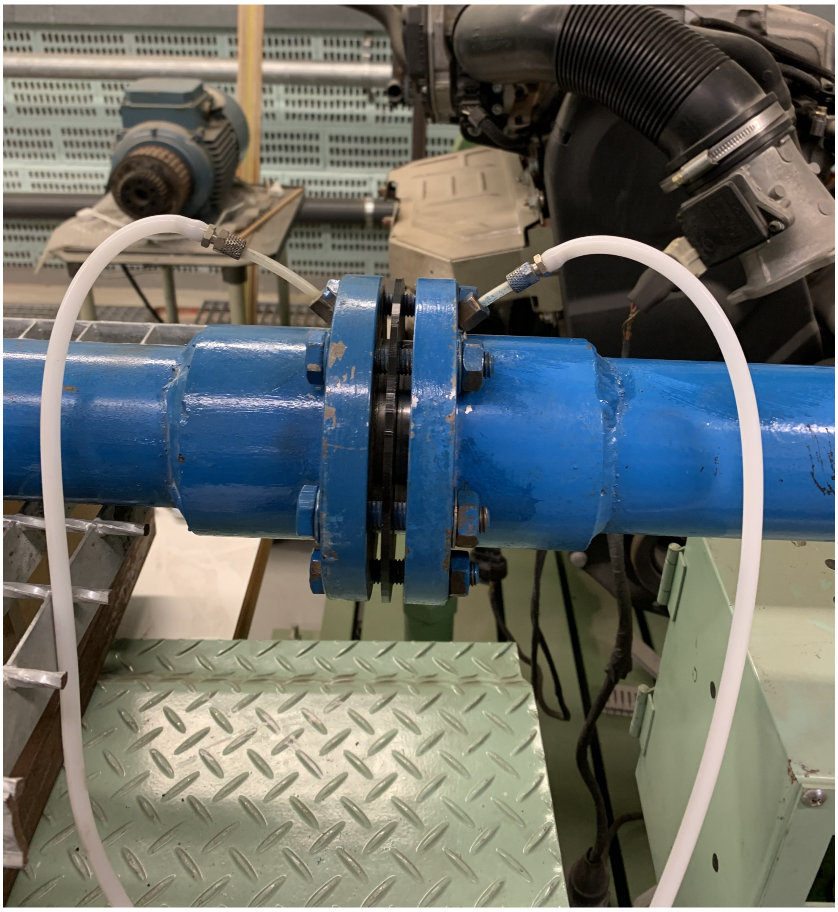

Figure G.2: Orifice plate at the pipe with pressures tappings

Figure G.3: Orifice plate testing assembly

There are two important points to be made regarding the assembly made.

1. To connect the orifice plate tube inlet to the blower outlet directly, it was necessary to increase the support height, as the table was low. Hence the present structure under the tube of the orifice plate. However, safety aspects were taken into account and the tube is properly fixed by means of various fastening elements. In addition to the fact that as this is not the final configuration in which the tube will be in the system, but it is only to do the testing, it has been decided not to make a definitive fastening, since at the final configuration neither the blower is connected to the orifice plate, nor the tube of the plate is in that position, but in the wall.

2. The differential pressure measuring device is different from the one presented in the report. This is due to the fact that the differential pressure measuring device was not available at the time of the test. The measuring instrument is a *Testo 510i*, which is used to measure differential pressures. This element, although available, was not chosen as an option as it does not have sufficient measuring range to measure the full range of differential pressures expected to be generated.

After running tests at different blower power settings, the results obtained are as shown in table G.1 and represented in figures G.4, G.5 and G.6.

Blower power [%]	ΔP $[Pa]$	$\dot{q}_m$ $[kg/s]$
2	5	0,0015
11,5	260	0,0102
20	790	0,0176
30	1840	0,0267
40	3210	0,035
50	4910	0,0431

