Optimal One-gas Model Furnace

January 13, 2021

Contents

1 Introduction

A problem that most players face in Stationeers is automating a furnace for smelting alloys, where the issue is reaching a specific pressure target and temperature target. With a naïve setup, we combust fuel to raise the pressure and temperature, cool the system if need be, then remove pressure until both are in range. A better, more precise setup is demanded for more difficult alloys.

2 Motivation

The immediate question is: given the initial conditions of a furnace, can I add a specifically formated mixture of gas in order to reach those targets without guess-work? The answer is yes: though we may need to remove some gas first, there is such a perfect mixture. However, embedded within this problem of temperature and pressure is an issue of specific heat. A volume M comprised of a mix of the seven gasses in the game may have a particular state (parameters pressure, temperature and moles per gas), but there are many different gas compositions such that the specific heat s_M will satisfy these conditions. In-fact, all such possible compositions lie within an

$\mathbb{R}^7$ subspace, whose support is on what gasses exist in the volume.

$$s_M = \vec{n_M} \cdot \vec{c} = (n_M(g_0), \cdots, n_M(g_6)) \cdot (c_{g_0}, \cdots, c_{g_6})$$
$$\vec{n_{M_g}} = n_M(g). \qquad \text{(The } g^{\text{th}} \text{ component)}$$

The dimension of this subspace depends on the number of gasses in the system, thus intentionally restricting the gasses (as per filtering) allows us to reduce the dimension as we see fit. This, along with the practical purposes of only using one gas (namely, removing gases which have useful applications outside pressurizing a volume) is why I've decided to implement a one-gas model furnace. In such a system, the dimension of the specific heat subspace is reduced to a single dimension, and calculation of the energy from combining two compositions is made significantly simpler. If we fix g^* the sole gas of the system, then the specific heat s_M of a volume M becomes:

$$n_M = \sum_{g \in G} n_M(g) = n_M(g^*), \quad \vec{n_{M_g}} = \begin{cases} n_M, & \text{if } g = g^* \\ 0, & \text{if } g \neq g^* \end{cases} \Rightarrow s_M = \vec{n_M} \cdot \vec{c} = n_M c_{g^*}.$$

Furthermore, equation defining the result of combining two volumes with different temperatures becomes significantly simpler, where specific heat is replaced simply by moles:

$$t_C s_C = t_A s_A + t_B t_B \;\Rightarrow\; t_C n_C c_{g^*} = t_A n_A c_{g^*} + t_B n_B c_{g^*} \;\Rightarrow\; t_C n_C = t_A n_A + t_B n_B.$$

3 Implementation and optimization

Now we move into the specifics of implementing such a system in the game, involving furnaces and pipe networks. Considering a system of only one gas with a hot source H of temperature t_H and a cold source C of temperature t_C, we can calculate an optimal formulation for bringing a furnace F (with volume v_F, initial pressure p_F, initial temperature t_F, and initial moles $n_F = \frac{p_F v_F}{R t_T}$, where R is the ideal gas constant) to a desired pressure p_T and temperature t_T. This is accomplished by removing an amount n_R from the furnace and/or adding an amount n_I at a specific temperature t_I, where I is composed from amounts n_H and n_C from the H and C sources. Several gas-law derived equations constrain this process:

$$t_T n_T = t_F(n_F - n_R) + t_I n_I \tag{1}$$
$$n_T = n_F - n_R + n_I \tag{2}$$
$$t_I n_I = t_H n_H + t_C n_I \tag{3}$$

Where n_R, t_I and $n_I = n_H + n_C$ are to be determined. Note that each term in this system obeys some constraints:

$$t_C < t_M < t_H, \quad 0 \leq n_M, \quad 0 \leq n_R \leq n_F.$$

Solving Equation 1 for n_I provides a surface bounded in two dimensions by $0 \leq n_R \leq n_F$ and $t_C \leq t_I \leq t_H$, but where $0 \leq n_I$ is potentially unbounded.

$$(n_R, t_I, f): \quad f(n_R, t_I) = n_I = \frac{t_T n_T + t_F(n_R - n_F)}{t_I}.$$

