

## Controller representing a cooled air compressor

To illustrate Thermoptim's ability to perform off-design calculations, we will study the behavior of a positive-displacement air compressor that fills a compressed-air storage tank of given volume at variable pressure. The compressed air is cooled before storage by means of a water heat exchanger.

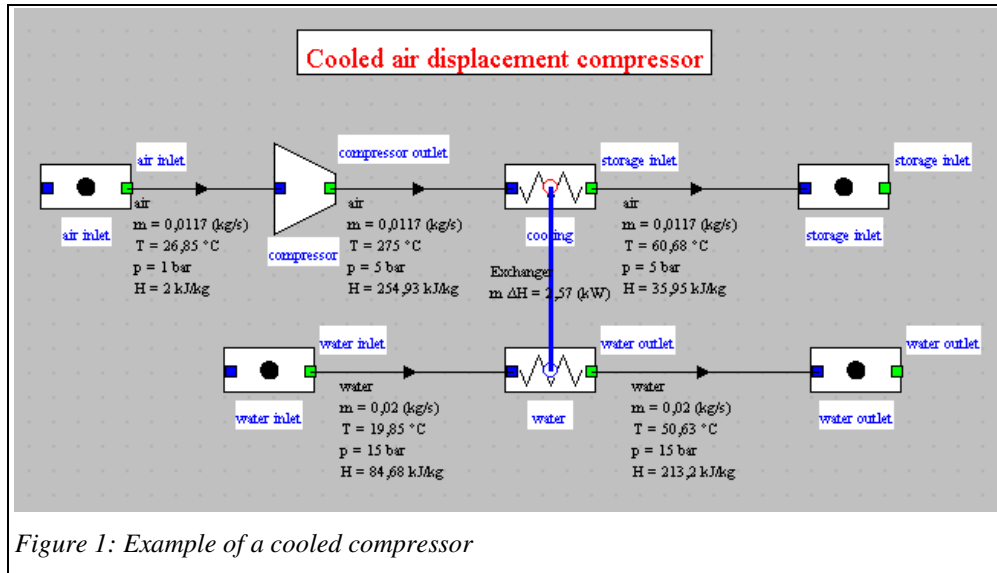


Figure 1: Example of a cooled compressor

The system can easily be modeled in Thermoptim and leads to a synoptic view similar to that in Figure 1. The technological sizing screen of the compressor is shown in Figure 2, and that of the heat exchanger in Figure 3.

**VolumComp**

**compressor**

**Quit**

**Parameters:**

- a0 vol efficiency: 0.93528
- alpha vol. efficiency: 0.04
- K1: 0.80169
- K2: -0.004
- K3: -0.5
- R1: 5
- R2: 0.3
- rotation speed: 1500
- Vs: 0.00055001
- isentropic efficiency: 0.6953071
- flow rate: 0.0117381
- volumetric efficiency: 0.7352800

**Buttons:** Calculate N, Calculate Vs, Calculate m

Figure 2: Compressor sizing screen

Note that a double-click on the component name (here “compressor”, located under the small arrows at the top right) opens its standard screen in the simulator. The configuration of these technology screens is explained in various pages of the Thermoptim-UNIT portal.<sup>1</sup>

<sup>1</sup> <http://www.thermoptim.org/sections/technologies/composants/compresseurs>  
<http://www.thermoptim.org/sections/technologies/composants/echangeurs>  
<http://www.thermoptim.org/sections/base-methodologique/dimensionnement/exemples-dtnn>  
<http://www.thermoptim.org/sections/enseignement/pedagogie/fils-d-ariane/fil-compr-volum>

external component sizing    <    >

Exchanger    Quit

**hlh**  
**hlvh**  
**hh = 2553.96 Re = 229.39**

**hlc**  
**hlvc**  
**hc = 759.07 Re = 1442.18**

e/λ      
Hx design area      
average U   

**cooling**

free flow area	<input type="text" value="0.0027"/>	airCoil_wcc   Wang, Chi & Chang correlation for fin air coil flow outside ...	
hydr. diameter	<input type="text" value="0.0013"/>	<span style="border: 1px solid black; padding: 2px 5px;">correlation settings</span>	
length	<input type="text" value="0.06"/>		
surface factor	<input type="text" value="4"/>	Total press. drop <input type="text" value="0.000698"/>	Sing. ΔP loss coeff. K <input type="text" value="0"/>
fin effectiveness	<input type="text" value="0.8"/>	friction factor <input type="text" value="0.507914"/>	Sing. press. drop <input type="text" value="0"/>

water

free flow area	<input type="text" value="0.000226"/>	int_tube   Mac Adams correlation for single phase flow inside tubes	
hydr. diameter	<input type="text" value="0.012"/>	<span style="border: 1px solid black; padding: 2px 5px;">correlation settings</span>	
length	<input type="text" value="0.9"/>		
surface factor	<input type="text" value="1"/>	Total press. drop <input type="text" value="0.000131"/>	Sing. ΔP loss coeff. K <input type="text" value="0"/>
fin effectiveness	<input type="text" value="1"/>	friction factor <input type="text" value="0.044378"/>	Sing. press. drop <input type="text" value="0"/>