Table G.1: Results obtained from testing

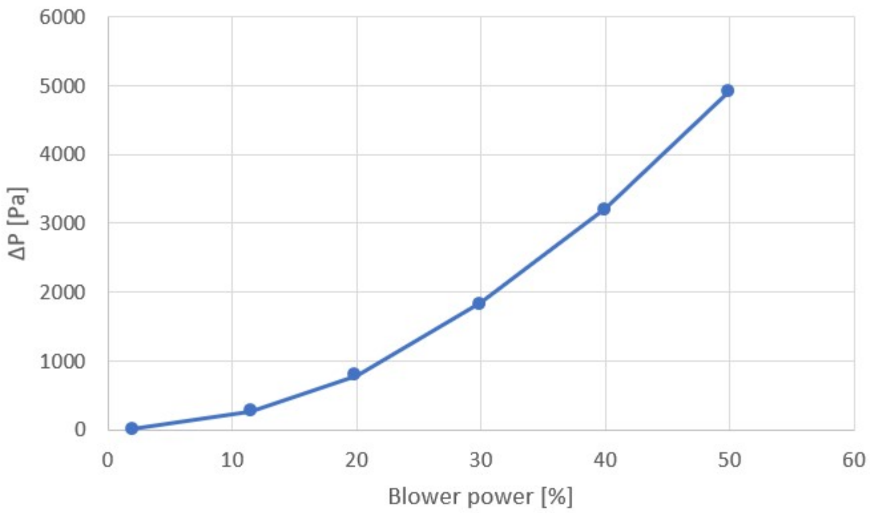

Figure G.4: ΔP as a function of the blower power [%]

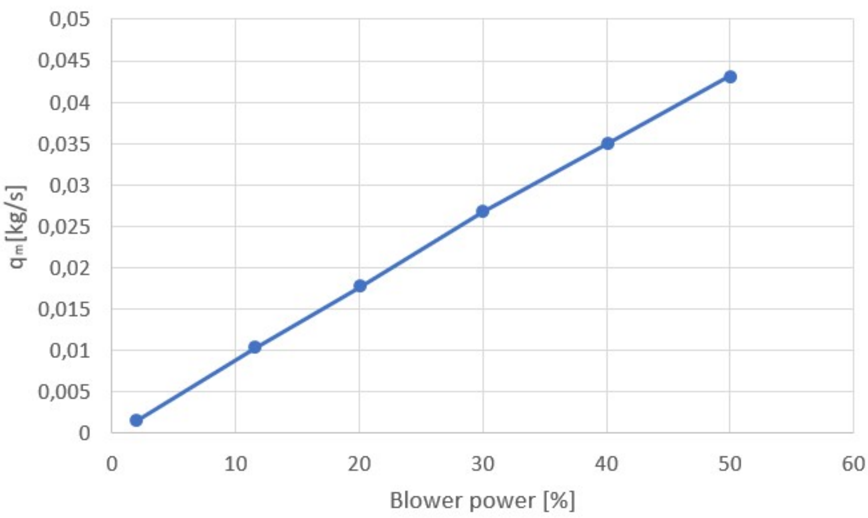

Figure G.5: $\dot{q}_m$ as a function of the blower power [%]

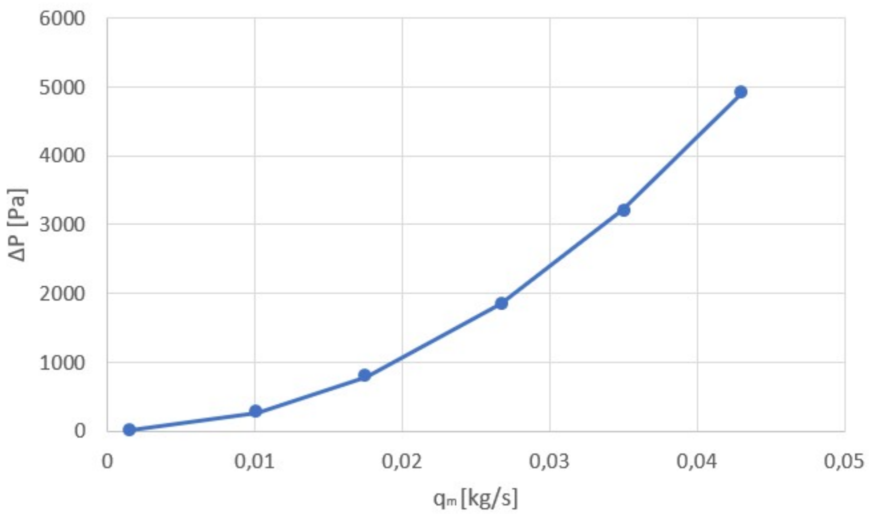

Figure G.6: ΔP as a function of $\dot{q}_m$

Bibliography

1. SENSORS, Guilcor. *VALORES DE RESISTENCIA EN OHMIOS DE 0 ° C A + 400 ° C*. Available also from: https://www.guilcor.es/content/31-table-de-conversion-des-sondes-pt100.