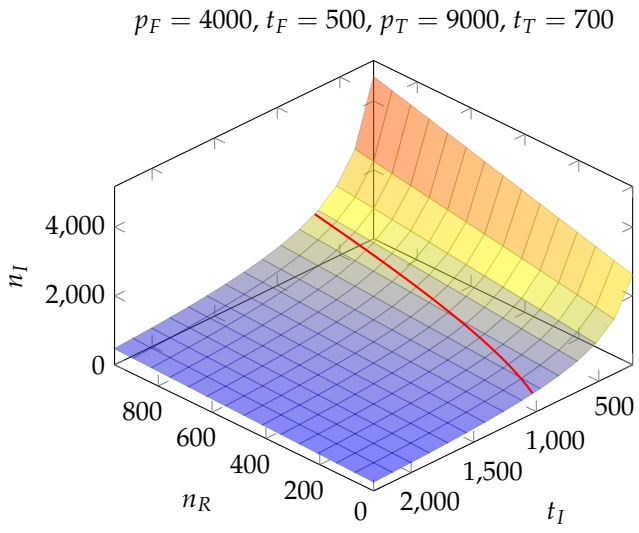


Figure 1: The (n_R, t_I, f) solution surface and (n_R, h, g) embedded curve.

Equation 2 further restricts potential solutions, but given a satisfactory amount to remove n_R provides a particular solution for the amount to add: $n_I = n_T - n_F + n_R$. Then we instead solve Equation 1 for t_I, and as a result these restricted solutions lie within a curve embedded within the surface.

$$(n_R, h, g): \quad g(n_R) = n_I = n_T - n_F + n_R, \quad h(n_R) = t_I = \frac{t_T n_T - t_F(n_F - n_R)}{n_T - n_F + n_R}.$$

With respect to t_I, h is monotone decreasing, but monotone increasing with respect to n_R. Thus minimizing with respect to n_I in the domain is a matter of minimizing n_R and maximizing t_I. However, $0 \leq n_R$ is naïve, and Equation 3 imparts a narrower lower-bound.

$$t_C n_I \leq t_I n_I \qquad t_I n_I \leq t_H n_I$$
$$n_F(n_F - t_C) - n_T(t_T - t_C) \leq n_R(t_F - t_C) \qquad t_T n_T - t_F(n_F - n_R) \leq t_H(n_T - n_F + n_R)$$
$$n_{R_C} = n_F - \frac{n_T(t_T - t_C)}{t_F - t_C} \leq n_R \quad (4) \qquad n_{R_H} = n_F - \frac{n_T(t_H - t_T)}{t_H - t_F} \leq n_R \quad (5)$$

Thus $\max(n_{R_C}, n_{R_H}) \leq n_R$. In-fact, the curve crosses a hyperplane boundary at a point (n_R, t_I, n_I) which minimizes n_I and for which $n_R = \max(0, n_{R_C}, n_{R_H})$. We then easily solve for the remaining coordinates.

In the end, finding the n_I minimizing point (n_R, t_I, n_I):

$$n_R = \max(0, n_{R_C}, n_{R_H}) \tag{6}$$

$$t_I = \frac{t_T n_T - t_F(n_F - n_R)}{n_T - n_F + n_R} \tag{7}$$

$$n_I = n_T - n_F + n_R. \tag{8}$$

Which hyperplane the curve crosses corresponds to a certain procedure of the gas mixer that in the end we are making: the curve either:

- Crosses $n_R = 0$: a mixture of H and C is added, or
- Crosses $t_I = t_H$: some volume of F is removed and only H is added, or
- Crosses $t_I = t_C$: some volume of F volume is removed and only C is added.