*Figure 3: Heat exchanger sizing screen*

### 1.1 Results obtained

Once the controller has been implemented, it is very easy to vary the compression ratio in order to obtain the evolution of the main quantities as the storage pressure changes. These results can be processed using the Excel macro for post-processing Thermoptim simulation files presented in Volume 1 of the reference manual. The only requirement is to save the project file under a different name after each simulation, then load these files into the macro and extract the relevant values.

Figure 4 shows, as a function of the storage pressure, the evolution of the compressor isentropic efficiency and power consumption, the inlet air flow rate, the temperature of the air entering the tank, the heat duty of the exchanger and the value of U.

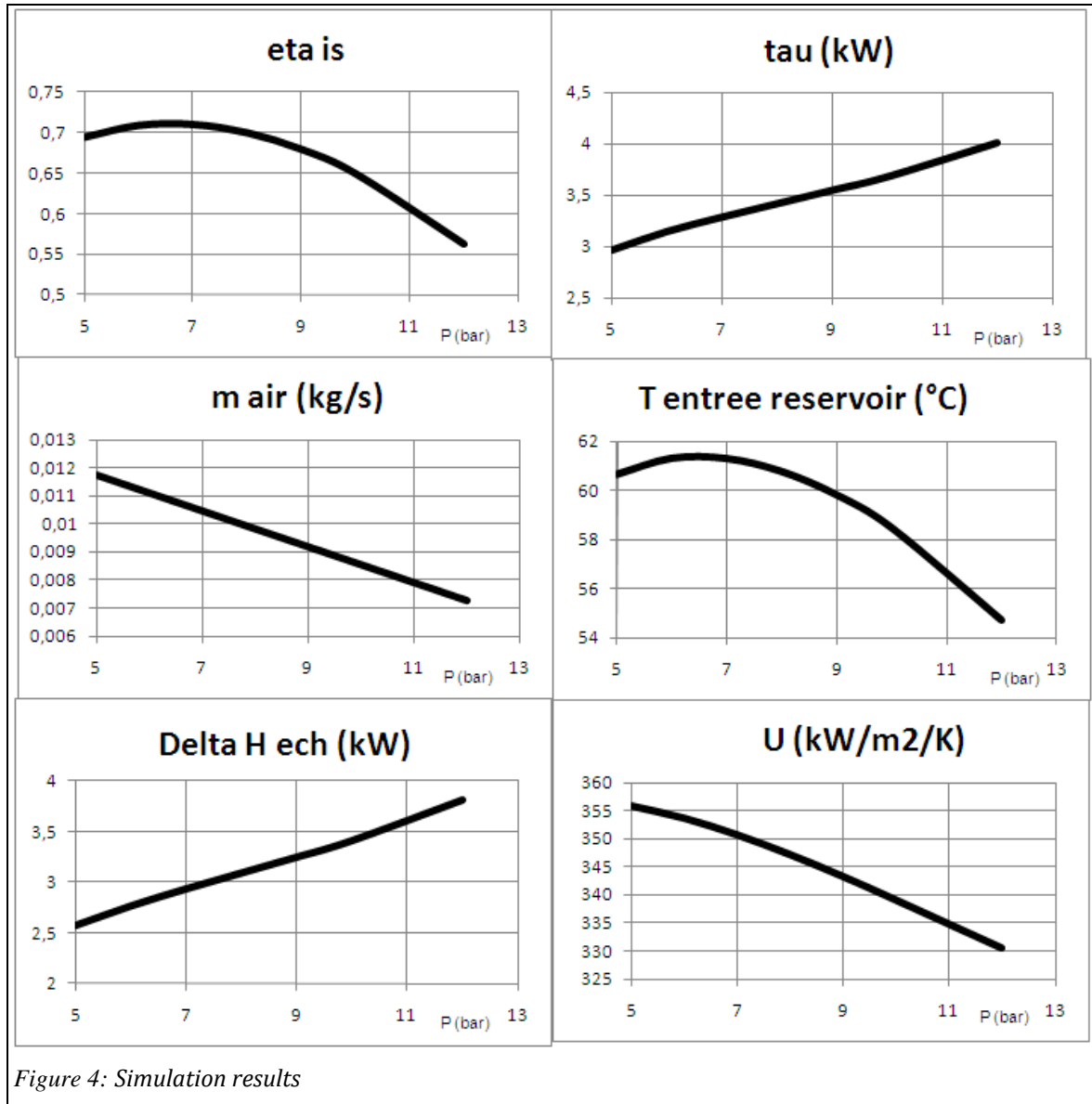


Figure 4: Simulation results

## 1.2 Design of the controller

The controller class is called `PiloteCompresseurVs`. Its graphical user interface is simple and conventional, so it will not be described in detail here. The controller screen is shown in Figure 5..

To implement the corresponding class, proceed as follows:

- First instantiate the technology screens of the compressor and the heat exchanger.
- Initialize them.

Figure 5 shows the 'Design settings' window of the controller. It includes an 'Initial settings' button and a 'Calculate' button. The parameters are organized into two columns:

Parameter	Value	Parameter	Value
heat exchanger UA	0.0216	swept volume	0.00055002
set exchanger area	0.0408	air storage pressure	8.0
calculated exchanger area	0.0408		
heat exchanger U	527.8182	volumetric efficiency	0.6153
flow rate	0.0099	isentropic efficiency	0.7008

Figure 5: Controller screen

- Define the calculations and the modifications to the simulator settings that must be performed when the downstream pressure is varied.:

### 1.2.1 Instantiations

```
//initializations for the simulation
//watch out: the names of points and components must be correct, other
hxName="Exchanger";
compressorName="compressor";
hotDownstream=new PointThopt(proj,"storage inlet");
hotUpstream=new PointThopt(proj,"compressor outlet");
comprUpstream=new PointThopt(proj,"air inlet");
coldUpstream=new PointThopt(proj,"water inlet");
coldDownstream=new PointThopt(proj,"water outlet");