Lastly, since n_I is composed of moles from H and C, we also need to calculate n_H and n_C, which is trivial given that $n_I = n_H + n_C$ and $t_I n_I = t_H n_H + t_C n_C$.

$$\begin{aligned} t_I &= n_H + n_C \\ \Rightarrow n_C &= n_I - n_H. \end{aligned} \tag{9}$$

$$\begin{aligned} t_I n_I &= t_H n_H + t_C n_C \\ &= t_H n_H + t_C(n_I - n_H) \\ &= t_C n_I + n_H(t_H - t_C) \\ \Rightarrow n_H &= \frac{t_I n_I - t_C n_I}{t_H - t_C}. \end{aligned} \tag{10}$$

And what we are left with is precisely the minimum number of moles n_R to remove from the furnace initially, and (with respect to n_R) n_H and n_C the precise numbers of moles to add from the H and C sources respectively, such that F will achieve pressure p_T and temperature t_T exactly. All that is left is game implementation.

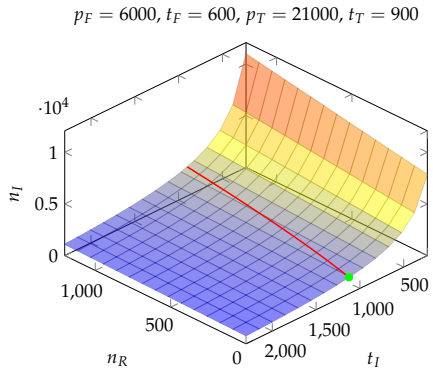
4 Game implementation with MIPS

I've decided to implement this entire mixing program along with controls onto a single IC housing, and implementation details will depend on what devices we specifically need.

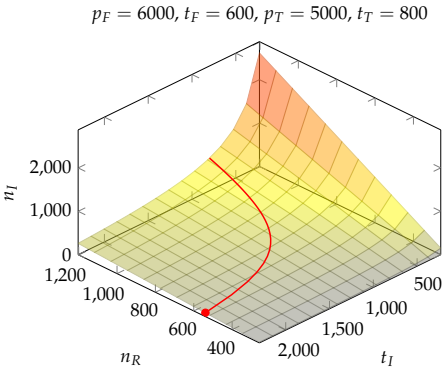
Note that for or clarity in describing the steps of the MIPS program, many uniquely labeled term of an equation will have a unique register alias in the code described below, but optimizations will need to be made in reusing registers in order to stay within the 16 register limit of the final MIPS program.

I've chosen to use the following device inputs: the furnace itself, for reading its state, three pipe analyzers; one each for H, C and I, one volume pump for removing the n_R initial furnace moles,

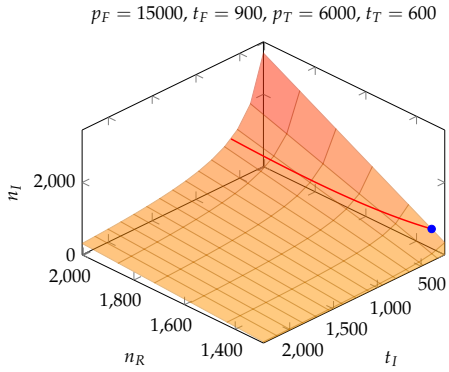
```
alias HAnalyzer d0
alias HPump d1
alias CAnalyzer d2
alias CPump d3
```



(a) H and C mixture added.



(b) Volume removed, H added.



(c) Volume removed, C added.

Figure 2: Example solutions.

and one gas mixer for composing I.

```
alias IAnalyzer d4
alias IPump d5

alias x <num> # for calulcations
alias y <num>
alias z <num>
alias v <num> # for volume in calculations
alias i <num> # for device indirection
```

4.1 Constants and utility functions

Define the ideal gas constant and volumes.

```
define R 8.31446261815324
define vF  1000
define vHC 500  #  5*100L per pipe
define vI  1500 # 15*100L per pipe
```

For emptying the input pipe I into the furnace

```
fillFurnace:
yield
s Furnace SettingInput 100
l x IAnalyzer TotalMoles
brgtz -3
s Furnace SettingInput 0
j ra
```

And for emptying the furnace into the filtration system, which filters all CO_2 back into the input pipe.

```
emptyFurnace:
yield
s Furnace SettingOutput 100
l x Furnace TotalMoles
brgtz -3
s Furnace SettingOutput 0
l x IAnalyzer TotalMoles
yield
l y IAnalyzer TotalMoles
beq x y ra # wait for CO2 filter
move x y
jr -4
```

4.2 Acquiring p_T and t_T

Since all six of our device pins are being used already, we have no way to directly set the target pressure and temperature for our program, or even any way to directly start it. Instead we can use a messaging system, where another program (a control program) sends messages which set these values or start the program. The `db` register (that is, the value corresponding to the IC housing)

acts as a channel through which a "sender" can communicate to this one, the "receiver", and vice versa.

db value	Description
0	Receiver (this program) is ready to receive a message
-1	Receiver awaiting a follow-up value
1	Target pressure message
2	Target temperature message
3	Start message

First we define the messages.

```
define MsgReadyMsg 0
define MsgReadyValue -1
define MsgPTarget 1
define MsgTTarget 2
define MsgStart 3
```

The receiver program contains in its main loop idle state a series of branches based on the kind of message it has received. In the case that db has been set to 1/2 then the receiver understand that a sender is going to send a value corresponding to target pressure/temperature. The receiver sets itself to -1 to announce to any and all senders that were *not* to program that sent the message to not send anything, and to announce to the actual sender that this receiver is ready to accept whatever the actual value is. The actual sender waits to see -1, before overwriting it with the actual value. Lastly the receiver sees the value, moves it into the appropriate register, and sets itself to 0 to announce that it is once again ready to accept messages. In the case that 3 was sent, then the receiver instead "starts", and moves on to the actual gas mixing program.

The receiver message component. In addition to receiving p_T and t_T, we also need to recalculate the target moles n_T.

```
checkInput:
l x db Setting
brne x MsgPTarget 4 # receive pT
jal waitReceive
move pT x
jal calculateTargetMoles
brne x MsgTTarget 4 # receive tT
jal waitReceive
move tT x
jal calculateTargetMoles
bne x MsgStart main
# ...

waitReceive:
s db Setting MsgReadyValue
yield
```

Example sender message component

```
checkInput:
l x GasControlIC Setting
bne x MsgReadyMsg ra # skip if rx busy
l x PTargetDial Setting # pTarget
breq x pTarget 6
move pTarget x
mul x x 100
s PTargetDisplay Setting x
move y MsgPTarget
j waitSend
l x TTargetDial Setting # tTarget
breq x tTarget 6
move tTarget x
mul x x 20
s TTargetDisplay Setting x
move y MsgTTarget
j waitSend
l x StartButton Activate # start
```

```
l x db Setting
breq x MsgReadyValue -2
s db Setting MsgReadyMsg
j ra

calculateTargetMoles:
mul nT pT FurnaceVolume # nT ...
div nT nT R
div nT nT tT # ... = (pT*vT)/(R*tT)
j ra
```

```
breq x start 5
move start x
breqz start 3 # skip if changed to 0
move y MsgStart
s GasControlIC Setting y
j ra

waitSend:
s GasControlIC Setting y
s db Setting 1
yield
l y GasControlIC Setting
brne y MsgReadyValue -2
s GasControlIC Setting x
s db Setting 0
j ra
```

4.3 Calculating n_R, t_I, n_I, and n_H

Once the start message has been sent, we carry out the entire gas mixing procedure for the current state of F, H and C.

$$n_{R_C} = n_F - \frac{n_T(t_T - t_C)}{t_F - t_C} \leq n_R$$

$$n_{R_H} = n_F - \frac{n_T(t_H - t_T)}{t_H - t_F} \leq n_R$$

$$n_R = \max(n_{R_C}, n_{R_H}, 0)$$

$$n_I = n_T - n_F + n_R$$

$$t_I = \frac{t_T n_T - t_F(n_F - n_R)}{n_I}$$

```
calculate:
sub nRC tT tC
mul nRC nT nRC
sub x tF tC
div nRC nRC x
sub nRC nF nRC # nRC=nF-nT(tT-tC)/(tF-tC) (eq.4)

sub nRH tT tH
mul nRH nT nRH
sub x tH tF
div nRH nRH x
sub nRH nF nRH # nRH=nF-nT(tH-tT)/(tH-tF) (eq.5)

max nR nRC nRH
max nR nR 0 # nR=max(nRC,nRH,0) (eq.6)

sub nI nT nF
add nI nI nR # nI=nT-nF+nR (eq.8)

mul tI tT nT
```

```
sub x nF nR
mul x tF x
div tI tI nI # tI=(tT*nT-tF(nF-nR))/nI (eq.7)
```