//instantiation of the TechnoDesign in the external classes
technoExchanger=new TechnoHx(proj, hxName, hotUpstream, hotDownstream,
addTechnoVector(technoExchanger);
technoCompr=new VolumCompr(proj, compressorName, comprUpstream, hotUps
addTechnoVector(technoCompr);

//initialization of the TechnoDesign in Thermoptim
setupTechnoDesigns(vTechno);
```

### 1.2.2 Initializations

During initialization, the heat-exchanger technology screen is updated from the simulator values, and its area is calculated.

```
if(!hxName.equals("")){//initialisation de l'évaporateur
    args=new String[2];
    args[0]="heatEx";
    args[1]=hxName;
    vProp=proj.getProperties(args);
    Double f=(Double)vProp.elementAt(15);
    UAech=f.doubleValue();
    String fluideChaud=(String)vProp.elementAt(0);
    args[0]="process";
    args[1]=fluideChaud;
    vProp=proj.getProperties(args);
    f=(Double)vProp.elementAt(4);
    DeltaH=-f.doubleValue();

    DeltaHech=DeltaH;
    hotUpstream.getProperties();
    hotDownstream.getProperties();
    coldUpstream.getProperties();
    coldDownstream.getProperties();

    mCpEau=DeltaH/(coldDownstream.T-coldUpstream.T);
    mCpAir=DeltaH/(hotUpstream.T-hotDownstream.T);
    UAech_value.setText(Util.aff_d(UAech,4));

//initialisations du TechnoDesign
    technoExchanger.UA=UAech;
    technoExchanger.makeDesign();
    AechReel=Util.lit_d(technoExchanger.ADesign_value.getText());
    U_ech=UAech/AechReel;
    AcalculatedEch_value.setText(Util.aff_d(AechReel,4));
    AdesignEch_value.setText(technoExchanger.ADesign_value.getText());
```

The compressor displacement  $V_s$  required to obtain the desired flow rate is then determined based on the configuration of the technology screen.:

```
if(!compressorName.equals("")){//initialization of the compressor
    args=new String[2];
    args[0]="process";
    args[1]=compressorName;
    vProp=proj.getProperties(args);
    String amont=(String)vProp.elementAt(1);
    String aval=(String)vProp.elementAt(2);
    Double f=(Double)vProp.elementAt(3);
    massFlow=f.doubleValue();
    N_value=Util.lit_d(technoCompr.N_value.getText());
    lambdaVol=technoCompr.getLambdaVol();
    Vs=massFlow*60*comprUpstream.V/N_value/lambdaVol;
    Vs_value.setText(Util.aff_d(Vs,8));
    technoCompr.setVs(Vs);
    technoCompr.setN(N_value);
}
```