$$n_H = \frac{t_I n_I - t_C n_I}{t_H - t_C}$$

```
mul nH tI nI
mul x tC nI
sub nH nH x
sub x tH tC
div nH nH x # nH=(tI*nI-tC*nI)/(tH-tC) (eq.10)
```

$$n_C = n_I - n_H$$

```
sub nC tI nH (eq.9)
```

4.4 Removing n_R, mixing, and adding I

If $n_R > 0$, we empty the furnce into I and use the dump pump to remove n_R moles. Using the I pipe analyzers, we calculate $n_{I_0} - n_R$ (where in this context n_{I_0} is the moles of furnace gas now in I initially, and n_I at a discrete time) and can write the dump pump setting optimally every tick as to not remove too much, given:

$$R\,(\text{setting}) = v_I \left(1 - \frac{n_I}{n_{I_0}}\right). \tag{11}$$

Note that the initial moles n_{M_0} of a volume M is assumed to be greater than what is needed in the mix n_M. If this isn't true, then the following code with become stuck in an infinite loop!

```
beqz nR mix # skip if nothing to remove
dump:
jal emptyFurnace
l y IAnalyzer TotalMoles
sub y y nR # target moles I
dumpLoop:
yield
move i d(IAnalyzer)
move v vI
jal setPump
bgtz x dumpLoop # loop if setting > 0
s IPump On 0
# ...
setPump: # y=target, v=volume
l x dr(i) TotalMoles # discrete moles
div x y x
sub x 1 x
mul x v x # x=(r=v(1-nt/n)) # (eq.11)
add i i 1 # move to pump
s dr(i) Setting x
s dr(i) On 1
add i i 1 # move to next analyzer (if any)
```

The mixing algorithm diverges depending on whether we chose to use a single gas mixer, or two volume pumps. One gas mixer is simple, but both less accurate and much slower than using two simultaneous volume pumps in the same manner as what we just performed for the furnace gas dumping algorithm.

(One gas mixer)

With H as input one, the mixer ratio is given by a fraction R_H in terms of n_H, n_C, n_{H_0} and n_{C_0}, where n_{M_0} is the initial amount of gas in pipe network M.

$$R_H = \frac{r_H}{r_H + r_C}, \quad r_M = \frac{n_M}{n_{M_0}}. \tag{12}$$

```
mix:
l x HAnalyzer TotalMoles
l y CAnalyzer TotalMoles
div x nH x # x=(rH=nH / total H moles)
div y nC y # y=(rC=nC / total C moles)
add y x y # y=rH+rC
div x x y # x=rH/(rH+rC) # (eq.12)
s Mixer Setting x
yield
s Mixer On 1
l x IAnalyzer TotalMoles
brlt x nI -3
s Mixer On 0
```

(Two volume pumps)

We use the same algorithm as dumping n_R, but simultaneously with sources H and C.

```
mix:
l x HAnalyzer TotalMoles
sub nH x nH # target moles H
l x CAnalyzer TotalMoles
sub nC x nC # target moles C
mixLoop:
yield
move i d(HAnalyzer)
move v vH
move y nH
jal setPump
sgtz z # H setting > 0
move v vC
move y nC
jal setPump
sgtz x # C setting > 0
and x x z # either > 0
bnez x dumpLoop # loop if either > 0
s HPump On 0
s CPump On 0
```

All that is left is to add the I mixture into the furnace, and we achieve p_T and t_T.

```
jal fillFurnace
s db Setting 0 # since db=3 all this time
j main
```