### 1.2.3 Calculations

The calculations to be performed are as follows:

- Start by initializing the displacement and updating the compressor outlet pressure.
- Compute the volumetric and isentropic efficiencies, determine the mass flow rate involved and propagate it upstream and downstream.
- Recalculate the compressor and the downstream transformation, knowing that the latter, defined as isobaric, propagates the new pressure.
- Update the inlet and outlet states of the heat exchanger, then recalculate it in off-design mode after updating the air heat load, which has been modified.
- Update the simulator and recompute the project several times to ensure convergence of the values.

The first five steps correspond to the code given below.

:

```
void bCalc_actionPerformed(java.awt.event.ActionEvent event)
{
    Vs=Util.lit_d(Vs_value.getText());//reads the compressor swept volume
    technoCompr.setVs(Vs);
    Preservoir=Util.lit_d(P_value.getText());
    double UA_ech=Util.lit_d(UAech_value.getText());

    hotUpstream.P=Preservoir;// updates the compressor outlet pressure
    hotUpstream.update(!UPDATE_T,UPDATE_P,!UPDATE_X);
    hotUpstream.getProperties();

    massFlow=technoCompr.getMassFlow(comprUpstream.V);
    double eta_is=technoCompr.getRisentr();// calculates compressor isentr
    lambdaVol=technoCompr.getLambdaVol();

    //recalculates the compressor and the downstream process
    updateprocess(compressorName, "Compression",RECALCULATE,IS_SET_FLOW, t
    hotUpstream.getProperties();
    updateprocess("refroidissement", "Exchange",RECALCULATE,IS_SET_FLOW, t
    updateprocess("entree air", "Exchange",RECALCULATE,IS_SET_FLOW, UPDATE

    hotUpstream.getProperties();//updates heat exchanger inlets and outlet
    hotDownstream.getProperties();
    coldUpstream.getProperties();
    coldDownstream.getProperties();
}
```

This example highlights the advantage of using PointThopt objects to exchange data between the controller and the simulator: the syntax of the updates is much more readable than when using only the `getProperties()` and `updatePoint()` methods of `Projet`.

The compressor calculation uses the two methods `getMassFlow()` and `getLambdaVol()` of the technology screen. The `updateProcess()` method modifies the flow rate and the compressor isentropic efficiency, then recalculates the compressor. Its downstream point, called here `amontChaud` because it corresponds to the hot-fluid inlet of the exchanger, is then updated.

The heat-exchanger calculation is performed as follows: a first calculation is made with the `updateHx()` method by imposing the value of `UA` read on the screen, `UA_ech` (the exchanger is configured for an off-design calculation, so that the specified `UA` value is taken into account). This calculation initializes the exchanger for the new operating conditions.

The `makeDesign()` method of `TechnoDesign` is then called, which updates the value of `U`. The actual `UA` is obtained by multiplying the new `U` by the initial value of `A` (`AechReel`), which makes it possible to recompute the exchanger correctly.

Since `U` depends on the mean fluid temperatures and therefore on the outlet temperatures, the calculations are iterated five times to ensure good convergence, resetting each time `UA_ech`.

```
//calculates the heat exchanger (several iterations)
for(int i=0;i<5;i++){
    updateHx(hxName, RECALCULATE, UPDATE_UA, UA_ech, !UPDATE_EPSI, 0,
    coldDownstream.getProperties();
    hotDownstream.getProperties();

    technoExchanger.UA=UA_ech;//calculates U
    technoExchanger.makeDesign();
    double U=technoExchanger.getU();

    UAech=U*AechReel/1000.;//updates UA and recalculates the heat exch
    UAech_value.setText(Util.aff_d(UAech,4));
    updateHx(hxName, RECALCULATE, UPDATE_UA, UAech, !UPDATE_EPSI, 0, !

    coldDownstream.getProperties();
    hotDownstream.getProperties();
    U_value.setText(Util.aff_d(U,4));
    UA_ech=UAech;
}

//updates the simulator and displays
for(int j=0;j<3;j++){proj.calcThopt();
    flow_value.setText(Util.aff_d(massFlow,4));
    eta_is_value.setText(Util.aff_d(eta_is,4));
    lambdaVol_value.setText(Util.aff_d(lambdaVol,4));
    AcalculatedEch_value.setText(technoExchanger.ADesign_value.getText());
}
```